*В.Дьяконов, В.Круглов*

# МАТЕМАТИЧЕСКИЕ ПАКЕТЫ РАСШИРЕНИЯ MATLAB. СПЕЦИАЛЬНЫЙ СПРАВОЧНИК

Книга известных специалистов, педагогов с большим стажем в справочной форме описывает ряд пакетов расширения MATLAB. Впервые в одной книге описаны пакеты Notebooks, Symbolic, Simulink, Neural Net, Fuzzy Logic Toolbox, Optimization Toolbox и Statistics Toolbox. Эти пакеты существенно расширяют возможности системы MATLAB при выполнении массовых математических вычислений и моделировании сложных объектов и систем — как обычных, так и использующих новые подходы: нечеткую логику и нейронные сети. Особое внимание уделено визуализации математических вычислений и их теоретическому обоснованию.

Книга рекомендуется широкому кругу читателей — студентам университетов и вузов, инженерам, научным работникам.

Краткое содержание



Содержание

Алфавитный указатель

# Введение

Система MATLAB (матричная лаборатория) была создана специалистами фирмы MathWorks, Inc. как язык программирования *высокого уровня* для технических вычислений. Она вобрала в себя не только передовой опыт развития и компьютерной реализации численных методов, накопленный за последние три десятилетия, но и весь опыт становления математики за всю историю человечества.

Одним из самых важных достоинств системы MATLAB является возможность ее расширения с целью решения новых научно-технических задач. Это достигается прежде всего созданием целого ряда пакетов расширения системы, охватывающих многие новые и практически полезные направления компьютерной математики. Однако если по самой системе MATLAB уже опубликован ряд известных книг, то книг по пакетам расширения все еще очень мало. Этот серьезный пробел и призвана в некоторой степени восполнить данная книга.

Данная книга является вполне самостоятельной справочной книгой по системе MATLAB и ее ключевым пакетам расширения, имеющим ярко выраженную математическую направленность. Это достигнуто благодаря введению в данную книгу главы 1 с кратким обзором основных возможностей системы MATLAB 5.3.1.

Одними из главных трудностей при подготовке данной книги были отбор и систематизация огромного по объему материала по системе MATLAB 5.3.1. Фирменное описание только собственно системы MATLAB, представленное в виде PDF-файлов, составляет около 4000 страниц, а об объеме описания некоторых пакетов расширения дают представление следующие данные:

| **Пакет расширения** | **Объем описания, с** |
|---|---|
| Simulink | 815 |
| Communications | 824 |
| Data Acquisition | 542 |
| DSP Blockset | 710 |
| Financial | 431 |
| Mapping | 1279 |

| Пакет расширения | Объем описания, с |
|---|---|
| μ-Analysis and Synthesis | 740 |
| Neural Networks | 742 |
| Report Generator | 304 |
| Real Time Workshop | 799 |
| Signal Processing | 780 |
| Statistics | 420 |
| Symbolic Math | 300 |
| Wavelet | 626 |

Хотя выше указана лишь небольшая часть наиболее известных пакетов расширения MATLAB 5.3.1, объем их описания уже превысил 10 тысяч страниц, что эквивалентно примерно 20 книгам, по объему подобным данной книге. Поэтому материал настоящей книги никоим образом не является прямым переводом фирменных описаний. Это авторский материал, содержащий тщательно отобранные и практически не повторяющиеся сведения об основных пакетах расширения системы MATLAB 5.3.1. По рубрикации и характеру описания он существенно отличается от фирменных руководств и дает описание ключевых пакетов расширения этой системы.

В книгу включены только те пакеты расширения, которые имеют ярко выраженную математическую направленность и служат как для расширения общих математических возможностей системы MATLAB (символьные вычисления, оптимизация и статистика), так и для реализации новых направлений компьютерной математики, таких как математическое моделирование, построение нейронных сетей и нечеткая логика. Пакеты, реализующие обработку данных, сигналов и изображений, а также пакеты, посвященные моделированию отдельных систем и устройств, по мнению авторов, должны быть описаны в отдельных книгах, посвященных этим вопросам.

# Предупреждения

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

Работа с такой мощной математической системой, как MATLAB, и тем более с ее пакетами расширения 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 исчерпывающим справочником по математике, численным методам вычислений и даже по самой системе. Она лишь излагает избранный материал по системе и ее ключевым пакетам расширения с математической направленностью. Авторы берут на себя ответственность за то, какие именно из пакетов расширения можно отнести к этой категории.

Многие математические методы в книге описаны кратко, а некоторые (в основном узкоспециализированные) лишь упоминаются по названию. Исключение из этого правила сделано только при описании малоизвестных у нас областей компьютерной математики, таких как нечеткая логика и нейронные сети. Здесь, несмотря на справочный характер книги, уделено достаточно внимания описанию теоретических положений соответствующих направлений, ибо без этого возможности соответствующих пакетов расширения будут просто непонятны большинству читателей.

Мы вынуждены предупредить читателя, что авторы и издательство не несут никакой ответственности за ошибки пользователей (особенно учащихся) при освоении системы и за моральный или даже экономический ущерб, который может иметь место вследствие ошибок и неудачного выбора математической системы для обучения и применения при решении конкретных задач пользователя.

Сказанное ни в коей мере не означает, что в данной книге или в системе MATLAB 5.3.1 заведомо имеются серьезные ошибки и недочеты. Просто такое предупреждение отвечает нормам современного юридического права в отношении сложных программных продуктов 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 (СГПУ) при поддержке ЗАО «Смоленский телепорт» (www.keytown.com). В ней частично отражены материалы работ одного из авторов (Дьяконова В. П.) по гранту Министерства образования РФ («Применение современных систем компьютерной математики в решении фундаментальных задач естествознания») и по гранту Международной соросовской программы образования в области точных наук (ISSEP, грант соросовского профессора № p99-962 в области математики).

Авторы благодарят генерального директора корпорации SoftLine (Россия) Игоря Боровикова, сотрудника этой фирмы Бориса Манзона и представителя фирмы MathWorks (создавшей систему MATLAB) Наоми Фернандес (Naomy Fernandes) за внимание к своей работе, ее поддержку и предоставление легальной системы MATLAB 5.3.1 для подготовки данной книги.

Авторы благодарят также генерального директора ЗАО «Смоленский телепорт» Григория Рухамина за предоставление услуг спутникового Интернета в ходе работы над книгой, что позволило посредством прямой оперативной связи с сайтом фирмы MathWorks, Inc. быть в курсе обновлений системы MATLAB и использовать самую свежую информацию.

С одним из авторов данной книги (В. П. Дьяконовым) можно связаться по электронной почте (dyak@keytown.com). Авторы заранее выражают признательность всем читателям, которые готовы сообщить свое мнение о данной книге. Кроме электронной почты замечания можно направлять по адресу: 214000, г. Смоленск, ул. Пржевальского, 4, СГПУ. Вы можете отправить свои письма и по адресу издательства, выпустившего книгу.

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

# От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на Web-сайте издательства http://www.piter.com.

# Глава 1
# Основы работы с системой MATLAB

Пакеты расширения системы MATLAB предполагают работу в ее среде. Цель первой главы — дать читателю начальное представление о системе MATLAB. Полное описание этой системы, по стилю близкое к стилю данной книги, дано в следующих книгах:

- Дьяконов В. П. MATLAB: учебный курс. — СПб: Питер, 2001.
- Дьяконов В. П., Абраменкова И. В. MATLAB 5. Система символьной математики. — М.: Нолидж, 1999.

Однако, чтобы сделать данную книгу полностью самостоятельной, ее первую главу мы начнем с краткого описания основ работы с системой MATLAB.

## Ориентация на матричные операции

Система MATLAB выполняет операции над векторами и матрицами. Одномерный массив называют *вектором*, а двумерный — *матрицей*:

| [1 2 3 4] или [1, 2, 3, 4] | $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 8 & 7 & 6 \end{bmatrix}$ | $\begin{bmatrix} a & a+b & a+b/c \\ x & yx & z \\ 1 & 2 & 3 \end{bmatrix}$ |
|---|---|---|
| Векторы из 4 элементов | Матрица размером 3×4 | Матрица с элементами разного типа |

Массивы в общем случае характеризуются размерностью и размером. *Размерность* массива определяет его структурную организацию в виде одной строки или одного столбца (размерность 1), страницы (размерность 2), куба (размерность 3) и т. д. MATLAB допускает задание и использование многомерных массивов ряда типов, в том числе массивов ячеек и записей.

*Размер* вектора — это число его элементов, а размер матрицы определяется числом ее строк $m$ и столбцов $n$. Обычно размер матрицы указывают как $m \times n$. Матрица называется квадратной, если $m = n$, то есть число строк матрицы равно числу ее столбцов. Многие элемен-

ты *разреженных матриц* — нули. Поэтому для эффективной работы с такими матрицами имеется ряд специальных функций.

Векторы и матрицы могут иметь имена, например, **V** — вектор или **M** — матрица. В данной книге имена векторов и матриц набираются **полужирным** шрифтом. Элементы векторов и матриц рассматриваются как *индексированные переменные*. Например, $\mathbf{V}_2$ — второй элемент вектора **V**; $\mathbf{M}_{2,3}$ — третий элемент матрицы **M**, расположенный во второй строке.

Интересно отметить, что даже обычные числа и переменные в MATLAB рассматриваются как матрицы размером 1×1, что дает единообразные формы и методы проведения операций над обычными числами и массивами. Это также означает, что большинство вычислительных функций могут работать с аргументами в виде векторов и матриц, вычисляя значения для каждого их элемента.

## Файловая система MATLAB

Система MATLAB состоит из многих тысяч файлов, находящихся в множестве папок. Файл — это некоторая совокупность данных в широком понимании, хранящихся в памяти компьютера (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 др. Исполняемые файлы имеют расширение .exe.

Особое значение имеет директория MATLAB/TOOLBOX/MATLAB. В ней содержится *Toolbox* — набор m-файлов стандартного расширения системы. 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 — работа со справкой, управление окном MATLAB, взаимодействие с операционной системой и т. д.

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 дескрипторной графики;
- uitools — графика пользовательского интерфейса.

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

# Запуск MATLAB

В этой книге предполагается, что MATLAB инсталлирован и используется в среде операционных систем Windows 95/98. Установка и запуск системы MATLAB 5.3.1 производятся, как обычно для любого приложения операционной системы Windows 95/98.

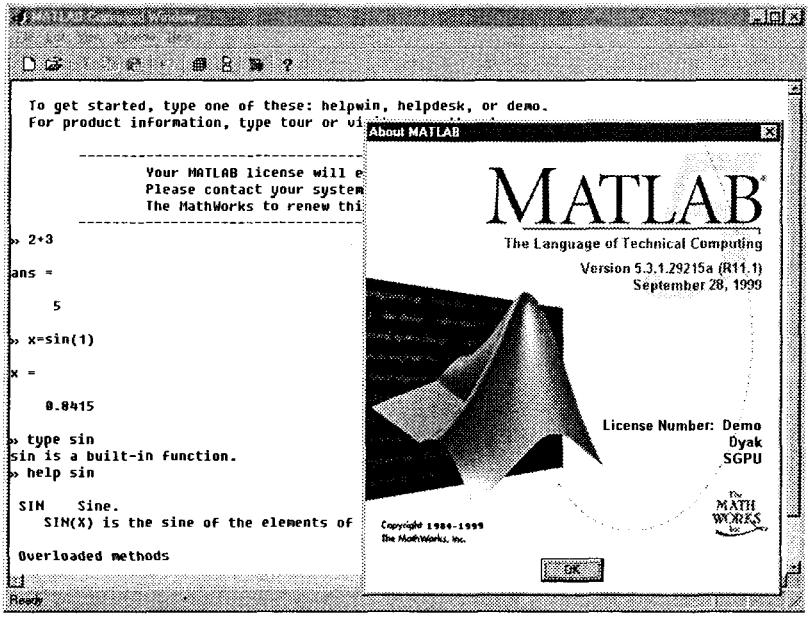


**Рис. 1.1.** Окно системы MATLAB после запуска и выполнения простых вычислений

После запуска одиночным (в Windows 98) или двойным (в Windows 95) щелчком на ярлыке MATLAB появляется основное окно системы MATLAB, показанное на рис. 1.1. Оно имеет обычные органы управления для расширения, перемещения и закрытия.

Система сразу же готова к проведению вычислений в *командном режиме*, присущем самым первым ее реализациям для MS-DOS. Для уточнения версии системы следует вывести окно с информацией о системе (команда About MATLAB в меню Help). Это окно представлено на рис. 1.1 справа.

## Операции строчного редактирования

При работе с MATLAB в командном режиме 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 и Ctrl+n — перелистывание предыдущих команд вверх или вниз для подстановки в строку ввода;
- Del или Ctrl+d — стирание символа справа от курсора;
- Backspace 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;
- Esc — очистка строки ввода.

Эти возможности кажутся примитивными, но позволяют пользователю быстро работать в стиле первых версий MATLAB для MS-DOS.

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.

## Команды управления окном

Полезно сразу усвоить некоторые команды управления окном командного режима:

- clc — очищает экран и размещает курсор в левом верхнем углу пустого экрана;
- home — возвращает курсор в левый верхний угол окна;

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

В версии MATLAB 5.3.1 обе команды, `clc` и `home`, действуют аналогично — очищают экран и помещают курсор в левый верхний угол окна командного режима работы. Команды `echo` позволяют включать или выключать отображение текстов m-файлов при каждом обращении к ним. Как правило, отображение текста файлов сильно загромождает экран и часто не является необходимым. При больших размерах файлов начало их текста (листинга) убегает далеко за пределы области просмотра (текущего окна командного режима). Поэтому для просмотра длинных листингов файлов полезно включить постраничный вывод командой `more on`.

## MATLAB как мощный калькулятор

Система MATLAB создана таким образом, что любые (подчас весьма сложные) вычисления можно выполнять в режиме *прямых вычислений*, то есть без подготовки программы. Это превращает MATLAB в необычайно мощный калькулятор, который способен производить не только обычные для калькуляторов вычисления, 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 и происходит по правилу «задал вопрос, получил ответ». Пользователь набирает на клавиатуре вычисляемое выражение, редактирует его (если нужно) в командной строке и завершает ввод

нажатием клавиши Enter. В качестве примера на рис. 1.1 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».

Попробуем выполнить некоторые простые вычисления:

```
» 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
```

В такой форме будет приводиться большая часть примеров в данной книге. Исключения относятся к примерам, в которых желательно видеть на экране расчетную и графическую части одновременно.

Обратите внимание на форму ответов при выполнении простых операций без указания переменной, которой присваивается результат

операции. В таких случаях MATLAB сам назначает переменную ans, которой присваивается результат и значение которой затем выводится.

## 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, записанных в строке MATLAB:

```
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.

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 многоточие ... (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. Максимальное число символов в одной строке — 4096.

## Типы данных системы MATLAB

### Числа целые и вещественные

Важное значение имеют типы данных, с которыми работает система MATLAB. Числа в MATLAB представляются в обычном виде целых, дробных и вещественных 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.

## Форматы чисел

По умолчанию MATLAB выдает числовые результаты в *нормализованной форме* с четырьмя цифрами после десятичной точки и одной до нее. Операции над числами выполняются в формате, который принято называть форматом чисел *с двойной точностью*. Для установки формата представления чисел используется команда

```
» format name
```

где `name` — имя формата. 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.

## Числа комплексные

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

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

Функция `real(z)` возвращает действительную часть комплексного числа $\mathrm{Re}(z)$, а функция `imag(z)` — мнимую, $\mathrm{Im}(z)$.

## Константы и системные переменные

*Константа* — это предварительно определенное числовое или символьное значение, представленное уникальным именем. Числа (например, 1, −2 и 1,23) являются безымянными *числовыми константами*.

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

- `i` или `j` — мнимая единица (корень квадратный из −1);
- `pi` — число π = 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 *нечисловой* характер данных (Not-a-Number).

Вот примеры применения системных переменных:

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

## Строки и текстовые комментарии

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

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

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

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 символа %, например:

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

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

```
» help Имя_файла
```

Считается правилом хорошего тона вводить в m-файлы достаточно подробные текстовые комментарии.

## Переменные и присваивание им значений

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

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

```
Имя_переменной = Выражение
```

Типы переменных заранее не декларируются. Они определяются выражением, значение которого присваивается переменной. Так, если это выражение — вектор или матрица, то переменная будет векторной или матричной. Примеры присваивания переменным значений были приведены выше.

*Имя переменной* (ее *идентификатор*) может содержать сколько угодно символов, но запоминается и идентифицируется только 31 начальный символ. Имя любой переменной не должно совпадать с именами других переменных, функций и процедур системы, то есть оно должно быть уникальным. Имя должно начинаться с буквы, может содержать цифры и символ подчеркивания _. Недопустимо включать в имена переменных пробелы и специальные знаки, например

+, -, *, / и т. д., поскольку в этом случае правильная интерпретация выражений становится невозможной.

Желательно использовать для обозначений переменных содержательные имена, 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'.

## Уничтожение определений переменных

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 нескольких переменных.

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.

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

## 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 *поэлементное* умножение и деление массивов.

Следующие примеры поясняют сказанное на примере операций с векторами:

```
» 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          .^
....
```

*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 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, различающихся списком 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 — с помощью команды `help specfun`. Функции могут быть *встроенными* (внутренними) и *внешними*, или *m-функциями*. Так, встроенными являются наиболее распространенные элементарные функции, например `sin(x)` и `exp(y)`, тогда как функция `sinh(x)` является внешней функцией. Внешние функции содержат свои определения в m-файлах. Встроенные функции хранятся в откомпилированном ядре системы MATLAB, в силу чего они выполняются предельно быстро.

## Применение оператора : (двоеточие)

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

```
Начальное_значение:Шаг:Конечное_значение
```

Данная конструкция порождает 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. 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
```

Переменные с множеством упорядоченных значений принято называть *ранжированными*.

Как уже 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**.

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

```
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.

## 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 примеров. Введем, к примеру, ошибочное выражение

```
» sqr(2)
```

и нажмем клавишу Enter. Система сообщит об ошибке:

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

Это сообщение говорит о том, что не определена переменная или функция, и указывает, какая именно — sqr. В данном случае, разумеется, можно просто набрать правильное выражение. Однако в случае громоздкого выражения лучше воспользоваться редактором. Для этого достаточно нажать клавишу ↓ для перелистывания 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 следующий 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 m-функция. Зато далее мы видим, что функция с именем `sinh` есть в составе функций системы MATLAB — она задана в виде m-функции.

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

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

# Простейшие приемы работы с векторами и матрицами

## Особенности задания векторов и матриц

Для задания вектора надо перечислить значения его элементов в квадратных скобках, разделяя их пробелами или запятыми (см. примеры выше). Задание матрицы требует указания различных строк. Для различения строк используется знак ; (точка с запятой). Этот же знак 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
```

## Доступ к отдельным элементам

Для указания отдельного элемента вектора или матрицы их рассматривают как *индексированные переменные*. Для этого 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:

$$\begin{bmatrix} 1+5i & 2+6i \\ 3+7i & 4+8i \end{bmatrix}$$

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

Имеется также ряд особых функций для задания векторов и матриц. Например, функция `magic(n)` задает *магическую матрицу* размером *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
```

### 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
```

## Сессия MATLAB

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 перемен-

ных и функций, расположенные в рабочей области памяти (но не саму сессию), можно записать на диск (в файл 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. Рассмотрим эти команды более подробно.

## Сохранение рабочей области сессии

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, то есть 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 — записывается только значение переменной X;
- save fname X Y Z — записываются значения переменных X, Y и Z.

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

- -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 4.
- -append — добавление в существующий mat-файл.

Возможно задание записи в формате функции, например:

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

В этом случае имена файлов и переменных задаются строковыми константами.

## Ведение дневника

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, то для этого служит команда diary:

- 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 функции diary('file'), где строка 'file' задает имя файла. Следующий пример поясняет технику применения команды diary:

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

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

```
1+2

ans =

     3
diary off
sin(1)

ans =
```

```
    0.8415

diary off
```

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

```
» myfile
??? ans =
          |
Missing variable or function.

Error in ==> C:\MATLAB\bin\myfile.m
On line 3  ==> ans =
```

Обычно это приводит к ошибкам, так как данный файл — это просто текстовая запись команд и результатов их выполнения, не проверяемая на корректность и содержащая ряд строк, ошибочных с позиций синтаксиса языка программирования MATLAB (например, выражение `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.

## Загрузка рабочей области сессии

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

- `load fname …` — загрузка ранее сохраненных в файле `fname.mat` определений со спецификациями, помещаемыми на месте многоточия, и подобными описанным для команды `save` (включая

ключ -mat для загрузки файлов с расширением .mat обычного бинарного формата, используемого по умолчанию);

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

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

### Завершение работы с системой

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

## Работа со справочной системой MATLAB

При изучении математических систем, особенно самостоятельном, трудно переоценить роль справочной системы. В MATLAB используется развитая (многоуровневая) справочная система, использующая все известные приемы получения справочных сведений — от прямого запроса в командной строке до использования гипертекстовых справочных подсистем и документов в формате PDF.

### Вызов списка примеров интерактивной справки

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

```
» help
```

которая выводит весь список папок (каталогов), содержащих m-файлы с определениями операторов, функций и иных объектов. Этот внушительный список дает наглядное представление о пакетах прикладных программ, расширяющих возможности системы MATLAB и содержащих массу серьезных примеров применения системы.

## Справка по конкретному объекту

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 m-файл отсутствует. Если ввести имя верно, получится вот что:

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

Теперь полученное сообщение будет содержать информацию о функции **sinh**. Хотя имя функции в MATLAB задается малыми (строчными) буквами, в сообщениях справочной системы имена функций и команд выделяются большими (прописными) буквами.

## Справка по определенной группе объектов

Пользователя системы MATLAB часто интересует набор функций, команд или иных понятий, относящихся к определенной группе объектов. 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.

## Справка по ключевому слову

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

```
lookfor Ключевое слово
```

или

```
lookfor 'Ключевые слова'
```

В первом случае ищутся все m-файлы, в заголовках которых встречается заданное ключевое слово, и заголовки обнаруженных файлов выводятся на экран. Следует отметить, что широкий поиск по одному ключевому слову может подчас привести к выводу многих десятков определений и длиться 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 набором пакетов прикладных программ (расширений) поставляется версия системы MATLAB.

## 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;

- `help script` — выводит сообщение о назначении m-файлов сценариев (Script-файлов);
- `help function` — выводит сообщение о назначении и структуре m-файлов функций;
- `info` — выводит информацию о фирме MathWorks с указанием адресов электронной почты;
- `subscribe` — позволяет создать файл с бланком регистрации;
- `ver` — выводит информацию о версиях установленной системы MATLAB и ее компонентов;
- `version` — выводит краткую информацию об установленной версии MATLAB;
- `what` — выводит имена файлов текущего каталога;
- `what name` — выводит имена файлов каталога, заданного именем `name`;
- `whatsnew name` — выводит на экран содержимое файлов `readme` заданного именем `name` класса для знакомства с последними изменениями в системе и в пакетах прикладных программ;
- `which name` — выводит путь доступа к функции с данным именем.

Как правило, эти команды выводят довольно обширные сообщения. Ниже показаны примеры применения отдельных команд этой группы (для команды `ver` дана только часть списка компонентов `Toolbox`):

```
»computer
ans =
PCWIN
» version
ans =
5.3.1.29215a (R11.1)
» ver
-------------------------------------------------
MATLAB Version 5.3.1.29215a (R11.1) on PCWIN
MATLAB License Number: Demo
-------------------------------------------------
MATLAB Toolbox                Version 5.3.1 (R11.1)  08-Sep-1999
xPC Target                    Version 1.0   (R11.1)  10-Sep-1999
Quantized Filtering Toolbox   Version 1.0   (R11.1)  01-Sep-1999
Financial Time Series Toolbox Version 1.0   (R11.1)  01-Apr-1999
...
Stateflow                     Version 2.0.1 (R11)    30-Aug-1999
Simulink                      Version 3.0.1 (R11.1)  10-Sep-1999
MATLAB Tour                   Version 1.2   (R11)    04-Sep-1998
```

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

## Вызов списка демонстрационных примеров

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
```

Мы настоятельно рекомендуем пользователям системы MATLAB просмотреть с десяток примеров из интересующих их областей. Здесь же мы ограничимся одним примером — тестом на быстродействие компьютера.

## Пример — тест на быстродействие компьютера

Исполнив команду bench, можно наблюдать исполнение комплексного теста по оценке быстродействия компьютера. Его итоги представляются в виде гистограммы и таблицы, которые показаны на рис. 1.2.




**Рис. 1.2.** Результаты тестирования компьютера на быстродействие

По этому тесту можно судить о быстродействии компьютера пользователя в сравнении с быстродействием ряда типовых моделей компьютеров. Данный пример показывает также тщательную отработку деталей интерфейса примеров, характерную для системы MATLAB.

### Просмотр текстов примеров и m-файлов

Хотя само по себе наблюдение за тем, как MATLAB расправляется со сложными примерами и задачами, довольно поучительно, жаждущие применить систему на деле пользователи, безусловно, захотят узнать, как же конкретно реализовано решение той или иной задачи. Для этого вам достаточно просмотреть соответствующий демонстрационный (или любой другой) m-файл. Это можно сделать с помощью любого текстового редактора, редактора/отладчика m-файлов, встроенного в систему (он будет описан в этой главе далее), или с помощью команды `type Имя_m-файла`. Надо отметить, что текст m-файла — типичная программа на языке программирования системы MATLAB.

### Запуск справочной системы Help Desk

В MATLAB имеется мощная справочная подсистема 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 броузер Интернета и при этом имеется возможность обновления документации.

Для запуска справочной подсистемы Help Desk следует использовать одноименную команду меню Help. При этом запустится броузер и откроется окно подсистемы Help Desk, показанное на рис. 1.3.

Каждый раздел справочной системы представлен в виде гипертекстовой ссылки (это подчеркнутые снизу надписи), активизация которой приводит к переходу на соответствующую HTML-страницу.

### Справка по функциям и полнотекстовый обзор

Будучи крупной математической системой, MATLAB имеет многие сотни функций, запомнить свойства которых трудно даже пользователю-профессионалу. Да и нужно ли? Справочная система MATLAB позволяет найти информацию по нужной функции в считанные секунды. Для этого в левой части окна рис. 1.3 надо найти раздел Go to MATLAB Function:. В этом окне есть поле задания нужной функции — перед кнопкой Go. Достаточно ввести имя функции и нажать эту кнопку. При выдаче данных о функциях используется справоч-

ник по ним (Function Reference) и обеспечивается автоматический поиск по нужной функции. Гипертекстовая ссылка Examples открывает окно с примерами использования данной функции. На рис. 1.4 показано окно с примером использования функции синуса с построением графика синусоиды.

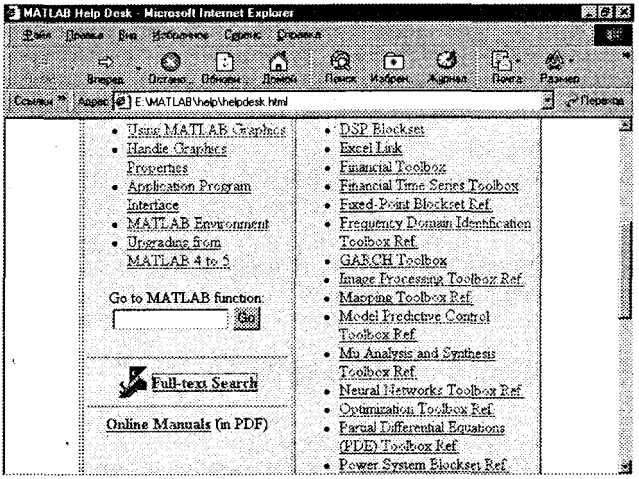


**Рис. 1.3.** Основное окно справочной подсистемы Help Desk

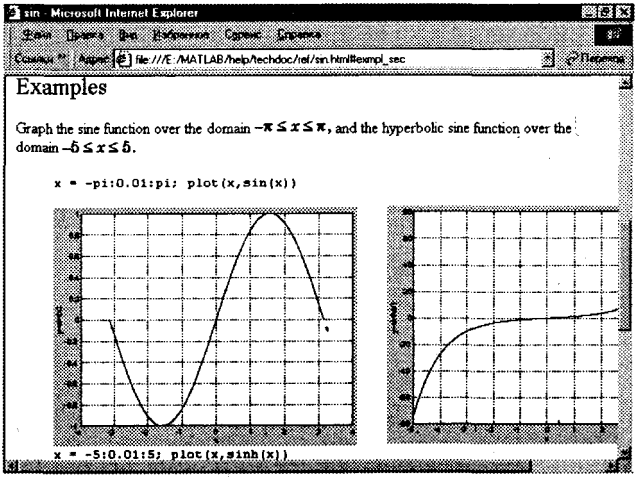


**Рис. 1.4.** Окно с примерами использования функций sin и sinh

Большие удобства в работе со справочной системой дает полнотекстовый поиск (Full-text Search). В этом случае поиск ведется по всему тексту справочной системы. Соответствующую гиперссылку можно найти в перечне документов (см. рис. 1.3). Если ее активизировать, появится окно полнотекстового поиска. В поле Search этого окна можно ввести любую фразу, по которой будет проводиться поиск.

### Просмотр документации в формате PDF

Справочная подсистема Help Desk содержит гиперссылку Online Manuals (in PDF), предоставляющую доступ к электронной справочной документации, которая поставляется в виде файлов PDF. Среди документов крупные (сотни страниц) руководства по функциям системы, языку ее программирования, графическим функциям и т. д.

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 превышает 200 МБайт, что делает документацию труднообозримой (см. данные, приведенные во Введении).

## Демонстрационные примеры

### Команда demo

В меню Help имеется команда Examples and Demos, содержащая 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. 1.5. Это же окно можно вызвать выполнением команды demo в командном режиме.

В этом окне имеются 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 (щелчком мыши), следует затем выбрать нужный пример.

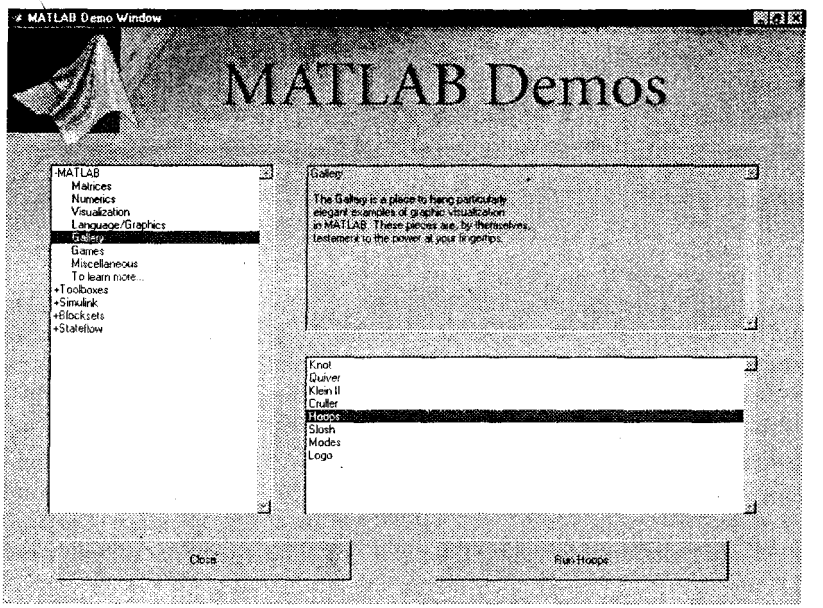


**Рис. 1.5.** Окно демонстрационных примеров

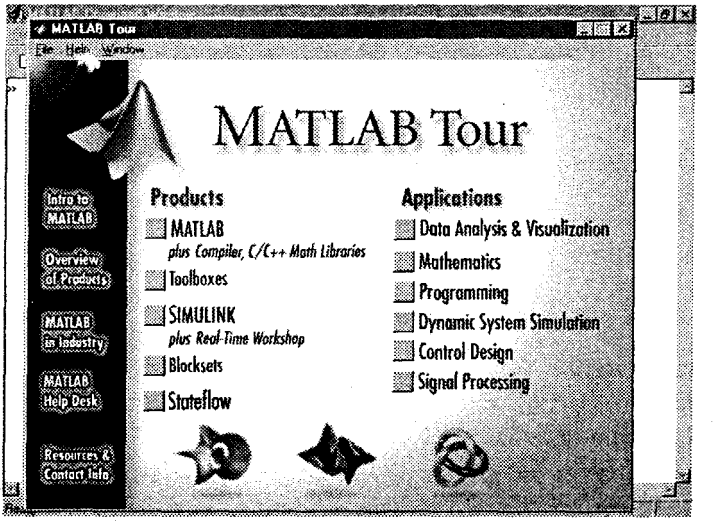


**Рис. 1.6.** Окно ознакомительной системы Tour

### Ознакомительная система MATLAB 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, в котором имеется перечень ее разделов (рис. 1.6).

Это окно также дает доступ к множеству примеров применения системы MATLAB, знакомство с которыми полезно для ее изучения.

## Пользовательский интерфейс

В новой версии MATLAB в полной мере сохранен командный интерактивный режим работы, который остается одним из наиболее удобных и апробированных методов работы с системой. Имеются и типовые средства Windows 95/98 — меню и панель инструментов.

Для редактирования и отладки m-файлов MATLAB имеет встроенный современный редактор, интерфейс которого выполнен в лучших традициях Windows-приложений. В том же стиле выполнены окно просмотра ресурсов памяти, окно просмотра путей файловой системы, справочник по возможностям системы и демонстрационные программы.

### Панель инструментов

На рис. 1.7 приведена часть окна системы MATLAB, содержащая главное меню и панель инструментов.

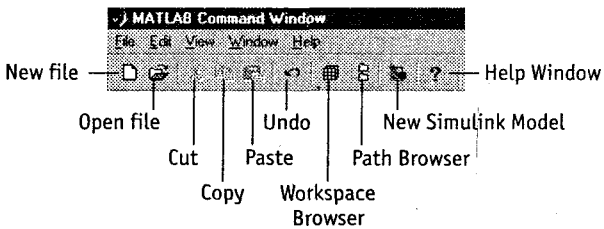


**Рис. 1.7.** Часть окна системы MATLAB с главным меню и панелью инструментов

Прежде всего перечислим назначение всех кнопок панели инструментов:

1. New file — выводит пустое окно редактора m-файлов.
2. Open file — открывает окно для загрузки m-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 Simulink.
10. Help Window — открывает окно справки по системе.

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

## Кнопки работы с файлами

Кнопка New file открывает окно редактора/отладчика m-файлов. Это окно показано на рис. 1.8.

По умолчанию файлу дается имя Untitled (безымянный), которое впоследствии (при записи файла) можно изменить на другое, отражающее тему задачи. Это имя отображается в титульной строке окна редактирования m-файла, которое размещается в окне редактора/отладчика, как видно на рис. 1.8. В редакторе/ отладчике можно редактировать несколько m-файлов, и каждый будет находиться в своем окне редактирования, хотя активным может быть только одно окно, которое будет расположено поверх других окон. На рис. 1.8 в окне редактирования нового файла приведен пример простого m-файла, обеспечивающего построение графика синусоидальной функции.

По завершении ввода текста файла он сохраняется на диске под своим текущим именем с помощью кнопки Save панели инструментов редактора/отладчика m-файлов. Можно также использовать команду Save As из меню File окна редактора.

Кнопка Open file служит для загрузки в редактор/отладчик ранее созданных m-файлов, например входящих в Toolbox системы или раз-

работанных пользователем. Она открывает стандартное окно загрузки файлов Windows-приложений. С его помощью можно «пройтись» по всем дискам, папкам и файлам текущей папки и выбрать нужный файл для загрузки.

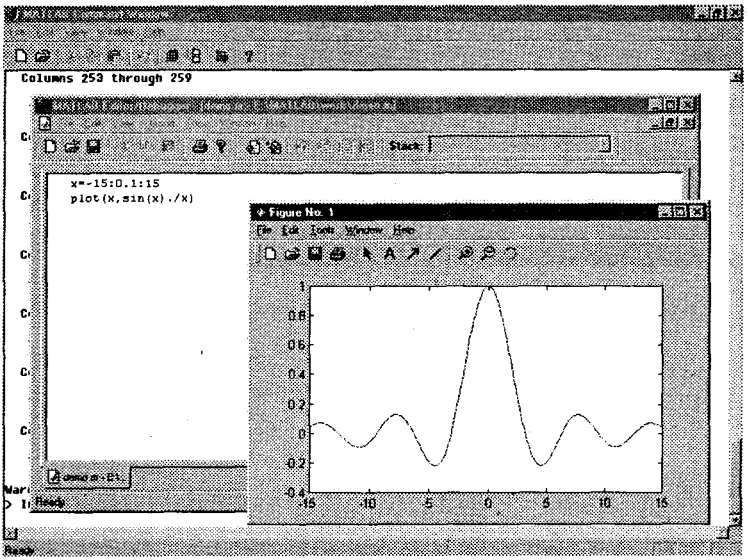


**Рис. 1.8.** Окно редактора/отладчика m-файлов

## Работа с буфером обмена

Кнопки Cut, Copy и Paste реализуют наиболее характерные команды работы с буфером обмена (Clipboard). Первые две операции относятся к выделенным фрагментам сессии или текста m-файлов (если они выполняются в окне редактора/отладчика m-файлов). Для выделения объектов можно использовать мышь, перемещая указатель по тексту при нажатой левой кнопке, или клавиши стрелок в комбинации с клавишей Shift.

Команда Cut осуществляет вырезание выделенного фрагмента и размещение его в буфере. При этом вырезанный фрагмент удаляется из текста документа. Команда Copy просто копирует выделенный фрагмент в буфер, сохраняя его в тексте. Команда Paste вызывает объект из буфера (сохраняя его в нем) и помещает на место в документе, указанное текстовым курсором. Эти команды реализуются как соответствующими кнопками, так и командами меню Edit.

В MATLAB можно использовать контекстно-зависимое меню, появляющееся при нажатии правой кнопки мыши. В этом меню активны только те команды, которые возможны в данный момент и в данной ситуации работы с системой.

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

## Броузер рабочей области

Кнопка Workspace Browser выводит окно специального броузера для просмотра ресурсов рабочей области памяти. Рисунок 1.9 иллюстрирует применение окна просмотра ресурсов при задании нескольких переменных различного типа. Броузер дает наглядную визуализацию содержимого рабочей области.

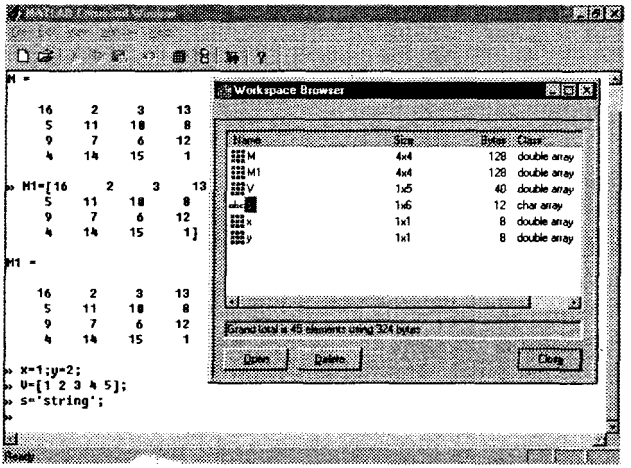


**Рис. 1.9.** Просмотр ресурсов памяти с помощью броузера рабочей области

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 мы-

ши и щелкнуть на кнопке Open. Объект можно открыть и двойным щелчком на его имени в списке. Откроется окно, подобное показанному на рис. 1.10 применительно к матрице M.

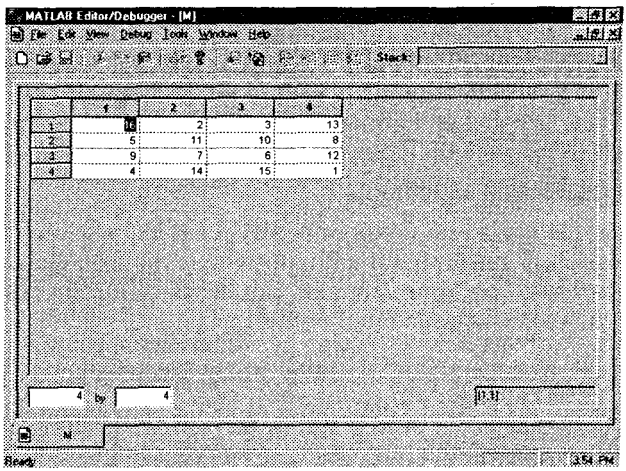


**Рис. 1.10.** Окно просмотра содержимого матрицы

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

## Команды просмотра рабочей области who и whos

Следует отметить, что просмотр рабочей области возможен и в командном режиме без обращения к броузеру 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, просмотр рабочего пространства с помощью броузера Workspace Browser более удобен и нагляден.

## Броузер файловой структуры

Для просмотра файловой структуры MATLAB служит специальный броузер, который запускается с помощью кнопки Path Browser панели инструментов, при этом на экран выводится окно просмотра файловой структуры, показанное на рис. 1.11.

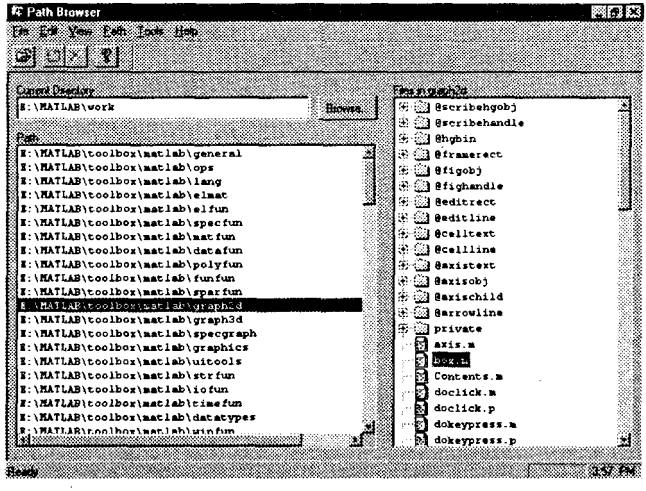


**Рис. 1.11.** Окно броузера файловой структуры

Окно разделено на две части. В левой расположен список папок и путей к ним, а в правой — список папок и файлов для выделенного в левой части каталога. Например, на рис. 1.11 выделен каталог graph2d с папками и m-файлами двумерной графики. Двойной щелчок на папке в правой части открывает ее, а двойной щелчок на файле запускает редактор/отладчик m-файлов с загруженным в него выделенным файлом.

Кнопка Browse позволяет вывести отдельное окно с содержимым выделенной папки. Окно представляет данные о файловой структуре в виде дерева папок и файлов. Каждую папку можно открыть, установив на ней указатель мыши и выполнив двойной щелчок. Открытая папка отображается со знаком «минус», а закрытая — со знаком «плюс».

Таким образом, броузер файловой структуры позволяет детально ознакомиться с файловой системой MATLAB.

Последняя кнопка панели инструментов — Help Window — открывает окно с перечнем разделов справочной системы. Оно выводится также командой Help Window меню Help.

# Меню системы

Меню MATLAB (см. рис. 1.7 сверху) выглядит довольно скромно и содержит всего пять позиций:

- File — работа с файлами;
- Edit — редактирование сессии;
- View — вывод и устранение панели инструментов;
- Windows — установка Windows-свойств окна;
- Help — доступ к справочным подсистемам.

По сравнению с предшествующими версиями добавлена только позиция View, с помощью которой можно убрать единственное «излишество» сверхскромного интерфейса MATLAB — панель инструментов, которая была рассмотрена выше.

## Меню, операции и команды

Открытая позиция строки меню содержит 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, соответствующей выделенному в названии команды символу.

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

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

## Меню File

Меню File содержит внушительный список операций и команд. Оно показано на рис. 1.12.

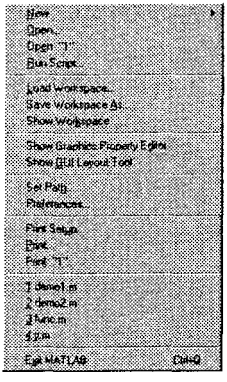

**Рис. 1.12.** Меню файловых операций File

Эти операции и команды разбиты на несколько характерных групп.

Операция New (Создать) после имени имеет знак в виде черного маленького треугольника. Это означает, что она открывает подменю. Оно служит для создания трех типов документов:

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

Операция Open (Открыть файл) открывает стандартное окно загрузки файла.

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. Например, 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 файлов (Script-файлов). Имя файла можно задать в явном виде или же, нажав кнопку Browse, открыть окно поиска файла. Кнопка OK служит для под-

тверждения загрузки файла и его пуска на исполнение, а кнопка Cancel позволяет отказаться от запуска.

## Операции с рабочей областью

Операция Load Workspace открывает обычное окно для загрузки файлов с расширением .mat. Это бинарные файлы, которые загружают в память компьютера определения переменных, векторов, матриц и т. д. Операция Save Workspace As открывает окно для записи рабочего пространства на диск в виде файла с заданным именем.

Команда Show Workspace выводит описанное выше окно просмотра рабочей области (см. рис. 1.9).

## Настройка MATLAB и операция Preferences

Следующие две операции относятся к настройке параметров системы. Операция Set Path выводит окно настройки путей файловой системы. Это окно уже было показано на рис. 1.11. В командном режиме пути файловой системы выводятся с использованием функции path.

Операция Preferences открывает окно настройки параметров системы, показанное на рис. 1.13. Это окно имеет ряд вкладок. В качестве примера на рис. 1.13 приведено данное окно с открытой вкладкой General, которая предназначена для установки ряда опций общего характера.

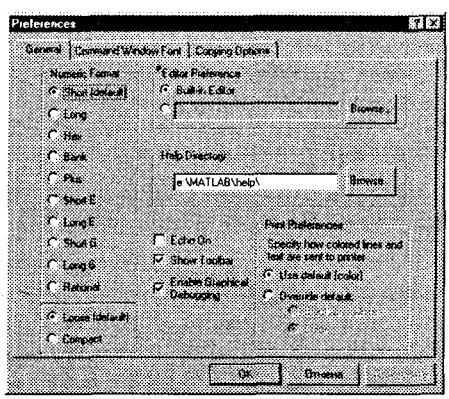


**Рис. 1.13.** Окно Preferences с открытой вкладкой General

Прежде всего следует отметить переключатель Numeric format, который определяет формат представления чисел. Обширный перечень форматов чисел дан в левой части окна.

Следующая вкладка — Command Window font — служит для выбора шрифта, используемого при работе MATLAB в командном режиме. Несмотря на широкие возможности выбора шрифтов, не рекомендуется менять набор, установленный по умолчанию, поскольку он оптимален для большинства пользователей.

Вкладка Copying Options служит для установки опций для работы с буфером обмена. Имеется возможность задания типа сохраняемых данных в виде метафайлов или растровых файлов, а также установки цвета фона при копировании графиков в буфер. Таким образом, окно Preferences дает обширные возможности по настройке системы.

## Операции печати

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

Другая команда — Print — служит для вывода окна печати. Следует отметить, что окно печати — типичное для операционной системы Windows 95/98. Его вид зависит от примененного принтера, точнее, от установленного для него драйвера принтера. Установки окна довольно очевидны, так что более подробно они не описываются.

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

## Меню Edit — средства редактирования документов

Меню Edit (рис. 1.14) содержит операции редактирования, типичные для большинства приложений Windows.

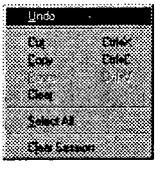


**Рис. 1.14.** Меню Edit

Как видно из рис. 1.14, 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 в текущую позицию курсора;
- Clear — операция очистки выделенной области;
- Select All — выделение всей сессии;
- Clear session — очистка сессии (с сохранением созданных объектов).

Назначение указанных команд и операций уже обсуждалось. Отметим лишь, что команда Clear session очищает окно командного режима работы и помещает курсор в верхний левый угол окна. Однако все определения переменных, сделанные в течение стертых таким образом сессий, сохраняются в памяти компьютера. Для очистки экрана используется также команда clc, вводимая в командном режиме.

### Меню View и Window

Позиции View и Window выглядят как дань моде, отданная разработчиками MATLAB с большой неохотой. Введенное в MATLAB 5.3.1 меню View имеет единственную опцию — Toolbar. Если пометить ее знаком птички, то появляется описанная ранее панель инструментов. Если снять птичку, то это «излишество» исчезает.

Все, что делает меню Window, — представляет окно командного режима MATLAB в стандартном (не полностью открытом) виде. Если открыто несколько окон, данное меню позволяет также выбирать и делать активным одно из них.

## Интерфейс редактора/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 меню File. После этого в окне редактора можно создавать свой файл, пользоваться средствами его отладки и запуска. Для запуска файла его необходимо записать на диск, используя команду Save as в меню File редактора.

На рис. 1.8 уже приводилось окно редактора/отладчика с текстом простого файла в окне редактирования и отладки. Подготовленный текст файла (это простейшая и первая наша программа на языке

программирования MATLAB) надо записать на диск. Для этого используется обычная команда Save As.

После записи файла на диск можно заметить, что команда Run в меню Tools 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 графическом окне и вывод надписи о делении на ноль в ходе вычисления функции $\sin(x)/x$ в командном окне системы (см. рис. 1.8).

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

### Цветовые выделения и синтаксический контроль

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

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

Благодаря цветовым выделениям вероятность синтаксических ошибок резко снижается.

### Понятие о файлах-сценариях и файлах-функциях

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

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

Видимый в окне редактора на рис. 1.8 файл является файлом-сценарием, или Script-файлом. Данный 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*, значения которых могут быть изменены в любой момент сеанса работы и в любом месте программы.

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

```
% Пример задания функции
function z=fun(x,y)
z=x^2+y^2;
return
```

Имя этой функции z, и под таким именем ее надо записать в виде m-файла[1]. (x,y) — список параметров (x и y). Оператор return задает возврат функцией значения в ответ на обращение к ней по имени с указанием фактических параметров[2]. Например, задав z(2,3), получим 13.

Входящие в файл-функцию переменные из списка параметров являются *локальными переменными*, изменение значений которых в теле функции никоим образом не влияет на их значения за пределами функции. Иными словами, локальные переменные могут иметь те же имена, что и глобальные переменные (хотя правила культурного программирования не рекомендуют смешивать имена локальных и глобальных переменных).

## Панель инструментов редактора/отладчика

Редактор имеет свое меню и свою инструментальную панель, которая может перемещаться мышью в любое подходящее место. Внешний вид инструментальной панели показан на рис. 1.15. По стилю данная панель похожа на панель инструментов окна командного режима работы, но имеет несколько иной набор кнопок.

---

[1] Имя m-файла для функции может быть любым, оно не должно обязательно совпадать с именем переменной-результата (тем более что возвращаемых результатов может быть несколько). В данном примере можно, например, сохранить m-файл под именем sumsqr.m и затем вызывать функцию в виде sumsqr(x,y). — *Примеч. ред.*

[2] При отсутствии в теле функции оператора return возврат произойдет по достижении конца файла. Результатом функции всегда является последнее значение, присвоенное выходному параметру (в данном случае — переменной z). Таким образом, использовать return имеет смысл только при необходимости досрочного выхода из функции. — *Примеч. ред.*

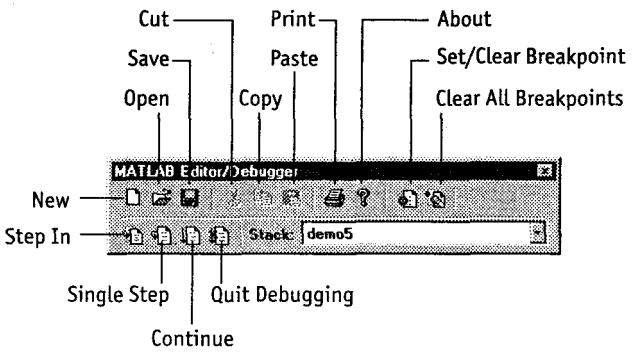


**Рис. 1.15.** Панель инструментов редактора/отладчика m-файлов

Назначение кнопок панели инструментов редактора/отладчика следующее:

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

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

## Работа с точками останова

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

Рассмотрим рис. 1.16, на котором в окне редактора/отладчика видна программная конструкция цикла. Как будет меняться переменная s, значение которой должно давать ряд натуральных чисел? Прежде всего для отладки надо записать программу на диск, а затем установить напротив выражения s=s+1 контрольную точку — она отчетливо видна на рис. 1.16 как красный кружок. Для установки контрольной точки необходимо поместить текстовый курсор в нужное место (напротив указанного выражения) и нажать кнопку Set/Clear Breakpoint.

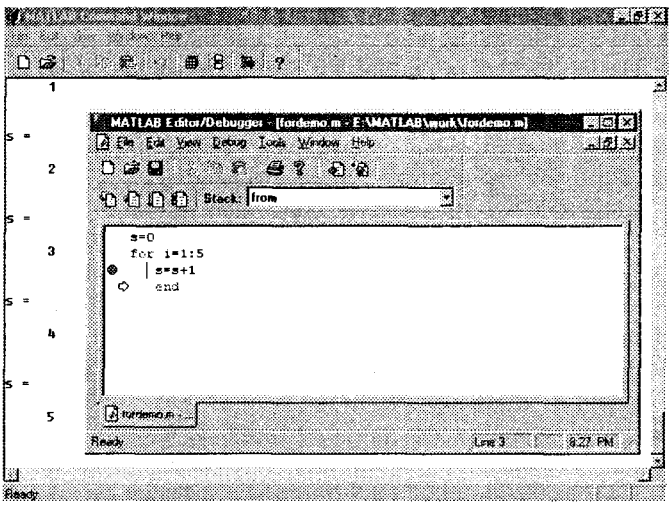


**Рис. 1.16.** Простейший пример применения контрольной точки в программе

Теперь при пуске программы командой Run она будет исполнена до контрольной точки, при этом текущие значения s будут выведены в окне MATLAB (поскольку операторы, вычисляющие значение s, не содержат заключительной точки с запятой). С помощью кнопки Continue (Продолжение) или Single Step (Одиночный шаг) можно выполнить очередной шаг вычислений и т. д. Если отпала необхо-

димость останова в контрольных точках, достаточно кнопкой Clear All Breakpoints удалить разом все контрольные точки. Желтая стрелка указывает, в каком месте программы произошел останов.

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

# Графика системы MATLAB

## Особенности графики системы MATLAB

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 трехмерных графиков с их проекцией на плоскость;
- возможность построения сечений трехмерных фигур и поверхностей плоскостями;
- функциональная многоцветная и полутоновая окраска;
- возможность имитации световых эффектов при освещении фигур точечными источниками 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 только вы не работаете с объектно-ориентированным программированием задач графики. Связано это с тем, что большинство команд графики, ориентированных на конечного пользователя, автоматически устанавливают свойства графических объектов и обеспечивают воспроизведение графики в нужной системе координат, палитре цветов, масштабе и т. д.

На более низком уровне решения задач используется *дескрипторная графика* (Handle Graphics), при которой каждому графическому объекту в соответствие ставится особое описание — *дескриптор*, на который возможны ссылки при использовании графического объекта. Дескрипторная графика позволяет осуществлять визуальное программирование объектов пользовательского интерфейса — управляющих кнопок, текстовых панелей и т. д.

## Интерфейс графических окон

Мы уже видели, что графики MATLAB выводит в отдельных окнах (см. рис. 1.8). Рассмотрим окно графики более подробно (рис. 1.17). Это обычное масштабируемое и перемещаемое окно Windows-приложений.

Меню этого окна похоже на меню окна командного режима работы системы MATLAB. Однако при внимательном просмотре заметен ряд отличий.

Прежде всего в меню Edit окна графики наряду со стандартными операциями работы с буфером есть две новые команды. Одна из них — Copy Figure, которая служит для копирования рисунка в буфер обмена, а другая — Copy Options — служит для настройки опций копирования.

В версии MATLAB 5.3.1 у графических окон появилось новое меню Tools (Инструменты) — на рис. 1.17 оно представлено в открытом состоянии. Это меню содержит ряд команд и опций:

- Show Toolbar — показать или скрыть панель инструментов;
- Enable Plot Editing — включение или выключение редактирования графика;
- Axis Properties — вывести окно свойств осей графика;

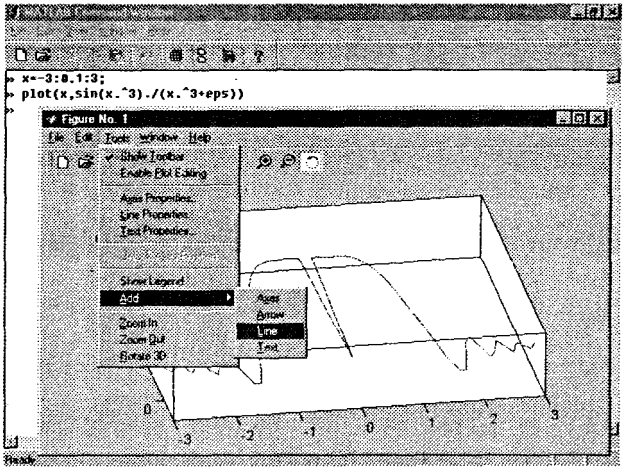


**Рис. 1.17.** Графическое окно MATLAB

- Line Properties — вывести окно свойств линий графика;
- Text Properties — задать вывод текстовых комментариев;
- Lock/Unlock Axes Position — заблокировать или разблокировать позицию осей;
- Show Legend — показать легенду;
- Add — добавить в график новый объект из подменю (см. рис. 1.17);
- Zoom In — увеличить размеры графика;
- Zoom Out — уменьшить размеры графика;
- Rotate 3D — начать вращение графика мышью.

Еще одно важное усовершенствование в версии MATLAB 5.3.1 — появление панели инструментов графического окна (эта панель, если она выведена, располагается под строкой меню графического окна). Первые четыре кнопки панели служат для создания нового графического окна, открытия файла, записи файла на диск и печати графиков принтером. Остальные повторяют уже перечисленные команды из меню Tools за исключением первой из них.

Хорошее впечатление оставляет возможность вращения графиков мышью — прием, введенный в целый ряд систем компьютерной математики (MathCAD, Maple V и Mathematica 4). При вводе этой команды вокруг фигуры появляется обрамляющий ее параллелепипед, который можно вращать мышью (при нажатой левой кнопке) в том

или ином направлении. Отпустив кнопку мыши, можно наблюдать график в пространстве. Интересно, что эта возможность действует даже в отношении двумерных графиков (см. рис. 1.17). При этом вращается плоскость, в которой расположен график.

Большинство графиков, которые описываются в книге, представлены копиями только самих графиков, а не всего графического окна. Для получения таких копий использовалась команду Copy Figure из меню Edit окна графики. Такое представление делает приведенные рисунки одинаковыми для всех версий MATLAB от 5.0 и выше.

## Построение графиков функций одной переменной

Выше мы рассмотрели построение графика одной функции одной переменной. Пойдем дальше и попытаемся построить графики сразу трех функций: $\sin(x)$, $\cos(x)$ и $\sin(x)/x$. Прежде всего отметим, что эти функции могут быть обозначены переменными:

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

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), а f1, f2, f3, ... — векторы значений функций, графики которых строятся в одном окне. В нашем случае для построения графиков указанных функций мы должны записать следующее:

```
» plot(x,y1,x,y2,x,y3)
```

Построив график (сделайте это самостоятельно), вы увидите, что графики функций, обозначенных переменными y1 и y2, будут благополучно построены, а вот график функции, обозначенной переменной y3, отсутствует. Причина этого казуса уже обсуждалась — при вычислении функции y3=sin(x)/x, если x представляет собой массив, 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. 1.18.

Разумеется, MATLAB имеет средства для построения графиков и таких функций, которые имеют особенности. Не обсуждая их подробно, просто покажем, как это делается, с помощью другого графического оператора — fplot:

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

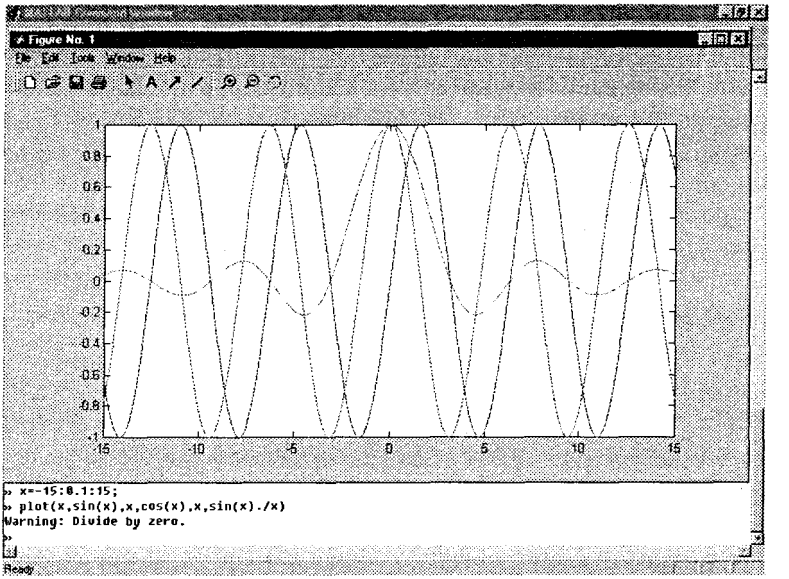


**Рис. 1.18.** Построение графиков трех функций

## Построение гистограммы

В прикладных расчетах часто встречаются графики, именуемые гистограммами. Чаще всего используются столбчатые гистограммы (или диаграммы), отражающие содержимое некоторого вектора V. При этом каждый элемент вектора представляется столбцом, высота которого пропорциональна значению элемента. Столбцы нумеруются и масштабируются по отношению к максимальному значению наиболее высокого столбца. Выполняет построение команда (оператор) `bar(V)` (рис. 1.19).

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

Рисунок 1.19 дает представление о новом меню Tools окна графики. Нетрудно заметить, что кроме опции вывода инструментальной панели здесь имеется целый ряд других команд, которые будут рассмотрены в дальнейшем. Это команды вывода свойств графических объектов, изменения масштаба графика, добавления на него осей и т. д.

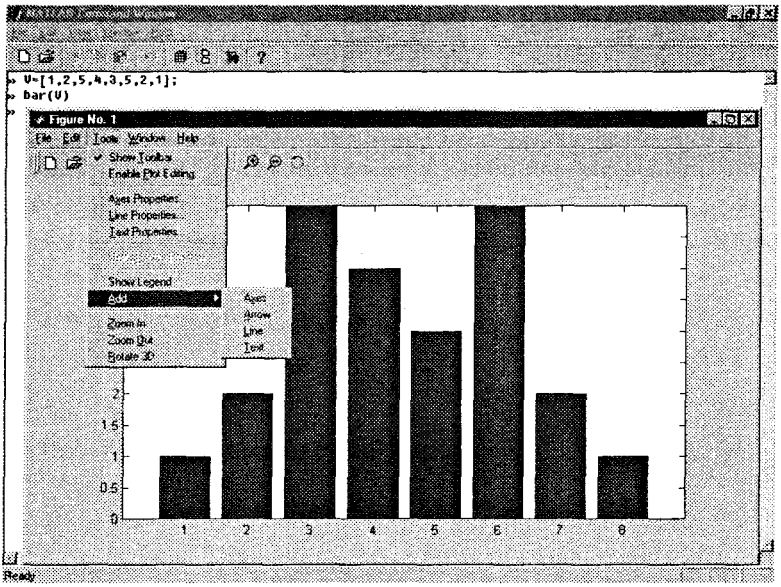


**Рис. 1.19.** Построение гистограммы значений элементов вектора

## Построение трехмерных графиков

*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. 1.20. 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.

Как видно из рис. 1.20, окно трехмерной графики также имеет панель инструментов. На рис. 1.20 показано также открытое меню Help окна трехмерной графики.

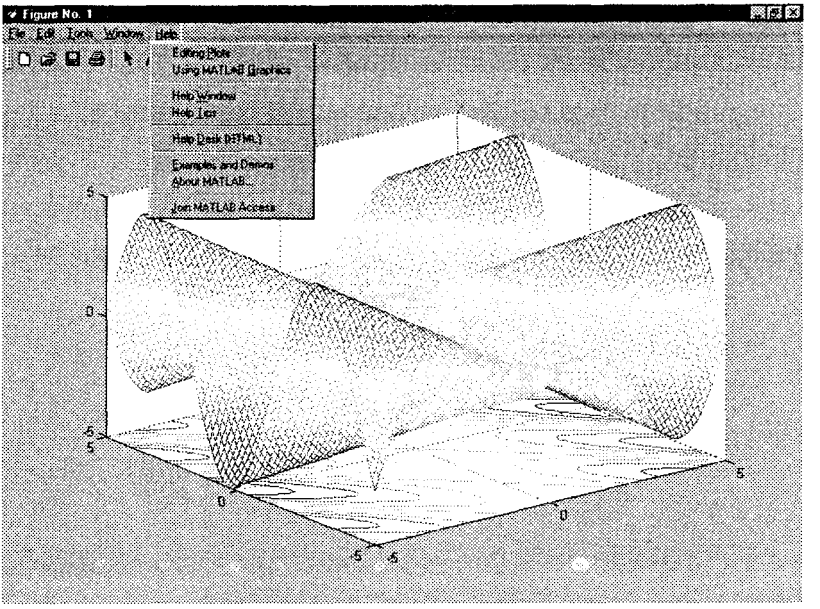


**Рис. 1.20.** Окно с графиком поверхности и ее проекции на плоскость под фигурой

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 помогут как данная книга, так и справочная система MATLAB.

## Вращение графиков мышью

В новой версии MATLAB 5.3.1 имеется возможность вращать построенную фигуру мышью и тем самым наблюдать ее под разными углами. Рассмотрим эту возможность на примере построения логотипа системы MATLAB — мембраны. Для этого, введя команду `membrane`, получим график поверхности мембраны. Для вращения графика достаточно активизировать последнюю кнопку панели инструментов графического окна трехмерной графики с изображением пунктирной окружности со стрелкой. Теперь, введя указатель мыши в область графика и нажав левую кнопку мыши, можно круговыми движениями заставить график вращаться вместе с обрамляющим его параллелепипедом (рис. 1.21).

Любопытно, что вращать можно и двумерные графики, наблюдая поворот плоскости, в которой они построены (см. пример на рис. 1.19).

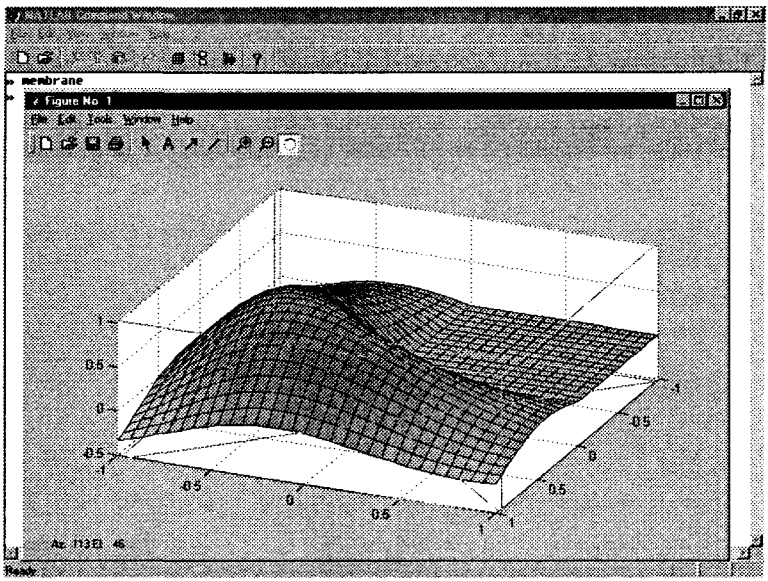


**Рис. 1.21.** Вращение трехмерной фигуры мышью

## Редактор свойств графиков

*Графики в системе MATLAB строятся обманчиво просто.* Связано это с тем, что многие свойства графиков установлены по умолчанию. К таким свойствам относятся вывод или скрытие координатных осей, положение их центра, цвет линии графика, толщина линии и т. д., и т. п. С помощью опций графических команд свойства и вид графиков можно менять в широких пределах. Однако этот путь требует хорошего знания деталей входного языка общения и программирования системы MATLAB.

MATLAB имеет возможность настраивать свойства графиков с помощью редактора свойств — Graphics Properties Editor. Его можно вызвать из меню File окна командного режима MATLAB с помощью команды Show Graphics Properties Editor. При этом можно просмотреть общие свойства графиков.

При построении графиков появляется графическое окно (рис. 1.22). В меню File этого окна имеется команда Property Editor, которая вы-

зывает редактор свойств, относящихся к графику, расположенному в данном окне. В полях над списком свойств можно видеть наименование выделенного свойства и его значения (в виде чисел, списков, спецзнаков и т. д.).

На рис. 1.22 показан пример коррекции опции цвета `Color` — вместо ее значения `[0.8 0.8 0.8]` задано `[0 0.8 0.8]`. Результат виден в небольшом окне просмотра и на графике. В данном случае меняется цвет обрамления графика. Этот способ удобен, когда нужно подправить одно или несколько свойств графика.

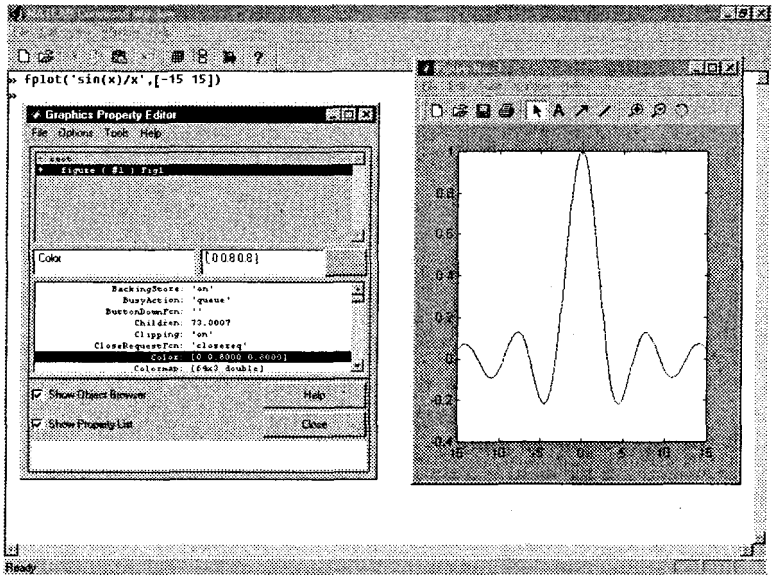


**Рис. 1.22**. Окно графика (справа) и окно редактора свойств графиков (слева)

## Управление форматом графиков

Изменение свойств графика принято называть его *форматированием*. Для этого окно графики имеет специальный инструментарий, который содержится в меню Tools окна графики. В указанном меню имеются три характерные команды:

- Axes Properties — установка параметров осей координат;
- Line Properties — установка параметров линий графика;
- Text Properties — установка параметров текста графика.

Рассмотрим пример форматирования осей координат. При выделении графика и исполнении команды Axes Properties появляется окно с очевидными свойствами координатных осей (рис. 1.23).

Нетрудно заметить, что в этом окне можно задать титульную надпись, надписи по осям, установить линейный или логарифмический масштаб осей, размеры области представления графиков по осям и включение показа сетки (Grid). На рис. 1.23 показаны новые установки свойств осей. Кнопка Apply (Применить) позволяет применить сделанные установки к графику до закрытия окна диалога. Кнопка OK вводит сделанные установки и закрывает окно установки свойств осей. Назначение других кнопок очевидно. Если компьютер оснащен должным набором шрифтов, то надписи на графиках могут быть сделаны на русском языке.

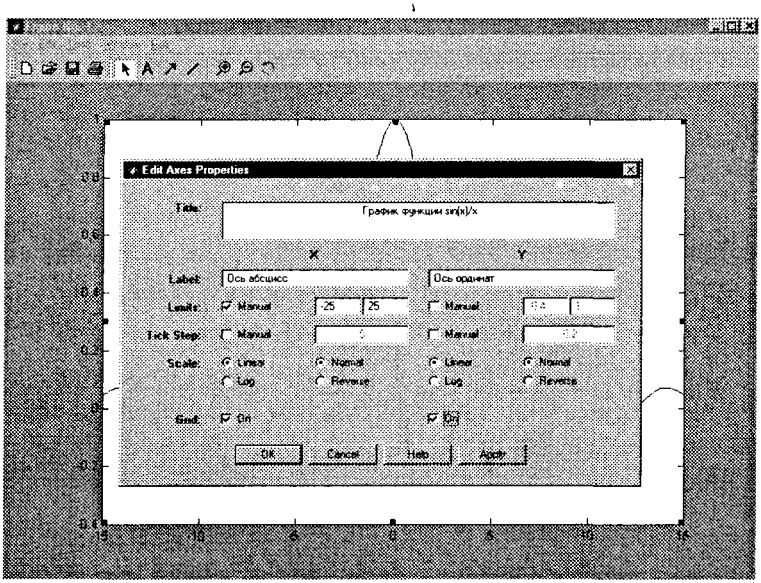


**Рис. 1.23.** Окно графики с окном свойств координатных осей

Теперь перейдем к форматированию линий графика. Для этого нужно щелкнуть на линии графика. Линия будет выделена, после чего можно будет использовать команду Line Properties для изменения ее свойств. С помощью этого окна можно задать ширину, тип и цвет выделенной линии, а также задать изображение метки в узловых точках, через которые проводится линия.

На график можно нанести надписи с помощью кнопки панели инструментов с буквой A. Место надписи фиксируется щелчком мыши. На рис. 1.24 показан отформатированный график с текстовым блоком, созданным таким образом в левой верхней части поля графика.

Полученную таким образом надпись можно выделить и перенести мышью в любое другое место. Рисунок 1.24 показывает перенос надписи вправо и применение для текстового блока контекстно-зависимого меню. Напоминаем, что это меню появляется при щелчке правой кнопкой мыши на заданном объекте. В этом меню имеются все команды, доступные для данного объекта в данной ситуации. На рис. 1.24, в частности, показано задание надписи увеличенным в размере (с 10 до 14 пунктов) шрифтом; затем производится задание линии-стрелки с помощью соответствующей кнопки панели инструментов (рис. 1.25).

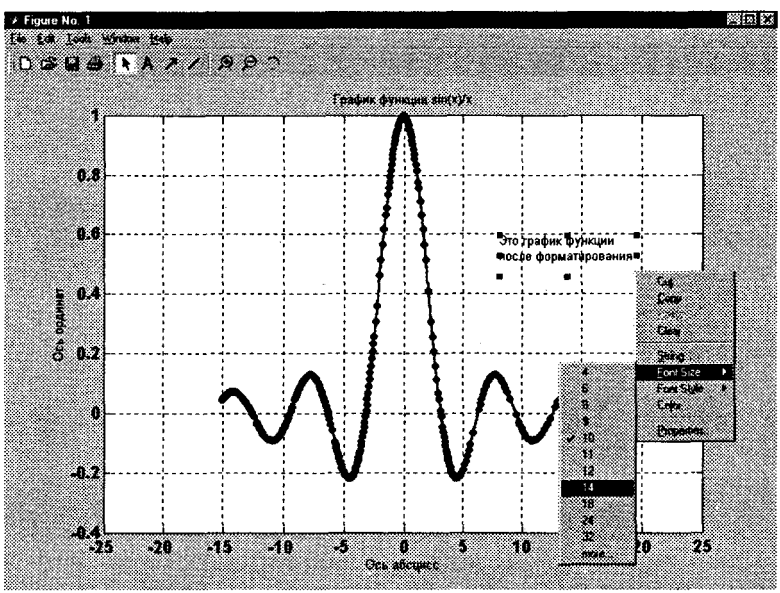


**Рис. 1.24.** Отформатированный график с надписью

Дополнительно можно изменить размеры графика (см. меню Tools и его команды Zoom In и Zoom Out), начать вращение мышью графика (команда Rotate 3D), добавить отрезок прямой или иной графический примитив (подменю Add) и подключить к графику *легенду* —

обозначение кривых (команда Show Legend). Поскольку наш график содержит одну кривую, то легенда представляет собой обозначение единственной линии в правом верхнем углу рисунка (рис. 1.25).

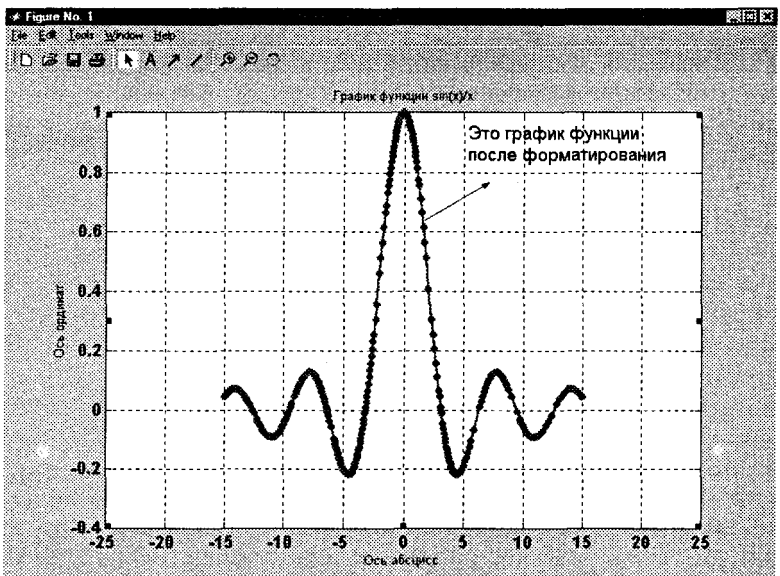


**Рис. 1.25.** Окончательно сформированный график

Завершая краткий обзор возможностей базовой системы MATLAB 5.3.1, мы еще раз отмечаем, что данная глава не претендует на исчерпывающее описание этих возможностей. Более детальное описание базовой системы читатель может найти в книгах, перечисленных в начале главы.

# Глава 2
# Расширение Notebook

## Назначение расширения Notebook

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

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

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

В основе Notebook лежит механизм *динамической связи* (DDE — Dynamic Data Exchange) между различными приложениями в операционных системах Windows 95/98. При этом возможна передача изменяемых данных из одного приложения в другое и наоборот. Приложение, передающее данные, называют сервером, а принимаю-

щее данные — клиентом. В системе «Word — MATLAB», по существу реализованной в Notebook, обе программы могут играть роли сервера и клиента.

## Создание 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 из папки NOTEBOOK системы MATLAB (или специальный «пустой» файл-шаблон). Можно также дать команду notebook из окна MATLAB — при этом произойдет загрузка текстового редактора Word 97 или Word 95 (в зависимости от того, какой из них установлен). Приведенные далее примеры даны для Word 97.

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

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

На рис. 2.1 показана подготовка к созданию Notebook. Необходимо запустить текстовый процессор Word (в нашем случае это Microsoft Word for Windows 95/98). В позиции Файл главного меню следует исполнить команду Открыть. Появится окно для поиска нужного файла в файловой системе компьютера — это окно на рис. 2.1 показано на фоне окна текстового процессора Word. В окне поиска файла видна настройка на загрузку файла readme.doc из папки NOTEBOOK системы MATLAB.

После этого надо нажать кнопку Открыть окна выбора 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 разнообразные результаты вычислений.

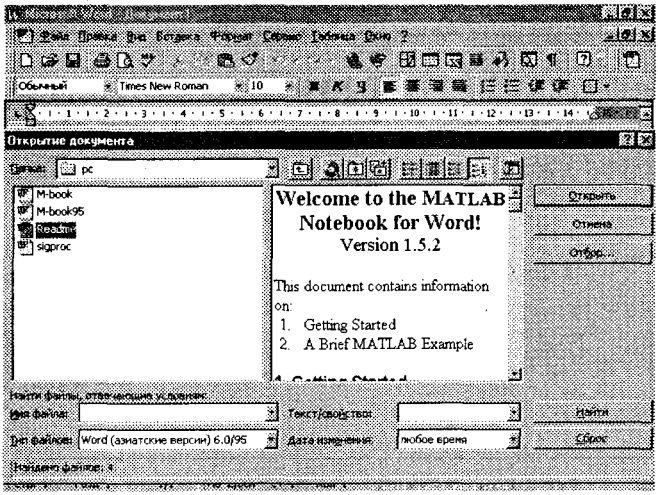


**Рис. 2.1.** Окно текстового процессора Word и окно открытия файла

В начале процесса загрузки файла вы увидите момент загрузки системы MATLAB — появление рисунка с ее логотипом. В конце процесса загрузки появится текст файла readme, как показано на рис. 2.2.

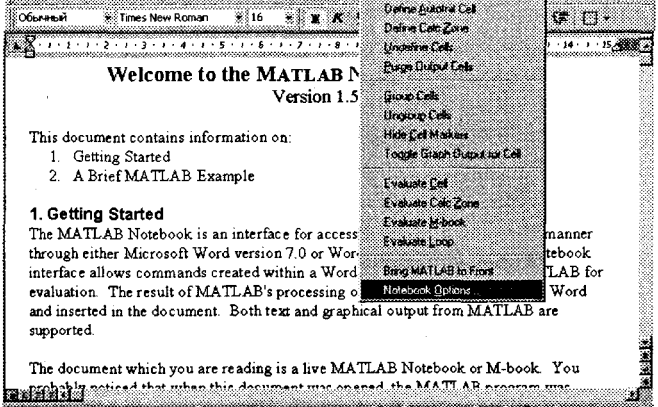


**Рис. 2.2.** Окно текстового процессора Word с загруженным файлом readme

Внимательный читатель тут же отметит некоторые необычные свойства окна текстового процессора Word. Первое, что бросается в глаза при сравнении рис. 2.1 и 2.2, — это появление в строке меню нового меню Notebook, которое на рис. 2.1 отсутствует. Это меню содержит множество команд, относящихся к приложению Notebook, созданному на основе текстового процессора Word. В раскрытом виде меню Notebook показано на рис. 2.2.



# Демонстрация возможностей Notebook

## Эволюция магической матрицы

Файл readme.doc содержит несколько наглядных примеров для демонстрации возможностей Notebook. Для оценки этих возможностей достаточно просмотреть файл и остановиться на разделе «A Brief MATLAB Example». Часть документа (рис. 2.3), заключенная в жирные квадратные скобки, представляет собой ячейки ввода и вывода, динамически связанные с решателем системы MATLAB.

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

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

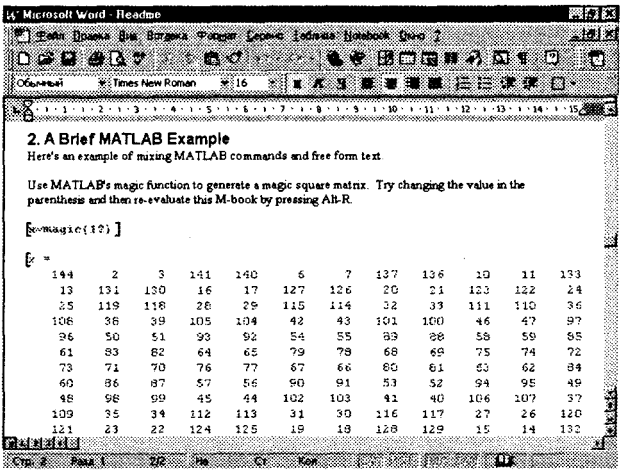


**Рис. 2.3.** Пример формирования магической матрицы размером 12×12

Под ней показана ячейка вывода, реализующая вывод магической матрицы размером 12×12. На рис. 2.2 и 2.3 виден также обычный

англоязычный текст, набранный в редакторе Word разными стилями. Напоминаем, что под стилем понимается совокупность параметров текста: используемые наборы шрифтов, их размеры и стили, цвета символов, межстрочные расстояния, отступы абзацев и другие параметры. Word имеет обширные возможности для создания текстов различного стиля.

Теперь покажем, что ячейки MATLAB в тексте документа способны к изменению. Для этого обычным путем поместим маркер ввода в ячейку ввода и заменим параметр функции `magic`, равный 12, на значение 4. Не выводя маркера из этой ячейки, нажмем одновременно клавиши Ctrl и Enter. Вы тут же увидите, что ячейка вывода изменится — вместо магической матрицы размером 12×12 появится магическая матрица меньшего размера — 4×4. Это и есть эволюция ячеек Notebook (рис. 2.4).

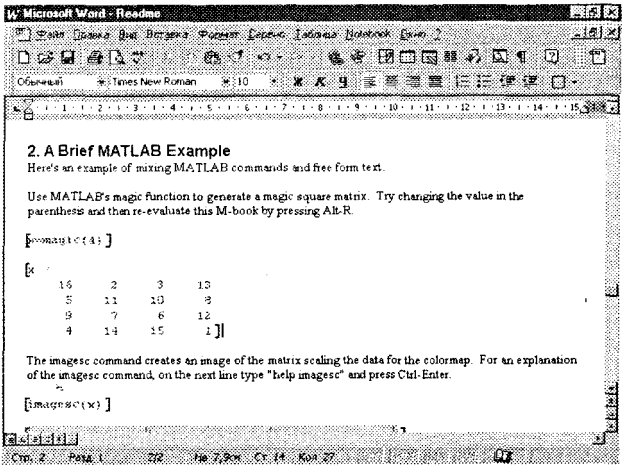


**Рис. 2.4.** Пример изменения размера магической матрицы

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

## Эволюция рисунка

Пролистав документ `readme` чуть дальше, мы увидим команду `images(x)`, которая дает графическое представление содержимого ма-

гической матрицы. При этом каждый ее элемент отображается квадратом с функциональной окраской, зависящей от значения элемента матрицы. Все это для матрицы размером 4×4 показано на рис. 2.5.

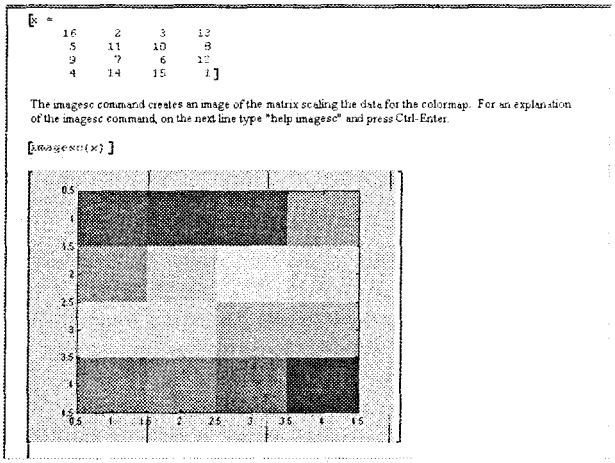


**Рис. 2.5.** Матрица размером 4×4 и ее графическое представление

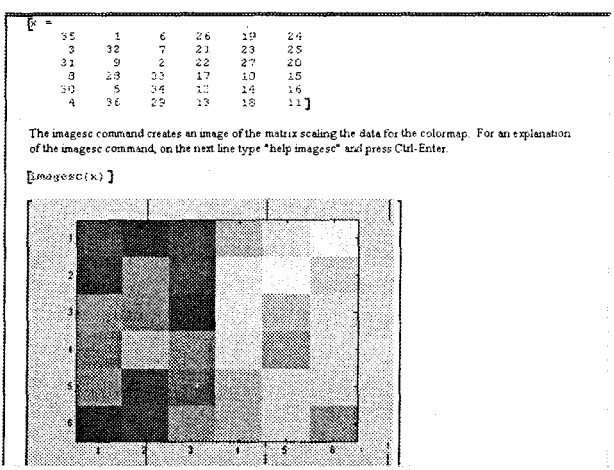


**Рис. 2.6.** Матрица размером 6×6 и ее графическое представление

Теперь измените размер матрицы, например на 6×6, и создайте новую матрицу так, как это было описано ранее. Далее поместите в

ячейку с командой images маркер ввода и нажмите клавиши Ctrl+Enter. На рис. 2.6 видно, как изменилась картинка — число квадратиков на ней возросло. Это и есть пример эволюции рисунка.

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

# Создание новых документов класса Notebook

## Открытие нового документа класса Notebook

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

Иногда из-за неточностей в установке файловой структуры такой способ не дает положительного результата. На этот случай в каталоге NOTEBOOK предусмотрено два «пустых» файла шаблонов: M-book.dot (для Word 97) и Mbook95.dot (для Word 95). Загрузив эти файлы, можно установить Word для подготовки документа в виде Notebook, причем окно документа будет пустым и готовым для создания нового документа в формате шаблона[1].

## Пример создания документа класса Notebook

На рис. 2.7 показан созданный Notebook, который поясняет, как выполнить построение графика трех функций. Вначале в нем введен абзац текста с описанием создания ячейки ввода.

Для создания ячейки ввода надо установить маркер ввода на свободную строку и исполнить команду Define Input Cell из меню Notebook текстового процессора Word. После этого вводится текст ко-

[1] Чтобы создать не *шаблон*, а новый *документ* Notebook, необходимо воспользоваться командой Файл ▸ Создать (именно *командой меню*, а не комбинацией клавиш Ctrl+N или кнопкой панели инструментов!) и выбрать шаблон M-book.dot или Mbook95.dot (файл шаблона должен находиться в каталоге шаблонов Microsoft Office). — *Примеч. ред.*

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



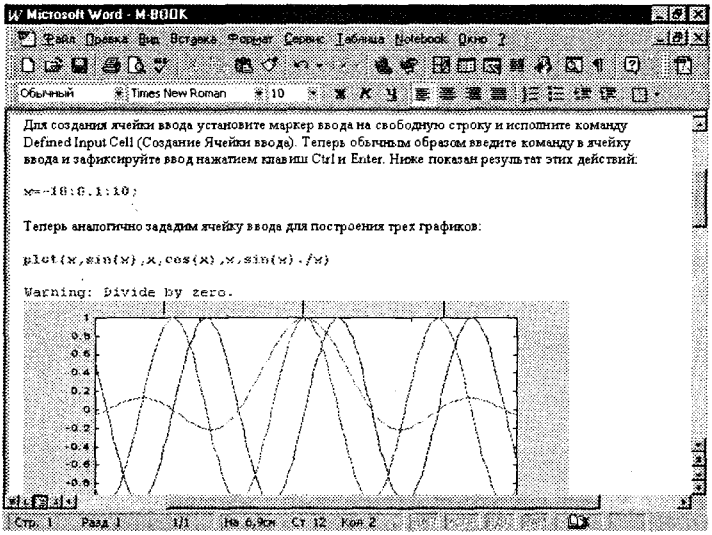


**Рис. 2.7.** Созданный новый Notebook

Далее аналогично введем команду построения графика трех функций (см. рис. 2.7). Теперь после нажатия клавиш Ctrl+Enter тут же появится ячейка вывода с графиком трех функций. Установив в нее курсор мыши и дважды щелкнув левой кнопкой, можно наблюдать выделение графика и появление в его рамке прямоугольников, за которые можно «уцепиться» курсором мыши, чтобы изменить размеры графика.

## Ячейки ввода MATLAB в тексте Word

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

## Преобразование текстов Word в ячейки ввода

Иногда желательно создать Notebook из самого обычного файла редактора Word. Для этого надо создать новый Notebook и загрузить нужный файл, используя команду Файл (File) меню Вставка (Insert). В созданный таким образом шаблон Notebook можно добавить ячейки ввода MATLAB[1].

## Сохранение документов класса Notebook

Созданный Notebook записывается так же, как любой другой документ Word. Если нужно сохранение Notebook с заданным именем, то надо исполнить команду Save As (Сохранить как) из меню File (Файл) главного меню. Появится стандартное окно записи файла, и его можно записать как файл документа с расширением .doc. Если использовался *шаблон*, то это будет *окно шаблона* (рис. 2.8). В этом случае файл записывается с расширением .dot.

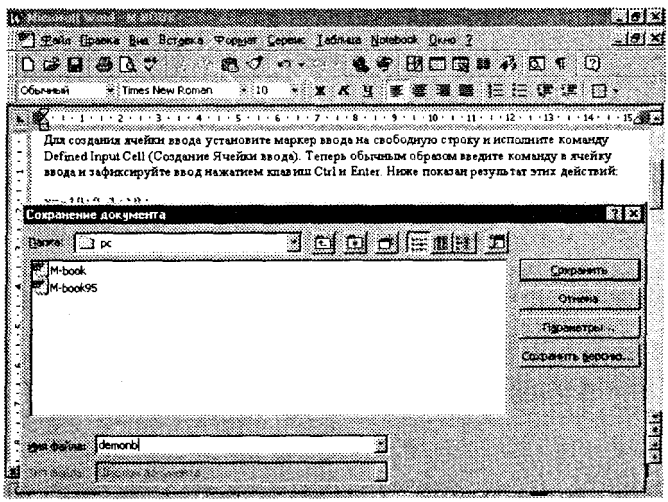


**Рис. 2.8.** Запись Notebook на жесткий диск

[1] Описанный способ является не вполне корректным, так как в результате будет создан не документ, а *шаблон* документа Word. Чтобы превратить текстовый документ Word в Notebook, необходимо воспользоваться командой меню Шаблоны и надстройки (в Word 95 она находилась в меню Файл, а в Word 97 переехала в меню Сервис), чтобы присоединить к документу шаблон M-book.dot или Mbook95.dot. — *Примеч. ред.*

Для редактирования уже созданного Notebook достаточно загрузить его с помощью команды Открыть из меню Файл.

# Меню Notebook



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

## Создание ячейки ввода

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

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

## Создание ячейки автостарта

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

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

## Создание зоны вычислений

Команда Define Calc Zone превращает выделенный текст (с ячейками ввода и вывода) в некоторую *зону вычислений*, решающую опреде-

ленную задачу. Таких зон в М-книге может быть много, и они могут использоваться для решения ряда задач. Примером, где такие зоны полезны, являются сборники различных задач с действующими примерами.



### Преобразование ячеек MATLAB в обычный текст

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

### Удаление ячеек вывода

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

### Создание многострочной ячейки ввода

Команда Group Gells объединяет все ячейки ввода в выделенной части документа в *группу* ячеек ввода. При этом выделенный текст размещается после этой группы, за исключением той части текста, которая размещена до первой ячейки. Ячейки вывода, имеющиеся в выделенном тексте, устраняются. Если первая ячейка ввода имеет статус ячейки автостарта, то такой статус приобретают все ячейки группы. Группе ячеек можно придать этот статус и с помощью команды AutoInit Cell.

### Преобразование группы ячеек в ячейки ввода

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

### Управление показом маркеров

Как уже указывалось, обычно ячейки ввода и вывода MATLAB отмечаются жирными серыми квадратными скобками (маркерами), отличными от обычных квадратных скобок. Маркеры видны только

на экране дисплея. Команда Hide/ Show Cell Markers позволяет убрать или, напротив, включить показ маркеров. При печати М-книг маркеры не печатаются независимо от того, видны они на экране дисплея или нет.



### Пуск оценки ячеек

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

### Пуск оценки зоны

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

### Пуск оценки всей М-книги

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

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

### Циклическая оценка

Выделенную ячейку ввода или группу ячеек можно выполнить циклически с помощью команды Evaluate Loop (Циклическая оценка). Рисунок 2.9 показывает подготовку к такой оценке. В первой строке

ввода задано значение переменной i=0. Во второй строке заданы вычисления i=i+1 и magic(i). Первоначально, таким образом, выдается магическая матрица размером 1×1.

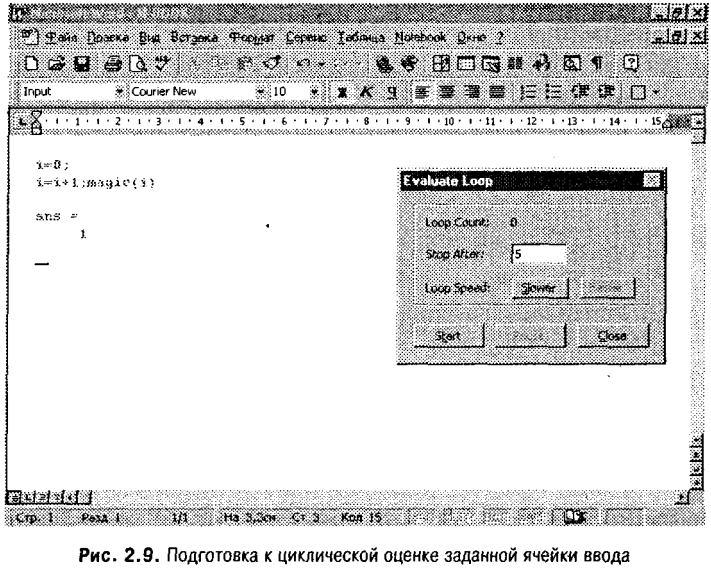


**Рис. 2.9.** Подготовка к циклической оценке заданной ячейки ввода

В правой части окна рис. 2.9 видно диалоговое окно Evaluate Loop, которое появляется при исполнении команды Evaluate Loop (Циклическая оценка). В нем имеется поле Stop After (Остановить После), задающее число циклов оценки. Кнопки Slower (Медленнее) и Faster (Быстрее) задают (условно) медленную и быструю оценку текущей ячейки ввода — в нашем случае это вторая ячейка из двух, показанных на рис. 2.9.

Запустив циклический просмотр нажатием кнопки Start, можно наблюдать построение матриц постоянно увеличивающегося размера i×i. По завершении циклов можно увидеть результаты, показанные на рис. 2.10. На ваших глазах произойдет вывод магических матриц с размером от 1×1 до 6×6, так что в конечном счете будет выведена такая матрица размером 6×6.

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

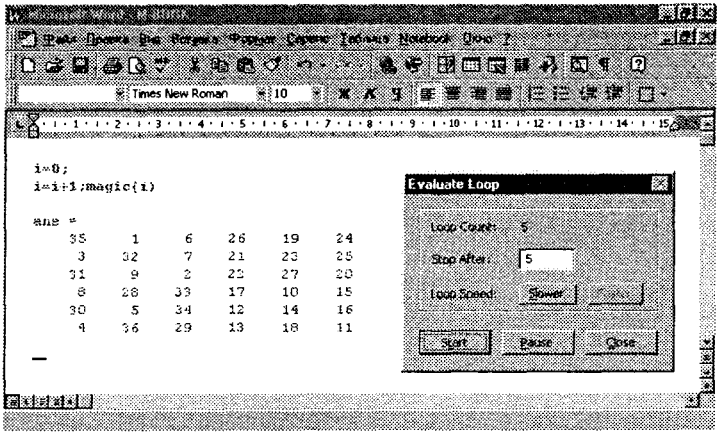


**Рис. 2.10.** Конец циклической оценки

## Вывод окна MATLAB на передний план

Команда Bring MATLAB to Front выводит на передний план командное окно MATLAB (рис. 2.11).

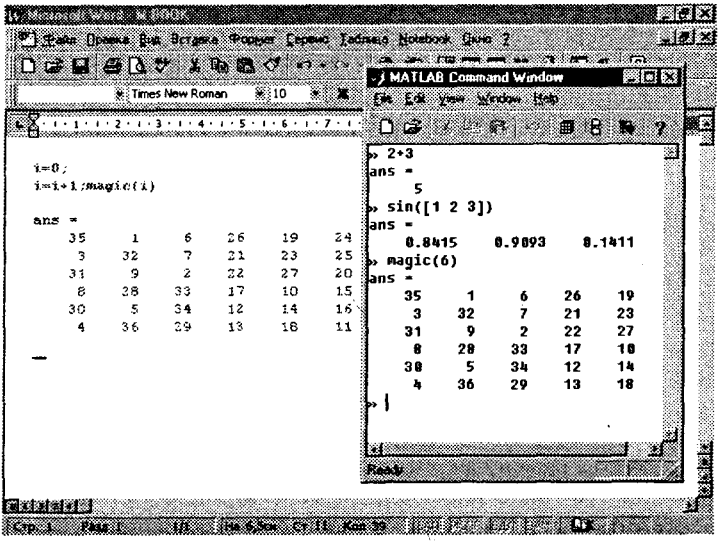


**Рис. 2.11.** Пример вывода окна системы MATLAB на передний план

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

## Установка опций Notebook

Последняя команда меню Notebook — Notebook Options — выводит окно установки опций, в основном связанных с параметрами вывода результатов вычислений. Это окно показано на рис. 2.12.

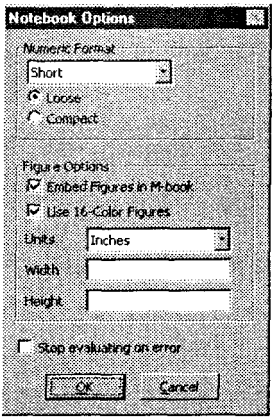


**Рис. 2.12.** Окно установки опций Notebook

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

Следующая панель — Figure Options — содержит опции отображения графики и рисунков. Установка флажка Embed Figures in M-book обеспечивает размещение рисунков прямо в тексте книги — как разновидность ячеек вывода. Снятый флажок означает, что рисунки будут выведены в отдельных окнах. Опция Use 16-Color Figures задает отображение рисунков в формате с 16 цветами, в противном случае отображается 256 цветов. Опция Units задает единицы измерения размеров графиков — в дюймах, сантиметрах или пикселях.

Ширину и высоту рисунков задают, соответственно, параметры Width и Height.

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

# Глава 3
# Пакет расширения Symbolic Math

## Назначение пакета Symbolic Math

В систему MATLAB 5.3.1 входит обновленная версия 3.0 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. Объективности ради надо отметить, что Maple V R4 — не последняя реализация, сейчас уже есть версия Maple V R5 и даже Maple 6.

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 m-файла, задающего данную 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 6 в своем ядре и в расширениях имеет около 3000 функций. Система MATLAB с пакетом Symbolic, включающим в себя чуть больше сотни символьных команд и функций, намного уступает Maple по количеству таких команд и функций. Однако в данный пакет включены лишь наиболее важные и широко распространенные функции. Кроме того, есть специальная команда, которая дает доступ к ядру

Maple, что заметно расширяет круг используемых функций. 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 новых реализаций R5 и Maple 6 системы Maple.

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



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

## Демонстрационные примеры

Для ознакомления с пакетом Symbolic можно использовать следующие команды:

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

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

## Работа с объектами и переменными

### Задание символьных переменных

Поскольку переменные системы MATLAB по умолчанию не определены и традиционно задаются как векторные, матричные, число-

вые и т. д., то есть не имеющие отношения к символьной математике, для реализации символьных вычислений нужно прежде всего позаботиться о создании специальных *символьных переменных*. В простейшем случае их можно определить как строковые переменные, заключив имена в апострофы. Следующие примеры иллюстрируют действие этого приема:

```
» sin(x)^2+cos(x)^2
??? Undefined function or variable 'x'.

» sin('x')^2+cos('x')^2
ans =
       1
```

Итак, в первом случае система MATLAB «возмутилась» нашей небрежностью и сообщила, что функция или переменная x не определена и ни о каких вычислениях синуса и косинуса речи быть не может. Вместе с тем она подсказала, как надо сделать это определение — заключить имя переменной в апострофы, ибо таким образом система получает информацию о необходимости включить символьный режим вычислений. Поэтому во второй раз получен вполне осмысленный результат — сумма квадратов синуса и косинуса переменной 'x' выдана равной 1.

Этот результат сразу демонстрирует характерную особенность символьных вычислений: они возвращают результаты даже в том случае, когда переменные не определены. Правила выполняемых при этом преобразований заложены в ядре символьных операций, в нашем случае это ядро системы Maple V R4.

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

## 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 сим-

вольная 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 следующем виде:

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

Примеры применения функции sym представлены ниже:

```
» sym(1/50,'f')
ans =
'1.47ae147ae147b'*2^(-6)
» sym(4/6,'r')
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 опознавать его как символьный, в отличие от обычных численных результатов.

## Функция создания группы символьных объектов syms

Для создания группы символьных объектов служит функция `syms`:

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

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

## Функция создания списка символьных переменных findsum

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

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

Примеры:

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

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

# Функции вывода и преобразования символьных выражений

## 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 т. д. Этот недостаток, однако, часто оборачивается достоинством — запись осуществляется в простом текстовом формате, облегчающем ввод выражений. Да и вывод символьных выражений в таком формате поддерживается практически всеми устройствами вывода.

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

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

Примеры:

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

Нетрудно заметить, что в выведенных выражениях используются надстрочные показатели степени и (в необходимых случаях) знаки деления в виде горизонтальной черты. Однако для простых констант (например, 1/3) по-прежнему используется знак деления в виде наклонной черты.

## 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.



## 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.

## 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 MATLAB-выражения S в форму, соответствующую записи на языке Fortran. Примеры:

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

```
ans =
     t0 = x**2*y**3/z**4
```

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



## Контроль допустимости имен — isvarname

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

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

## 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 символов «^», «*» и «/». 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
```

## Арифметика произвольной точности

### Установка количества знаков чисел — digits

Арифметикой произвольной точности называют вычисления, у которых все числа результатов являются точными. Функция digits слу-

жит для установки числа цифр в числах арифметики произвольной точности. Она используется в следующем виде:

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

Примеры:

```
» digits
Digits = 32
» vpa pi
ans =
3.14159265358979323846264338327 95
» digits(6)
» pi
ans =
    3.1416
```

## 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 вычислений вообще без какой-либо погрешности или со сколь угодно малой погрешностью. Такие вычисления не очень удачно называют *арифметикой произвольной точности* — правильнее говорить просто о *точной* арифметике.

Для проведения вычислений в арифметике произвольной точности служит функция `vpa`:

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

Примеры:

```
» vpa(exp(1),50)
```

```
 ans =
 2.71828182845904509079559829842764884233474731445 31
» vpa([2*pi exp(1) log(2)],10)
 ans =
 [ 6.283185308, 2.718281828, .6931471806]
```

# Символьные операции с матрицами



## Задание или извлечение диагональных элементов матриц — diag

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

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

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

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

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

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



## 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*-й диагонали (правила задания `K` были описаны выше).

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

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

## 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*-й диагонали (правила задания `K` были описаны выше).

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

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

## 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, то есть $\mathbf{X}^{-1} = \mathbf{E}/\mathbf{X}$. Для обращения (инвертирования) матрицы в символьном виде используется функция `inv`:

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

## Вычисление детерминанта матрицы — det

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

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

## 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)` — аналогично предшествующей функции при `tol=max(size(A))*norm(A)*eps`.

Пример:

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

## Приведение матрицы к верхней треугольной форме — rref

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

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

## 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]
```

## 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]
```

## Вычисление собственных значений и 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
```

## Сингулярное разложение матриц — svd

Для сингулярного разложения матриц используется функция svd в ряде форм:

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

Примеры:

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

## 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 подобия V, так что V\A*V = J. Столбцы матрицы V являются обобщенными собственными векторами. Примеры:

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

## Вычисление характеристического полинома матриц — poly

Для вычисления характеристического полинома матриц используется функция poly:

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

Пример:

```
» syms a b c d;
» A=[a b;c d];
» poly(A,'p')
```

```
ans =
p^2-p*d-a*p+a*d-b*c
```

### 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)]
```

## Символьные операции математического анализа

### 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')) Она возвращает символьное значение первой производной от символьного выражения или массива символьных выражений S по переменной v. Эта функция возвращает

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

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

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

Примеры:

```
» x=sym('x');y=sym('y');
» diff(x^y)
 ans =
```

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



## Функция вычисления интегралов — int

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

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

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

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

$$\int_0^a \int_0^a \int_0^a (x^2 + y^2)z\,dx\,dy\,dz.$$

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 из приведенных примеров вычисляет интеграл с бесконечным верхним пределом, а второй — тройной интеграл.

## 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 $x \to a$) значение функции неограниченно приближается к $L$. Это обозначается следующим образом:

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

Предел может быть конечным числом, положительной или отрицательной бесконечностью.

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

Для вычисления пределов аналитически (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 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]
```

## Функция разложения выражения в ряд Тейлора — taylor

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

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

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

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

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

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

Примеры разложения функций в ряд:

```
» x=sym('x');
» F=sin(x);
» taylor(F)
 ans =
 x-1/6*x^3+1/120*x^5
```

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



## 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

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

Для ее вычисления используется функция jacobian:

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

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

```
» jacobian(x*y,v)
ans =
[ y, x]
```

## Функция вычисления сумм рядов — symsum

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

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

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

Для аналитического вычисления суммы ряда служит команда `symsum`:

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

Примеры на вычисление сумм даны ниже:

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

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

- для одного уравнения и одной переменной решение возвращается в виде одномерного или многомерного массива ячеек;
- при одинаковом числе уравнений и переменных решение возвращается в упорядоченном по именам переменных виде;
- для систем с одним выходным аргументом решение возвращается в виде массива записей.

Примеры решения уравнений:

```
» 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 частей уравнений для получения графической интерпретации решений. Это особенно полезно в том случае, если решение носит множественный характер, поскольку функция `solve` дает только одно решение.

## 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, добавив ее в конец списка параметров функции `dsolve`. Символ `D` обозначает производную по независимой переменной, то есть $d/dt$, при этом `D2` означает $d^2/dt^2$ и т. д. Имя независимой переменной не должно начинаться с буквы `D`.

Начальные условия задаются в виде равенств `'y(a)=b'` или `'Dy(a)=b'`, где `y` — независимая переменная, `a` и `b` — константы. Если число начальных условий меньше, чем число дифференциальных уравнений, то в решении будут присутствовать произвольные постоянные `C1`, `C2` и т. д. Правила вывода подобны приведенным для функции `solve`.

Примеры применения функции `dsolve`:

```
» dsolve('D2x=-2*x')
ans =
```

```
C1*cos(2^(1/2)*t)+C2*sin(2^(1/2)*t)
» dsolve('D2y=-2*x+y','y(0)=1','x')
 ans =
 (2*x*exp(x)+(-C2+1)*exp(x)^2+C2)/exp(x)
```

# Интегральные преобразования



В этом разделе описаны важные интегральные преобразования. Преобразования Фурье составляют основу метода спектрального анализа сигналов и вопросов, связанных с анализом радиотехнических цепей. Трудно переоценить и преобразования Лапласа, составляющие основу символического метода расчета электрических и радиотехнических цепей. Z-преобразования широко применяются в теории автоматического управления. Все эти преобразования (прямые и обратные) рассматриваются в данном разделе.

## 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 `fourier`:

- `F=fourier(f)` — возвращает $F(w)$ и определяет независимую переменную с помощью функции `findsym` (по умолчанию это `x`). Если $f = f(w)$, то возвращается функция $F = F(t)$. Таким образом, преобразование имеет вид

  $f = f(x) \Rightarrow F = F(w)$.

- `F=fourier(f,v)` — аналогична ранее приведенной функции, но заменяет аргумент возвращаемой функции $F$ (по умолчанию — $w$), на $v$, то есть реализует преобразование Фурье по формуле

  $$F(v) = \int_{-\infty}^{\infty} f(x)e^{-ivx}dx,$$

- `fourier(f,u,v)` — аналогична исходной функции, но заменяет аргумент `x` в $f(x)$ на $u$, а аргумент в $F(w)$ на $v$, то есть дает следующее преобразование:

  $$F(v) = \int_{-\infty}^{\infty} f(u)e^{-ivu}du.$$

Примеры прямого преобразования Фурье:

```
» syms f F x w u v
» fourier(0.1*x)
ans =
1/5*i*pi*Dirac(1,w)
» F=fourier(sin(x),v)
F =
-i*pi*Dirac(v-1)+i*pi*Dirac(v+1)
» syms t
» f=1/t^2;
» F=fourier(f,v)
F =
pi*v*(Heaviside(-v)-Heaviside(v))
» fourier(exp(-x^2),x,t)
ans =
pi^(1/2)*exp(-1/4*t^2)
» fourier(diff(sym('F(x)')),x,w)
ans =
i*w*fourier(F(x),x,w)
```

## Обратное преобразование Фурье — 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 `ifourier`:

- `f=ifourier(F)` — возвращает результат обратного преобразования Фурье над скалярной символьной функцией `F` независимой переменной `w`. По умолчанию возвращается функция `F(x)`. Таким образом, преобразование имеет вид $F = F(w) \Rightarrow f = f(x)$. Если $F = F(x)$, то данная функция возвращает функцию переменной $t$: $f = f(t)$.

Существуют и другие формы обратного преобразования Фурье, указанные ниже:

- `f = fourier(F,u)` — осуществляет обратное преобразование Фурье с заменой в $f(x)$ $x$ на $u$. Таким образом, реализуется следующая формула преобразования:

$$f(u) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(w)e^{iwu}\,dw.$$

- `f=ifourier(F,v,u)` — осуществляет обратное преобразование Фурье, заменяя в $f(x)$ $x$ на $u$ и в $F(w)$ $w$ на $v$, реализуя следующую формулу преобразования:

$$f(u) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(v)e^{ivu}\,dv.$$

Примеры обратного преобразования Фурье приведены ниже:

```
» syms t x u w
» ifourier(sin(w))
ans =
1/2*i*(-Dirac(x+1)+Dirac(x-1))
» ifourier(w*exp(-2*w)*sym('Heaviside(w)'))
ans =
1/2/(-2+i*x)^2/pi
» ifourier(1/(1 + 2*w),u)
ans =
-1/4*i*exp(-1/2*i*u)*(-Heaviside(u)+Heaviside(-u))
» ifourier(sym('fourier(f(x),x,w)'),w,x)
ans =
f(x)
```



## 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 `laplace`:

- `L=laplace(F)` — обеспечивает прямое преобразование Лапласа для скалярной символьной функции $f(t)$ с независимой переменной $t$. Результат — функция $L(s)$. Если $f = f(s)$, то возвращается функция $L = L(t)$.
- `L=laplace(F,t)` — обеспечивает прямое преобразование Лапласа по модифицированной формуле

$$L(t) = \int_0^{\infty} f(x)e^{-tx}\,dx.$$

- `L=laplace(F,w,z)` — обеспечивает преобразование Лапласа по формуле

$$L(z) = \int_0^{\infty} f(w)e^{-zw}\,dw.$$

Примеры прямого преобразования Лапласа:

```
syms a s t w x
```

```
laplace(t^5)                    возвращает  120/s^6
laplace(exp(a*s))               возвращает  1/(t-a)
laplace(sin(w*x),t)             возвращает  w/(t^2+w^2)
laplace(cos(x*w),w,t)           возвращает  t/(t^2+x^2)
laplace(x^sym(3/2),t)           возвращает  3/4*pi^(1/2)/t^(5/2)
laplace(diff(sym('F(t)'))) возвращает  laplace(F(t),t,s)*s-F(0)
```



## 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 `ilaplace`:

- `F=ilaplace(L)` — возвращает результат обратного преобразования Лапласа для скалярной символьной функции $L$ с независимой переменной по умолчанию $s$. Если $L = L(t)$, то возвращается функция $F = F(x)$.

Есть еще две формы преобразования:

- `F=ilaplace(L,y)` — возвращает результат обратного преобразования Лапласа по формуле
$$f(y) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} L(s)e^{sy}\,ds.$$
- `F=ilaplace(L,y,x)` — выполняет результат обратного преобразования Лапласа по формуле
$$f(y) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} L(x)e^{xy}\,dx.$$

Примеры обратного преобразования Лапласа:

```
syms s t w x y
ilaplace(1/(s-1))                    возвращает  exp(t)
ilaplace(1/(t^2+1))                  возвращает  sin(x)
ilaplace(t^(-sym(5/2)),x)            возвращает  4/3/pi^(1/2)*x^(3/2)
ilaplace(y/(y^2 + w^2),y,x)          возвращает  cos(w*x)
ilaplace(sym('laplace(F(x),x,s)'),s,x) возвращает  F(x)
```

## 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 z-преобразование для скалярной функции $f$ независимой переменной $n$ (по умолчанию). В MATLAB оно реализуется функцией ztrans:

- `F=ztrans(f)` — обеспечивает прямое Z-преобразование вида `f` = $= f(n) \Rightarrow$`F` = $F(z)$, где `n` — символьная переменная, определяемая функцией `findsym`. Если `f` = $f(z)$, то `ztrans(f)` возвращает `F` = $F(w)$.
- `F=ztrans(f,w)` — возвращает `F`, заменяя аргумент по умолчанию `z` на `w`, то есть осуществляет преобразование

$$F(w) = \sum_{n=0}^{\infty} \frac{f(n)}{w^n}.$$

Наконец, еще одна функция — `F=ztrans(f,k,w)` — дает z-преобразование по формуле

$$F(w) = \sum_{k=0}^{\infty} \frac{f(k)}{w^k}.$$

Примеры прямого z-преобразования:

```
syms k n w z
```

| | | |
|---|---|---|
| `ztrans(2^n)` | возвращает | `z/(z-2)` |
| `ztrans(sin(k*n),w)` | возвращает | `sin(k)*w/(1-2*w*cos(k)+w^2)` |
| `ztrans(cos(n*k),k,z)` | возвращает | `z*(-cos(n)+z)/(-2*z*cos(n)+z^2+1)` |
| `ztrans(cos(n*k),n,w)` | возвращает | `w*(-cos(k)+w)/(-2*w*cos(k)+w^2+1)` |
| `ztrans(sym('f(n+1)'))` | возвращает | `z*ztrans(f(n),n,z)-f(0)*z` |

## Обратное z-преобразование — 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, ..., а $R$ — положительное число, определяющее аналитичность функции $F(z)$ вне круга $|z| = R$. Такое преобразование реализуется следующей функцией:

- `f=iztrans(F)` — возвращает результат обратного z-преобразования для скалярной символьной функции `F` независимой переменной `z`. Это преобразование определяется как `F` = $F(z) \Rightarrow$ `f` = $f(n)$. Если `F` = $F(n)$, то `iztrans` возвращает функцию `f` = $f(k)$.

Другая функция — `f=iztrans(F,k)` — дает то же, но заменяет аргумент возвращаемой функции по умолчанию `n` на `k`. Таким образом, она реализует преобразование по формуле

$$f(k) = \frac{1}{2\pi i} \oint_{|z|=R} F(z) z^{k-1} dz,$$

где $k = 1, 2, \ldots$

Наконец, функция `f=iztrans(F,w,k)` делает еще одну замену, заменяя аргумент исходной функции (по умолчанию — `z`) на `v`, реализуя соотношение

$$f(k) = \frac{1}{2\pi i} \oint_{|v|=R} F(v) v^{k-1} dv.$$

Эти преобразования можно проверить по следующим примерам:

| | | |
|---|---|---|
| `iztrans(z/(z-2))` | возвращает | `2^n` |
| `iztrans(exp(x/z),z,k)` | возвращает | `x^k/k!` |

# Символьные операции с выражениями

## 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.

## Функция расширения выражений — expand

Функция expand(S) расширяет выражения, входящие в массив S. Рациональные выражения она раскладывает на простые дроби, полиномы — на полиномиальные разложения и т. д. Функция работает со многими алгебраическими и тригонометрическими функциями. Ниже представлены примеры использования функции expand:

```
» syms a b x;
» S=[(x+2)*(x+3)*(x+4) sin(2*x)];
» expand(S)
 ans =
 [ x^3+9*x^2+26*x+24,   2*sin(x)*cos(x)]
» expand(sin(a+b))
 ans =
 sin(a)*cos(b)+cos(a)*sin(b)
» expand((a+b)^3)
 ans =
 a^3+3*a^2*b+3*a*b^2+b^3
```

## Разложение выражений на простые множители — factor

Функция factor(S) поэлементно разлагает выражения вектора S на простые множители, а целые числа — на произведение простых чисел. Следующие примеры иллюстрируют применение функции:

```
» x=sym('x');
» factor(x^7-1)
ans =
(x-1)*(x^6+x^5+x^4+x^3+x^2+x+1)
» factor(x^2-x-1)
ans =
x^2-x-1
» factor(sym('123456789'))
ans =
(3)^2*(3803)*(3607)
```

## 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 перемен-

ной, 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]
```

## 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 |

## 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
```



## 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)]
```

## 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 обо-*значения `%1`, `%2` и т. д. Пример:

```
» t=solve('a*x^3+b*x+c=0');
» [r,s]=subexpr(t,'s')
r =
[ 1/6/a*s^(1/3)-2*b/s^(1/3) ]
[ -1/12/a*s^(1/3)+b/s^(1/3)+1/2*i*3^(1/2)*(1/6/a*s^(1/3)+2*b/s^(1/3)) ]
[ -1/12/a*s^(1/3)+b/s^(1/3)-1/2*i*3^(1/2)*(1/6/a*s^(1/3)+2*b/s^(1/3)) ]
 s =
 (-108*c+12*3^(1/2)*((4*b^3+27*c^2*a)/a)^(1/2))*a^2
```

## 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 размеров 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)]
```

## 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 к f. Считается, что f — функция одной переменной, например 'x'. Тогда g(f(x)) = = x.
- g=finverse(f,v) — возвращает функцию, обратную к f, относительно заданной переменной v, так что g(f(v)) = v. Эта форма используется, если *f* — функция нескольких переменных.

Примеры:

```
» syms x
```

```
» finverse(sinh(x))
ans =
asinh(x)
» finverse(exp(x))
ans =
log(x)
```

### Суперпозиция функций — compose

К числу часто встречающихся в символьной математике манипуляций с функциями относится суперпозиция функций, реализуемая функциями compose:

- compose(f,g) — возвращает $f(g(y))$, где f = $f(x)$ и g = $g(y)$. Независимые переменные x и y находятся с помощью функции findsym.
- compose(f,g,z) — возвращает $f(g(z))$, где f = $f(x)$, g = $g(y)$.
- compose(f,g,x,z) — возвращает $f(g(z))$ и при этом рассматривает $x$ как независимую переменную для функции $f$. Так, если f = $\cos(x/t)$, то compose(f,g,x,z) возвращает $\cos(g(z)/t)$, а compose(f,g,t,z) возвращает $\cos(x/g(z))$.
- compose(f,g,x,y,z) — возвращает $f(g(z))$ и рассматривает $x$ как независимую переменную для функции $f$ и $y$ как независимую переменную для функции $g$. Для f = $\cos(x/t)$ и g = $\sin(y/u)$ compose(f,g,x,y,z) возвращает $\cos(\sin(z/u)/t)$, а compose(f,g,x,u,z) возвращает $\cos(\sin(y/z)/t)$.

Следующие примеры поясняют применение функции compose:

```
syms x y z t u;
f = 1/(1 + x^2); g = sin(y); h = x^t; p = exp(-y/u);
```

| | | |
|---|---|---|
| compose(f,g) | возвращает | 1/(1+sin(y)^2) |
| compose(f,g,t) | возвращает | 1/(1+sin(t)^2) |
| compose(h,g,x,z) | возвращает | sin(z)^t |
| compose(h,g,t,z) | возвращает | x^sin(z) |
| compose(h,p,x,y,z) | возвращает | exp(-z/u)^t |
| compose(h,p,t,u,z) | возвращает | x^exp(-y/z) |

## Специальные возможности

При практической работе с системами символьной математики зачастую необходимы взаимные преобразования различных объектов, например численных в символьные и наоборот. К примеру, если вы

аналитически получили производную функции, то для построения графика производной вам надо преобразовать формулу для нее в вычисленные численные значения. В этом разделе рассматриваются такие преобразования, введенные в пакет расширения Symbolic. Рассматриваются также специальные математические функции, калькулятор, средства графики и доступа в ядро символьных операций.



## 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 такого преобразования даны ниже:

```
» double('sin(1)')
ans =
   115   105   110    40    49    41
» double(sym('sin(1)'))
ans =
    0.8415
» double(sym('1+2*sin(1)'))
ans =
    2.6829
```

## 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 $x^3$ оказался представленным значением sin(1) с дробно-рациональным представлением.

### 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.

### 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
```

## Вычисление специальных функций

### Интегральный синус — sinint

Для вычисления интегрального синуса

$$\mathrm{Si}(z) = \int_0^t \frac{\sin(t)}{t}\,dt$$

служит функция sinint(z). Примеры ее применения:

```
» sinint(1)
ans =
```

```
    0.9461
» sinint(2+3i)
ans =
   4.5475 + 1.3992i
```

### Интегральный косинус — cosint

Интегральный косинус определяется выражением

$$\mathrm{Ci}(z) = \gamma + \ln(z) + \int_0^z \frac{\cos(t) - 1}{t} dt$$

при $|\arg(z)| < \pi$. Здесь $\gamma$ — постоянная Эйлера (0,5772...). Примеры:

```
» cosint(1)
ans =
    0.3374
» cosint(pi/4)
ans =
    0.1853
» cos(2+3i)
ans =
  -4.1896 - 9.1092i
```

### Дзета-функция Римана — zeta

Дзета-функция Римана определяется выражением:

$$\varsigma(s) = \sum_{k=1}^{\infty} k^{-s}$$

при $\mathrm{Re}(s) > 0$. Примеры ее применения показаны ниже:

```
» zeta(2)
ans =
    1.6449
» zeta(pi+2i)
ans =
   0.9794 - 0.1336i
» zeta([1 2 3])
ans =
       Inf    1.6449    1.2021
```

### 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 $K$-ю комплексную ветвь для многозначной функции. Примеры на применение этой функции:

```
» lambertw(2+3i)
ans =
   1.0901 + 0.5301i
» lambertw(3,2+3i)
ans =
  -1.6214 +18.1726i
```

### Суммы Римана — rsums

Функции `rsums(f)` и `rsums f` вычисляют приближение Римана к интегралу с подынтегральной функцией $f(x)$ и строят в виде столбцовой диаграммы график функции и площадей под кривой.

Пример применения данной функции представлен на рис. 3.1.

Обратите внимание на характерный ползунковый регулятор под графиком представления площадей. Меняя положения движка, можно изменять в широких пределах число сумм и следить за тем, насколько точно они представляют выбранную функцию.

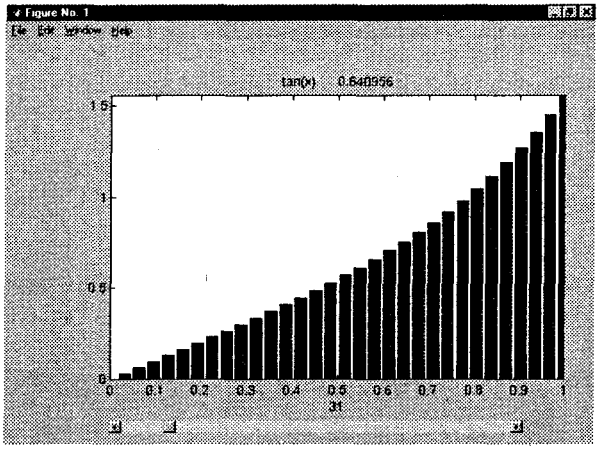


**Рис. 3.1.** Пример применения функции rsums

## Графические возможности пакета расширения Symbolic Math

### Графики символьных функций — ezplot

Чтобы избавить пользователя от возни с вполне возможным построением графиков функций с помощью стандартных средств (на-

пример, команды plot), в пакет Symbolic введены довольно удобные команды класса ezplot:

- ezplot(f) — строит график символьно заданной функции $f(x)$ независимой переменной 'x' в интервале [-2*pi, 2*pi].
- ezplot(f,xmin,xmax) или ezplot(f,[xmin,xmax]) — делает то же, но позволяет задать диапазон изменения независимой переменной x от xmin до xmax.
- ezplot(f,[xmin xmax],fig) — обеспечивает спецификацию графика с помощью параметра fig.

Команды класса ezplot позволяют строить графики функций, имеющих особенности. С примером построения графика функции $\sin(x)/x$ с особенностью при $x = 0$ мы уже знакомились в главе 1. Другой пример такого рода — построение графика функции $\tan(x)$, имеющего разрывы:

```
» ezplot('tan(x)',0,20)
» grid on
```

На рис. 3.2 показан вид построенного графика. Он выглядит намного естественнее, чем аналогичный график, построенный командой plot.

Эти команды лежат в основе специального приложения — вычислителя функций и графопостроителя Funtool, описанного ниже.

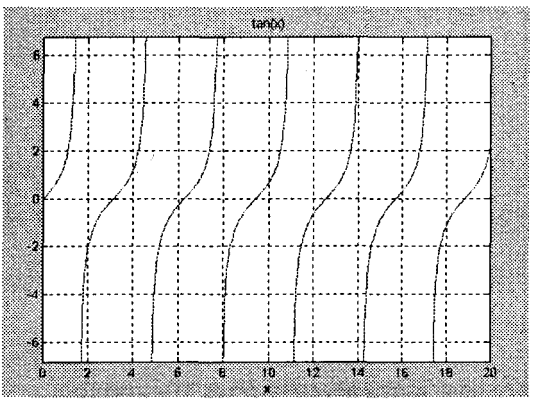


**Рис. 3.2.** График функции tan(x)

## Калькулятор 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. 3.3). 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.

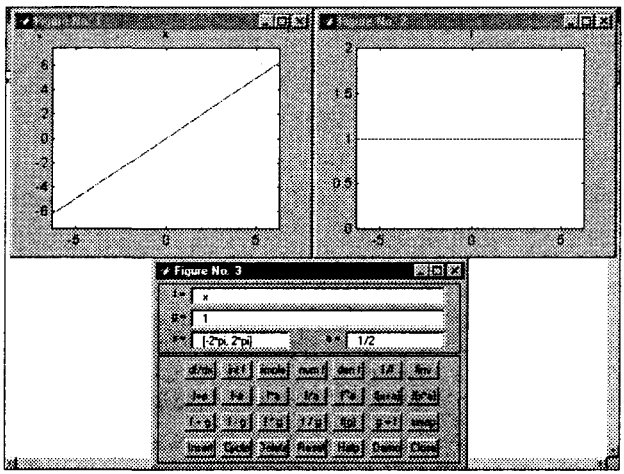


**Рис. 3.3.** Внешний вид вычислителя функций и окон двух графиков

Верхний ряд кнопок вычислителя относится только к функции $f(x)$ и задает следующие операторы:

- `df/dx` — символьное дифференцирование $f(x)$;
- `int f` — символьное интегрирование $f(x)$ при наличии замкнутой формы;
- `simple f` — упрощение выражения, если таковое возможно;
- `num f` — выделение числителя рационального выражения;
- `den f` — выделение знаменателя рационального выражения;
- `1/f` — замена $f(x)$ на $1/f(x)$;
- `finv` — замена $f(x)$ инверсной функцией.

Второй ряд клавиш выполняет операции масштабирования и сдвига $f(x)$ с применением параметра `'a'`:

- `f + a` — заменяет $f(x)$ на $f(x) + a$;
- `f - a` — заменяет $f(x)$ на $f(x) - a$;
- `f * a` — заменяет $f(x)$ на $f(x)\,a$;
- `f / a` — заменяет $f(x)$ на $f(x) / a$;
- `f ^ a` — заменяет $f(x)$ на $f(x) \wedge a$;
- `f(x+a)` — заменяет $f(x)$ на $f(x + a)$;
- `f(x*a)` — заменяет $f(x)$ на $f(x\,a)$.

Третий ряд клавиш предназначен для осуществления бинарных операций над функциями $f(x)$ и $g(x)$:

- `f + g` — заменяет $f(x)$ на $f(x) + g(x)$;
- `f - g` — заменяет $f(x)$ на $f(x) - g(x)$;
- `f * g` — заменяет $f(x)$ на $f(x)\,g(x)$;
- `f / g` — заменяет $f(x)$ на $f(x) / g(x)$;
- `f(g)` — заменяет $f(x)$ на $f(g(x))$;
- `g = f` — заменяет $g(x)$ на $f(x)$;
- `swap` — меняет $f(x)$ и $g(x)$ местами.

Калькулятор имеет особую память для функций в виде списка `fxlist`. Четвертый ряд клавиш служит для работы с памятью калькулятора и иных операций:

- `Insert` — помещает текущую функцию в список функций.
- `Cycle` — выполняет текущую функцию из списка.
- `Delete` — удаляет выделенную функцию из списка.
- `Reset` — устанавливает `f`, `g`, `x`, `a` и `fxlist` в исходное состояние.
- `Help` — выводит описание калькулятора.
- `Demo` — запускает демонстрационный пример.
- `Close` — завершает 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. 3.4 показано построение графика функции $f(x) = \sin(x)/x$ и ее производной — графика $g(x)$. Для получения этих графиков потребовался ввод $f(x)$, нажатие клавиши `df/dx` и клавиши `swap`.

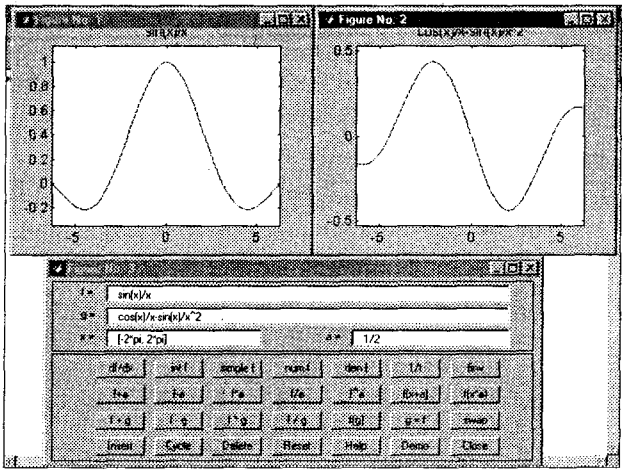


**Рис. 3.4.** Графики функции sin(x)/x и ее производной

Кнопка Cycle позволяет просмотреть графики ряда интересных функций (рис. 3.5) 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 интересующие вас функции.

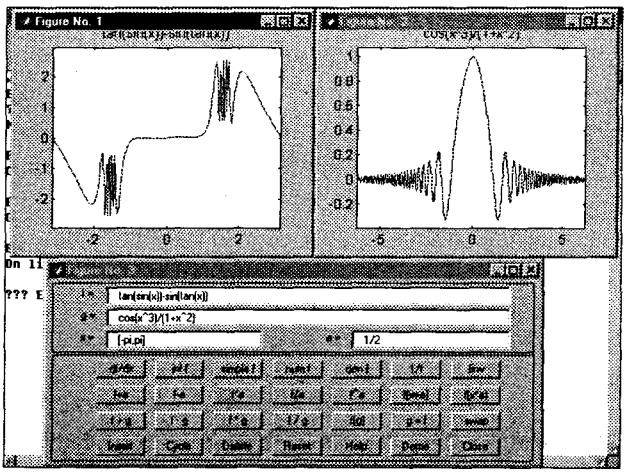


**Рис. 3.5.** Фрагмент демонстрации построения функций из списка fxlist

Вычислитель и графопостроитель и впрямь является весьма удобным средством визуализации графиков самых различных функций. Эта утилита заменяет множество объемных книг по графикам элементарных и специальных функций.

## Контурные графики — ezcontour

Помимо упомянутых графических функций пакет Symbolic поддерживает построение графиков различных типов. Так, функция ezcontour служит для построения контурных графиков функций вида $f(x, y)$. Эта функция записывается в следующем виде:

- `ezcontour(f)` — строит контурный график с настройкой по умолчанию;
- `ezcontour(f,domain)` — строит контурный график с заданными параметром `domain` и пределами изменения $x$ и $y$;
- `ezcontour(…,n)` — обеспечивает ранее указанные построения при явном задании числа линий `n`.

Следующий пример:

```
» syms x y
» ezcontour(sin(x*y),[-3,3],30)
```

строит контурный график функции $\sin(xy)$, представленный на рис. 3.6.

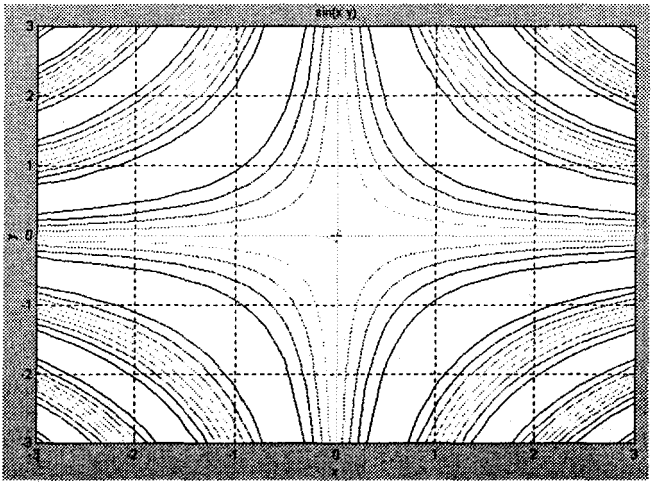


**Рис. 3.6.** Пример построения контурного графика командой ezcontour

## Контурные графики с закраской — ezcontourf

Похожая на рассмотренную выше функция `ezcontourf` строит контурные графики с функциональной окраской (fill) областей между линиями равного уровня:

- `ezcontourf(f)` — строит контурный график с окраской и настройкой по умолчанию;
- `ezcontourf(f,domain)` — строит контурный график с окраской с заданными параметром `domain` и пределами изменения $x$ и $y$;
- `ezcontourf(...,n)` — обеспечивает ранее указанные построения при явном задании числа линий `n`.



Следующий пример иллюстрирует применение этой функции для построения контурного графика выражения $\sin(x)\sin(y)$:

```
» syms x y
» ezcontourf(sin(x)*sin(y),[-3,3],50)
```

График данной функции представлен на рис. 3.7.

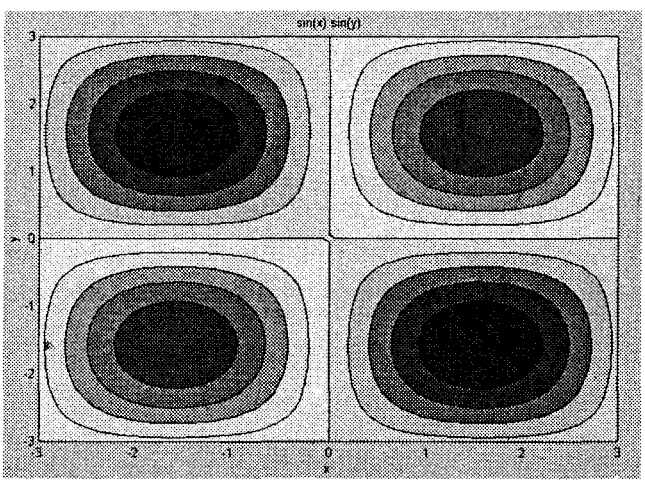


**Рис. 3.7.** Пример построения контурного графика с функциональной окраской командой ezcontourf

## Трехмерные графики параметрически заданных функций — ezplot3

Для построения трехмерных графиков параметрически заданных функций служит команда `ezplot3`:

- `ezplot3(x,y,z)` — строит трехмерный график функции, заданной параметрически уравнениями $x(t)$, $y(t)$, $z(t)$ при настройке по умолчанию;
- `ezplot3(x,y,z,[tmin tmax])` — строит трехмерный график функции, заданной параметрически уравнениями $x(t)$, $y(t)$, $z(t)$ при изменении аргумента $t$ от `tmin` до `tmax`;
- `ezplot3(...,'animate')` — аналогична предшествующим командам, но обеспечивает анимацию при построении графика.



Следующий пример показывает, как можно построить пространственную спираль с элементами анимации:

```
» syms t; ezplot3(cos(t),sin(t),t,[0 20],'animate')
```

Заключительная стадия построения показана на рис. 3.8. При наличии опции `'animate'` по спирали движется шарик, который также можно видеть на рис. 3.8. Можно повторить движение шарика, нажав кнопку Repeat в графическом окне. Эти кнопка и шарик отсутствуют при отсутствии опции `'animate'`.

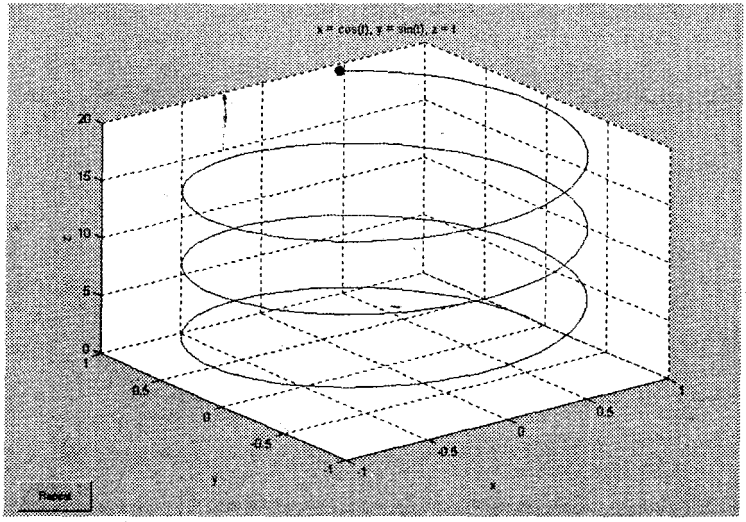


**Рис. 3.8.** Построение объемной спирали и движущегося по ней шарика

## Полярный график — команда ezpolar

*График функции $f(t)$ в полярной системе координат строит команда* `ezpolar`:

- `ezpolar(f)` — строит график функции $f(t)$ в полярной системе координат при изменении угла $t$ от 0 до $2\pi$;
- `ezpolar(f,[a b])` — строит график функции $f(t)$ в полярной системе координат при изменении угла $t$ от $a$ до $b$.

Применение этой команды иллюстрирует следующий пример:

```
» syms t
» ezpolar(sin(2*t))
```

Окно с построенным графиком представлено на рис. 3.9.



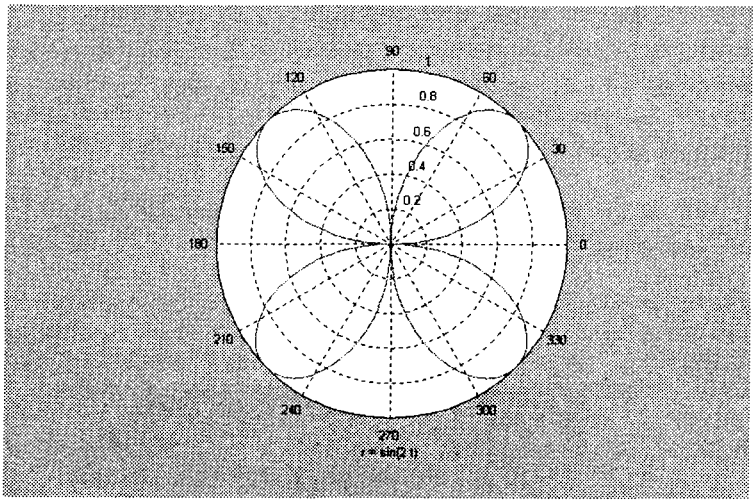


**Рис. 3.9.** Пример построения графика в полярной системе координат командой ezpolar

## Графики поверхностей — ezsurf и ezsurfc

Команда `ezsurf` служит для построения графиков поверхностей, задаваемых функциями двух переменных $f(x, y)$:

- `ezsurf(f)` — построение поверхности $f(x,y)$ с параметрами $x$ и $y$, меняющимися по умолчанию от $-2\pi$ до $2\pi$;
- `ezsurf(f,domain)` — построение поверхности $f(x,y)$ с пределами изменения $x$ и $y$, заданными параметром `domain`;
- `ezsurf(x,y,z)` — построение поверхности, заданной параметрически зависимостями $x(s,t)$, $y(s,t)$, $z(s,t)$ при $s$ и $t$, меняющихся в интервале от $-2\pi$ до $2\pi$;

- `ezsurf(x,y,z,[smin, smax, tmin, tmax])` — построение поверхности, заданной параметрически зависимостями $x(s,t)$, $y(s,t)$, $z(s,t)$ при $s$ и $t$, меняющихся в заданном интервале;
- `ezsurf(x,y,z,[min, max])` — построение поверхности, заданной параметрически зависимостями $x(s,t)$, $y(s,t)$, $z(s,t)$ при $s$ и $t$, меняющихся в одинаковом интервале от `min` до `max`;
- `ezsurf(…,n)` — аналогична описанным выше командам, но с заданным числом линий сетки `n`;
- `ezsurf(…,'circ')` — аналогична описанным выше командам, но вписывает поверхность в окружность.

Следующий пример показывает действие этой команды:

```
» syms x y
» ezsurf(real(asec(x+i*y)))
```

Построенный в этом примере график представлен на рис. 3.10.

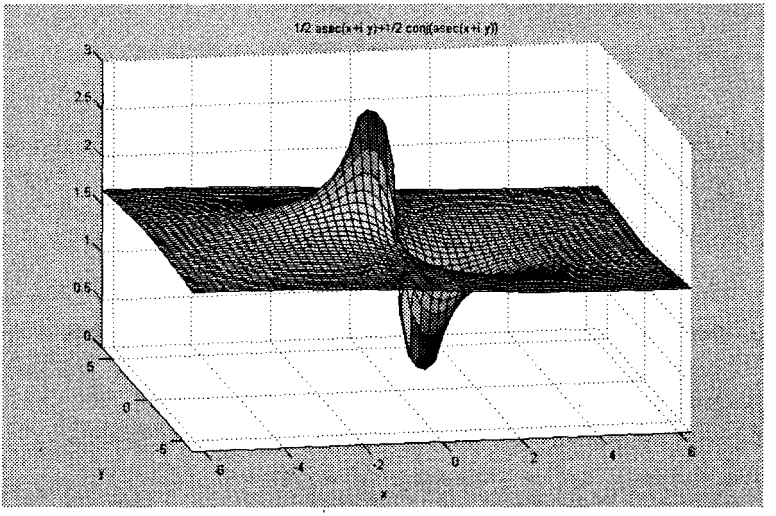


**Рис. 3.10.** Пример построения поверхности командой ezsurf

Есть совершенно аналогичная по синтаксису записи группа команд `ezsurfc`. В отличие от предшествующей группы команд в этом случае строится еще и контурный график поверхности на плоскости, лежащей под поверхностью. Следующий пример поясняет такое построение:

```
» syms x y
» ezsurfc(real(asec(x+i*y)))
```

Сами графики (поверхности в пространстве и контурный на плоскости) показаны на рис. 3.11.

Описанные выше команды графики похожи на соответствующие команды MATLAB. Однако благодаря применению средств символьной математики они задаются в более естественной форме и обычно упрощают построение сложных математических графиков.



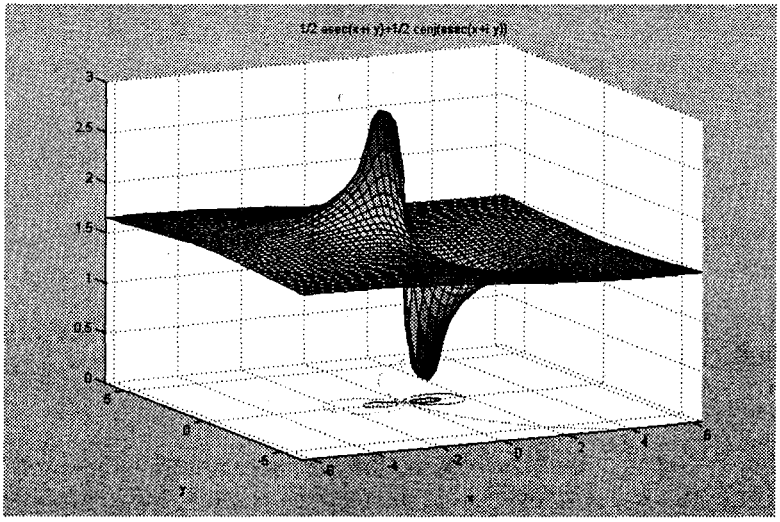

**Рис. 3.11.** Пример построения поверхности командой ezsurfc и контурного графика под поверхностью

# 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 MATLAB эти возможности отсутствуют.

Применение возможностей системы Maple V R4 совместно с возможностями системы MATLAB придает последней особую гибкость и резко расширяет возможности решения сложных математических задач, где целесообразно объединять аналитические (символьные) методы с численными вычислениями.

## 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 команд Maple. Возвращает результат в форме строки с нотацией системы Maple V.
- `maple('function',ARG1,ARG2,...)` — дает доступ к Maple-функциям с заданными аргументами.
- `[RESULT,STATUS]=maple(...)` — возвращает результат и статус сообщений/ошибок, если они возникают в ходе вычисления заданной функции.

Примеры:

```
» maple('sin(1)')
ans =
sin(1)
» maple('evalf(sin(1))')
ans =
.84147098480789650665250232163030
» maple('sqrt',2)
ans =
2^(1/2)
» maple('evalf(sin)',1)
ans =
sin(1)
```

## Численное вычисление Maple-функций — mfun

Для вычисления числовых значений функций ядра системы Maple используется функция `mfun`:

- `mfun('fun',p1,p2,...,pk)` — возвращает в численном виде значение Maple-функции с именем `'fun'` и списком параметров функции `p1`, `p2`, ..., `pk`. Если, вычисление функции невозможно (например, из-за сингулярных точек), то возвращается константа `NaN`.

```
» mfun('sin',1)
ans =
    0.8415
```

```
» mfun('sqrt',2+3i)
ans =
   1.6741 + 0.8960i
» mfun('ln',0)
ans =
   NaN
» mfun('ln',2)
ans =
    0.6931
```



## 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
```

## Получение справки по ядру Maple V — mhelp

Для получения справки по Maple-функциям непосредственно из среды MATLAB служат команда и функция `mhelp`:

`mhelp topic` или `mhelp('topic')`

Пример:

```
» help LCM
 LCM   Least common multiple.
    LCM(A,B) is the least common multiple of corresponding elements
    of A and B.  The arrays A and B must contain positive integers
    and must be the same size (or either can be scalar).

    See also GCD.
```

## Инсталляция Maple-процедур — procread

С помощью команды `procread(FILENAME)` можно готовить процедуры с синтаксисом языка системы Maple V R4. Например, подготовим

процедуру с предполагаемым именем файла check.src и следующим текстом:

```
check := proc(A)
    #   check(A) computes A*inverse(A)
    local X;
    X := inverse(A):
    evalm(A &* X);
end;
```

Создав процедуру, можно проинсталлировать ее с помощью команды

```
procread('check.src')
```

После этого возможно использование этой процедуры в виде

```
maple('check',magic(3))
```

или

```
maple('check',vpa(magic(3)))
```

Применение процедур с синтаксисом Maple расширяет возможности программирования системы MATLAB и свидетельствует об углублении процесса интеграции систем MATLAB и Maple.

# Глава 4
# Пакет расширения Simulink

## Назначение пакета Simulink

В состав расширенных версий системы MATLAB входит пакет моделирования динамических систем — Simulink. В MATLAB 5.3.1 используется последняя (во время подготовки книги) версия этого пакета — Simulink 3.1. Она, за редкими исключениями, совместима с предшествующими версиями 1.0, 1.3 и 2.0, сыгравшими большую роль в популярности этого уникального пакета. С Simulink органично связан целый ряд других пакетов, краткое описание которых приводится в данном разделе.

Пакет Simulink является ядром интерактивного программного комплекса, предназначенного для *математического моделирования* линейных и нелинейных динамических систем и устройств, представленных своей функциональной блок-схемой, именуемой *S-моделью* или просто *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 т. д.

Полное описание пакета Simulink значительно превышает объем данной книги, так что приведенные далее 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 уверенной работы с ним.

Для построения функциональной блок-схемы моделируемых устройств Simulink имеет обширную *библиотеку* блочных компонентов и удобный *редактор блок-схем*. Он основан на графическом интерфейсе 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 системы или устройства.

Simulink автоматизирует следующий, наиболее трудоемкий этап моделирования: он составляет и решает сложные системы алгебраических и дифференциальных уравнений, 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 негативный оттенок, так что мы будем использовать термин «моделирование».



Средства визуализации результатов моделирования в пакете Simulink настолько наглядны, что порой создается ощущение, что 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 т. д.

В библиотеке имеется целый набор виртуальных регистрирующих устройств — от простых измерителей типа вольтметра или амперметра до универсальных осциллографов, позволяющих просматривать временные зависимости выходных параметров моделируемых систем — например, токов и напряжений, перемещений, давлений и т. п. 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, к ним относятся, например, программа Tutsim или

LabVIEW for Industrial Automation. Однако для эффективного применения таких средств необходимы высокоскоростные решающие устройства. Интеграция одной из самых быстрых матричных 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 в 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 применения Simulink для решения практически любых задач имитационного моделирования. Наиболее важные пакеты кратко описаны ниже, а в дальнейшем будут рассмотрены и примеры их применения.

## Новые возможности Simulink 3.1

В новую версию пакета Simulink 3.1, интегрированную с MATLAB 5.3.1, добавлены следующие возможности:

- интегрированный броузер моделей (Windows 95/98/NT);
- возможность увеличения блок-схем (zooming);
- блок Scope может работать с многими портами и осями координат;
- интегрированные возможности линейного анализа;
- графический интерфейс для описания свойств сигнала;
- интегрированный броузер библиотек (только Windows 95/98/NT);
- новые блоки Subsystem, Round Sum, Enhanced Mux, Bus Selector и Model Info;
- 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.

## 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 интегрированного броузера библиотеки, показанное в левой части рис. 4.1.

Нетрудно заметить, что пользовательский интерфейс окна броузера выполнен в общем стиле, характерном для Проводника Windows 95/98. Это позволяет отказаться от детального описания его особенностей. Отметим лишь главные возможности работы с броузером.

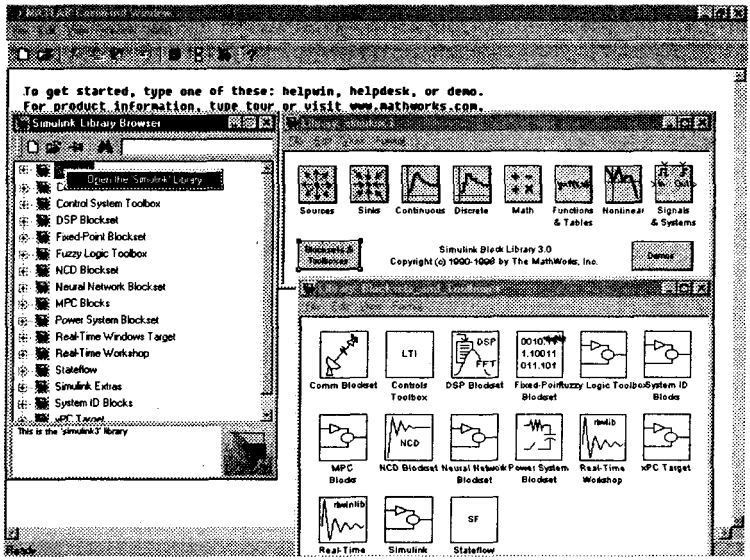


**Рис. 4.1.** Интеграция средств пакета Simulink с системой MATLAB

В окне броузера видно дерево компонентов, которые содержит библиотека компонентов Simulink. Для просмотра раздела библиотеки достаточно выделить его мышью и нажать ее правую кнопку. Появится контекстно-зависимое меню, содержащее единственную команду — команду открытия окна соответствующего раздела библиотеки. Исполнив эту команду, можно получить окно с набором графических отображений всех компонентов заданного раздела. На рис. 4.1 показан пример вывода главного окна библиотеки Simulink.

С помощью главного меню броузера или кнопок его панели инструментов можно открыть окно для задания новой модели или окно для загрузки примеров применения системы Simulink. Работа с Simulink происходит на фоне открытого окна системы MATLAB и в последнем нередко можно наблюдать за выполняемыми операциями — если их вывод предусмотрен программой моделирования.

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 (а также ее язык программирования).

## 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 меню Simulation 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).

## Особенности интерфейса Simulink

Интерфейс версии Simulink 2.0 полностью соответствует стилю интерфейса типовых приложений Windows 95/98, в том числе интер-

фейсу системы MATLAB. В то же время он концептуально строг, дабы не досаждать пользователю многочисленными «излишествами» 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 обмена и создание подсистем.
- View — вывод или удаление панели инструментов и строки состояния.
- Simulation — управление процессом моделирования (старт, пауза, вывод окна настройки параметров моделирования).
- Format — операции форматирования модели (смена шрифтов, редактирование надписей, повороты блоков, использование тени от блоков, операции с цветами линий блоков, их фоном и общим фоном.
- Tools — управление видом анализа (в линейной области и в режиме реального времени RTW).



Первые три 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 открывает доступ к множеству других подобных окон для новых пакетов компонентов (Blocksets&Toolboxes) и примеров их применения (Demos). Это дает пользователю возможность постепенно знакомиться с новыми областями применения Simulink.

## Демонстрация возможностей Simulink

Даже незнакомый с Simulink пользователь может быстро оценить возможности этого пакета, воспользовавшись множеством интересных и поучительных примеров. 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 меню File основного окна броузера библиотеки Simulink. В уже зна-

комом вам окне с названием Open Model (Открыть модель) надо войти в указанную директорию и выбрать подходящий пример (рис. 4.2).

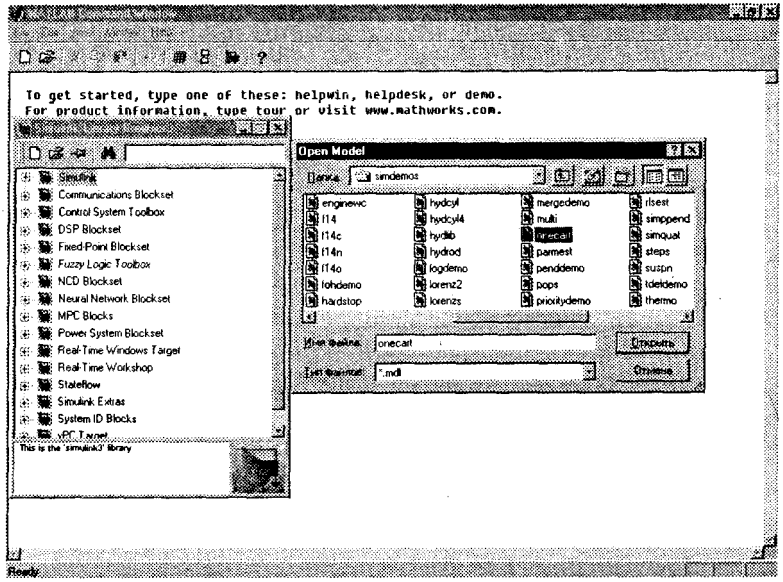


**Рис. 4.2.** Окно с полным перечнем файлов демонстрационных примеров пакета Simulink

Выбрав нужный пример, нужно нажать кнопку Открыть. Появится окно редактирования графической модели моделируемого устройства. На рис. 4.3 справа от окна броузера библиотеки показано основное окно редактирования выбранной нами модели, позволяющее 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 о колебаниях реального механического маятника под действием внешней силы.

Как можно заметить, графическая модель этого примера содержит 11 блоков. Каждый блок имеет наглядное общепринятое обозначение в виде прямоугольника, треугольника и т. д. Блоки имеют входы и выходы и описываются различными математическими зависимостями. Блоки соединяются друг с другом линиями — стрелками,

причем стрелка указывает направление от выходов одних блоков ко входам других. Имеются также текстовые комментарии и средства для вывода подсказок и открытия окон справочной системы.

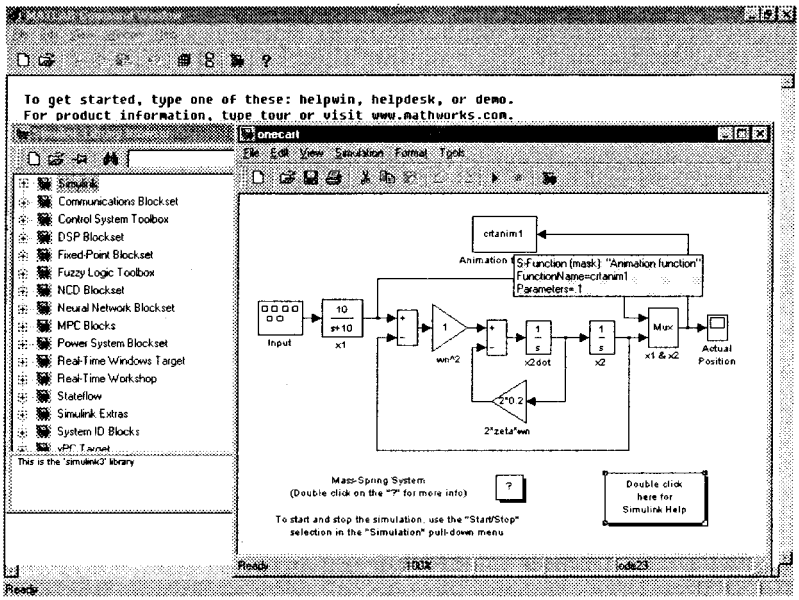


**Рис. 4.3.** Графическая модель колебательной системы – массивного кубика на пружине

В конце инструментальной панели Simulink находятся две важные кнопки управления моделированием. Одна из них, в виде черного треугольника (Start/ Pause Simulation), запускает или приостанавливает начатый процесс моделирования, а другая, в виде черного квадратика (Stop), останавливает его. Все, что нужно для запуска моделирования, — это нажать кнопку с изображением треугольника. Рисунок 4.4 показывает результат моделирования для выбранной модели. Вместо кнопок можно использовать команды Start и Pause в меню Simulation окна модели.

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.

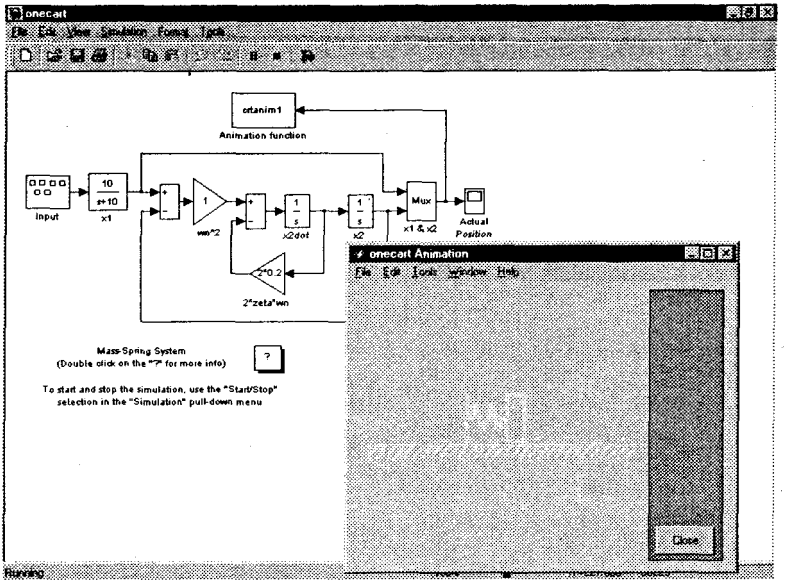


**Рис. 4.4.** Кадр моделирования поведения массивного кубика, через пружину привязанного к палочке

# Запуск моделей Simulink из среды MATLAB

Обычно Simulink запускается соответствующей кнопкой из панели инструментов, после чего все последующие действия выполняются в среде Windows. Можно также запустить Simulink, исполнив в командной строке окна MATLAB команду

```
» simulink
```

Для вывода полного перечня команд Simulink надо выполнить команду

```
» help simulink
```

Вывод будет дан в следующей форме (дано несколько команд):

```
Simulink
Version 3.0.1 (R11.1) 10-Sep-1999

Model analysis and construction functions.

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 trimming.
    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.

  Hardcopy and printing.
    frameedit     - Edit print frames for annotated model printouts.
    print         - Print graph or Simulink system; or save graph to
                    M-file.
    printopt      - Printer defaults.
    orient        - Set paper orientation.

  See also BLOCKS and SIMDEMOS.
```

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.

# Библиотека компонентов пакета Simulink

## Основная палитра компонентов

Версия симулятора Simulink 3.1 имеет существенно обновленную и расширенную 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 (см. рис. 4.1). На рис. 4.5 представлено отдельно это окно. 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.

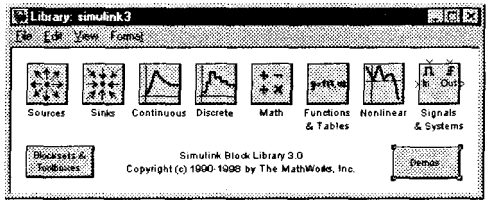


**Рис. 4.5.** Оглавление библиотеки компонентов (блоков) системы Simulink

Как видно из рис. 4.5, 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;
- Continious — открытие окна с перечнем линейных компонентов;
- Discrete — открытие окна с перечнем дискретных компонентов;
- Math — открытие окна с перечнем математических компонентов;
- Functions&Tables — открытие окна с перечнем функций и табличных компонентов;
- Nonlinear — открытие окна с перечнем нелинейных компонентов;
- Connections — открытие окна с перечнем подключающих компонентов;
- Signals&Systems — открытие окна с перечнем сигнальных и системных 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 строку состоя-

ния. С его помощью можно вводить и загружать из файла модели моделируемых устройств и систем. Окно можно свернуть или закрыть. Если после закрытия окна основной библиотеки оно понадобится вновь, то из основного окна библиотеки надо загрузить файл simulink, который находится в директории MATLAB/TOOLBOX/SIMULINK/BLOCK10.

## 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 т. д. Окно Sources представлено на рис. 4.6.



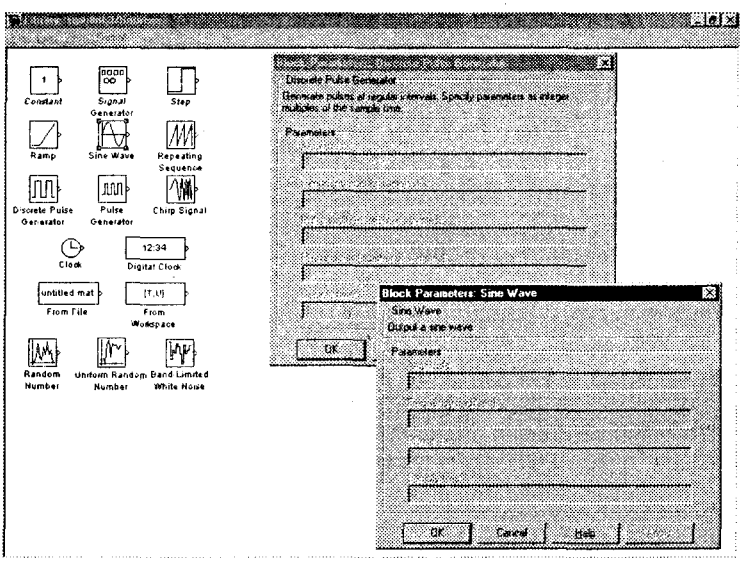


**Рис. 4.6.** Окно с перечнем источников сигналов и воздействий

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 т. д.

Набор блоков содержит практически все часто используемые при моделировании источники воздействий с самыми различными функциональными и временными зависимостями. 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. 4.6 справа от окна выбора источников. Для открытия этого окна достаточно выполнить двойной щелчок на изображении нужного элемента. Под этим окном показано также окно установки параметров синусоидального источника Sine Wave.

*Естественно, что таких окон множество, как и самих графических* элементов. Тем не менее для пользователей, имеющих хотя бы начальные представления об имитационном моделировании систем, установка параметров графических элементов не вызывает трудностей, *поскольку названия большинства параметров вполне очевидны.* К примеру, для блока Sine Wave устанавливаются амплитуда синусоиды, ее частота, фаза и время действия. Словом, задаются типичные параметры сигнала (воздействия).

При вызове окон параметров путем активизации графических элементов в окнах библиотек отображаются установки параметров по умолчанию. Как правило, они нормализованы — например, заданы единичная частота, единичная амплитуда, нулевая фаза и т. д. 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.

## Регистрирующие элементы

Регистрирующие элементы — важный фактор качественной визуализации *результатов моделирования. На рис. 4.7 показано окно* виртуальных регистраторов пакета Simulink. Некоторые регистраторы выполнены в виде, весьма похожем на реальные приборы.

Каждый регистратор имеет свое окно настройки, выводимое при активизации его пиктограммы в окне компонентов или в окне моделей. В качестве примера на рис. 4.7 показаны окна настройки двух компонентов — графопостроителя и дисплея.

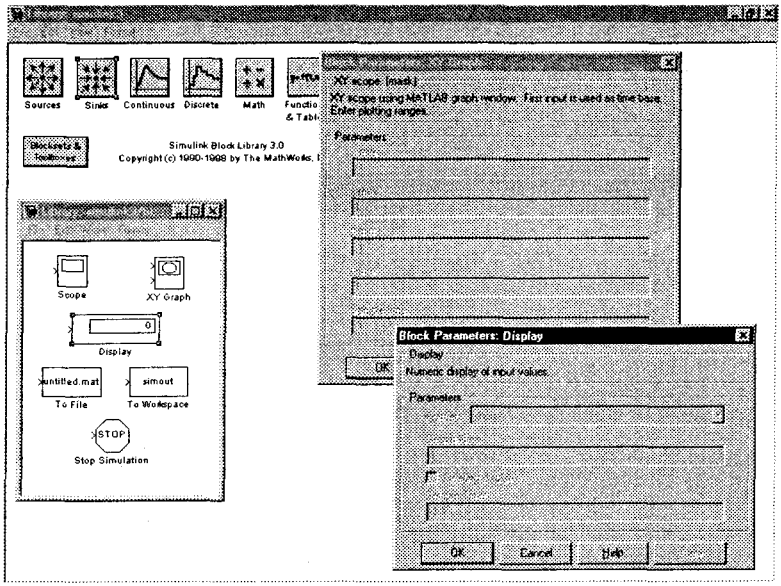


**Рис. 4.7.** Окно регистраторов пакета Simulink и окна настройки графопостроителя и дисплея



В состав виртуальных регистраторов входят:

- Scope — осциллограф 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 записи в переменную рабочего пространства;
- Stop Simulation — блок остановки моделирования.

Важно отметить, что виртуальные регистраторы фиксируют параметры любого типа, а не только электрические. Это придает некоторым виртуальным регистраторам (приборам) уникальный характер. Например, об осциллографе, 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.

## Дискретные компоненты

Дискретные компоненты включают в себя устройства задержки, дискретно-временной интегратор, дискретный фильтр и т. п. На рис. 4.8 представлено окно с этими устройствами.

На рис. 4.8 показаны также окна установки параметров ряда дискретных компонентов. Они позволяют судить о том, какие параметры компонентов могут устанавливаться и изменяться.

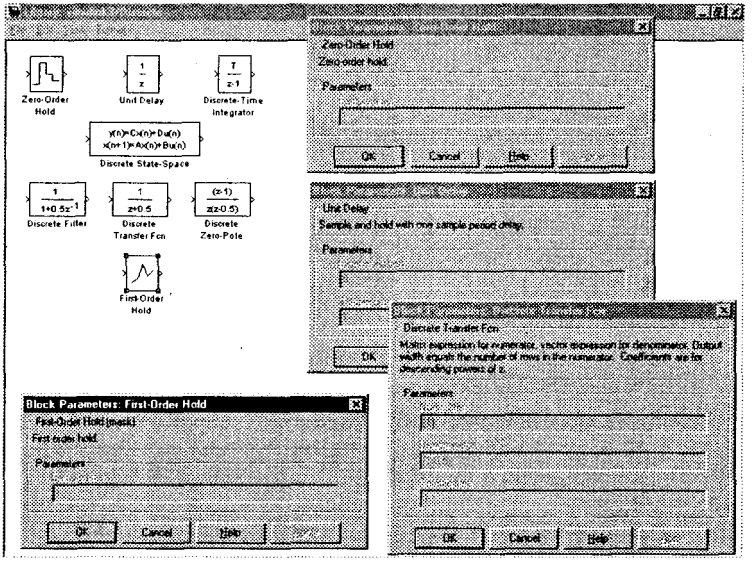


**Рис. 4.8.** Окно дискретных компонентов и окна настройки некоторых из них

## 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. 4.9 представлено окно с линейными компонентами пакета Simulink — окно Continuous.

Как видно из рис. 4.9, 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. На рис. 4.9 показаны также окна настройки трех линейных компонентов — интегратора, дифференциатора и блока задания временной задержки.

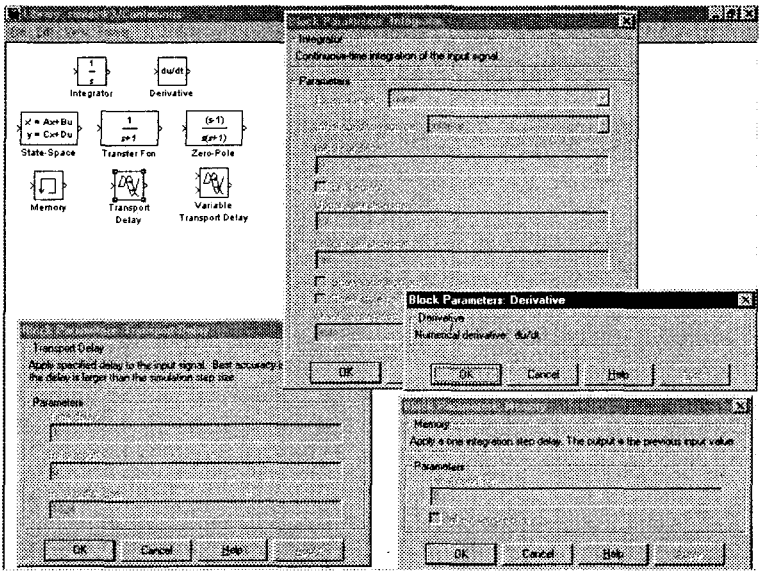


**Рис. 4.9.** Окно с линейными компонентами

## Нелинейные компоненты

Поскольку Simulink предназначен главным образом для моделирования нелинейных динамических систем, раздел, посвященный нелинейным компонентам, впечатляет уже их обилием (рис. 4.10).

Среди нелинейных компонентов следует отметить компоненты с типичными нелинейностями, например вида `abs(u)`, с характеристиками, описывающимися типовыми математическими функциями, компонентами идеальных и неидеальных ограничителей и т. д. 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, на рис. 4.10 представлены окна настройки пяти компонентов, дающие достаточное представление об их характерных параметрах.

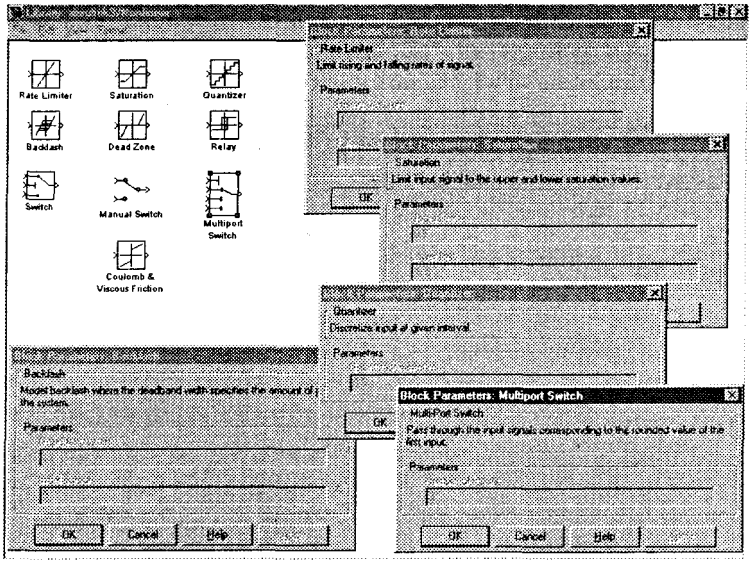


**Рис. 4.10.** Окно библиотеки с нелинейными компонентами и окна настройки пяти компонентов

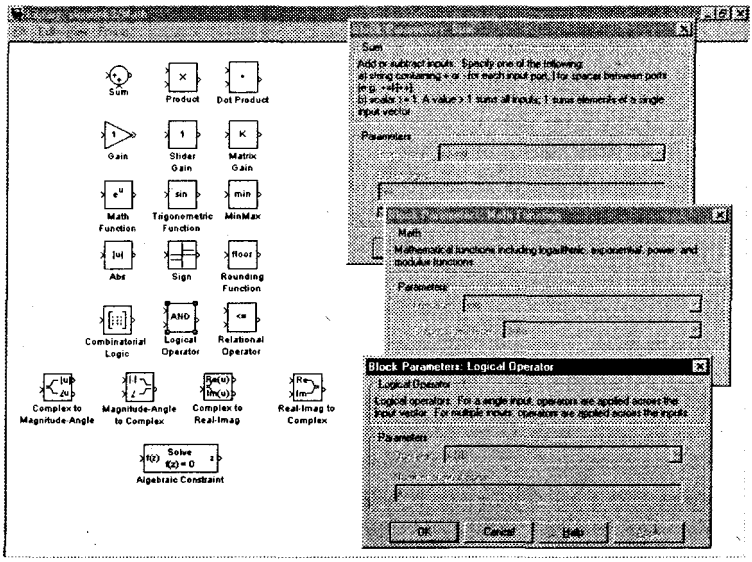


**Рис. 4.11.** Окно библиотеки математических компонентов и окна настройки трех компонентов

## Математические компоненты

Новым разделом библиотеки Simulink 3.1 стали математические компоненты. Окно Math с ними представлено на рис. 4.11. В этом окне показаны окна настройки трех математических компонентов — аналогового сумматора, математической функции пользователя и логического оператора AND.

Возможность задания множества математических компонентов с настраиваемыми свойствами играет решающее значение для выполнения прозрачного для пользователя математического моделирования как простых, так и сложных устройств и систем.

## Подключающие компоненты

Окно с подключающими компонентами Signals&Systems показано на рис. 4.12. Оно также содержит впечатляющий набор таких компонентов — от портов входа In, выхода Out и заземления Ground до компонентов, имитирующих работу триггера (Trig) и даже задания подсистем Subsystem. Последний компонент представляет собой пустое окно, в котором можно создать функциональную схему, рассматриваемую как подсистему (блок). Такая подсистема может многократно использоваться различными моделями.

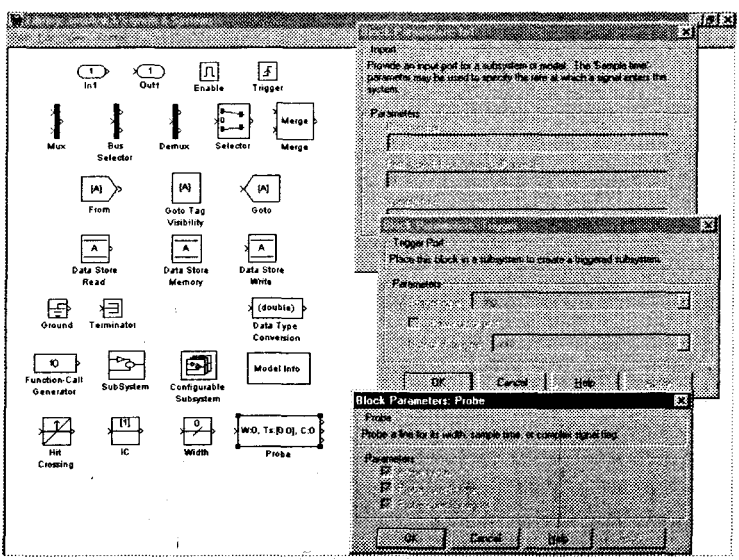


**Рис. 4.12.** Окно библиотеки с подключающими компонентами и окна настройки трех компонентов

Каждый компонент, как обычно, имеет окно установки своих параметров. На рис. 4.12 представлены такие окна для трех компонентов — порта ввода, триггера и блока Probe.

## Компоненты функций и таблиц

Еще один новый раздел библиотеки — компоненты функций и таблиц. Окно Functions&Tables с пятью компонентами этой группы представлено на рис. 4.13. Окна настройки параметров всех этих компонентов также представлены на рис. 4.13.

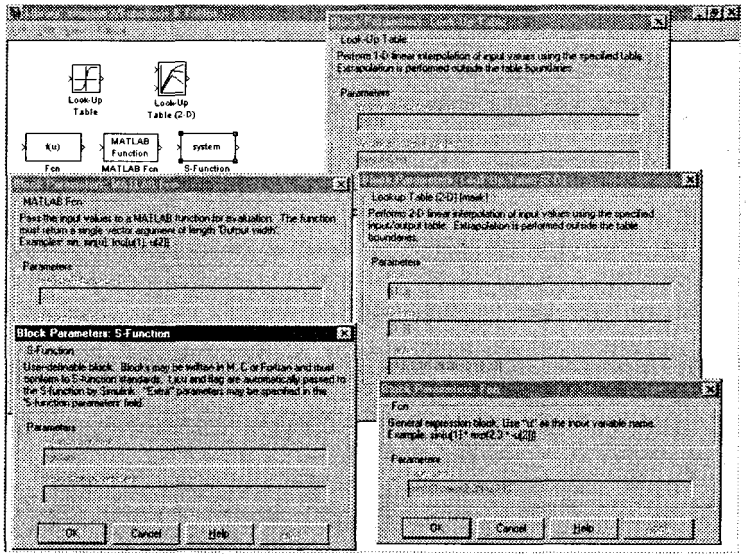


**Рис. 4.13.** Окно с компонентами функций и таблиц и окна их настройки

## 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. 4.14). Каждый ее подраздел (соответствующий рисунок) открывает дополнительное окно с новым перечнем компонентов. Для примера на рис. 4.14 открыты окна трех подразделов.

Представленные в этих разделах компоненты относятся к достаточно серьезным приложениям системы MATLAB, причем подчас вполне законченным. В качестве примера на рис. 4.15 показан последо-

вательный 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.

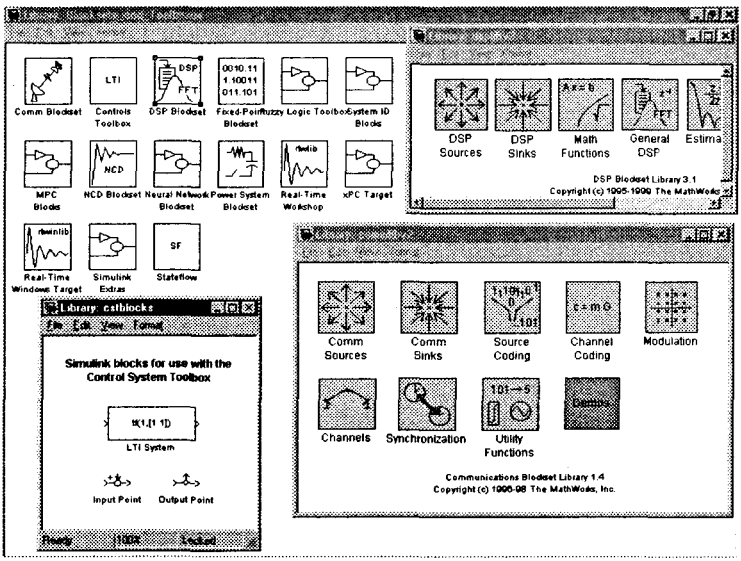


**Рис. 4.14.** Окно дополнительных подразделов библиотеки компонентов и окна трех подразделов

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).

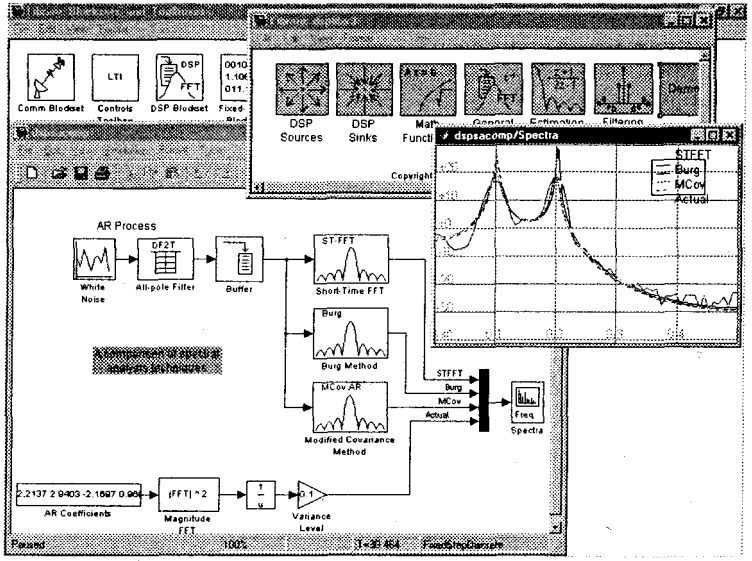


**Рис. 4.15.** Пример поиска и исполнения примера спектрального анализа

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 3.1 входит множество библиотечных компонентов, с лихвой удовлетворяющих большинство пользователей, всерьез занимающихся математическим моделированием систем и устройств. При этом есть возможность корректировать описания компонентов и вводить новые компоненты — в том числе в виде функциональных схем отдельных подсистем. В версии Simulink 3.1, включенной в систему MATLAB 5.3.1, набор компонентов существенно расширен и уменьшено время моделирования систем со сложными моделями. Однако весь приведенный далее материал полностью справедлив и для предшествующих версий пакета Simulink, начиная с входящей в систему MATLAB 5.0.

# Основы работы

## Постановка задачи — моделирование ограничителя

Решение любой проблемы в системе Simulink должно начинаться с постановки задачи. Чем глубже продумана постановка, тем больше

вероятность успешного решения задачи. В ходе постановки нужно оценить, насколько суть задачи отвечает возможностям пакета Simulink и какие компоненты последнего могут использоваться для построения модели.

В качестве примера рассмотрим тривиальную задачу моделирования работы идеального ограничителя, на вход которого подается синусоидальное напряжение с амплитудой 10 В и частотой 100 рад/с. 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 — осциллограф. Так что функциональная схема моделируемого устройства в данном случае вполне очевидна, и мы можем перейти к ее реализации.



## Создание модели устройства (системы)

Создание модели в нашем примере начинается с активизации кнопки Simulink на панели инструментов окна MATLB (рис. 4.16). При этом откроется окно броузера библиотеки компонентов, представленное на рис. 4.16 слева в окне MATLAB.

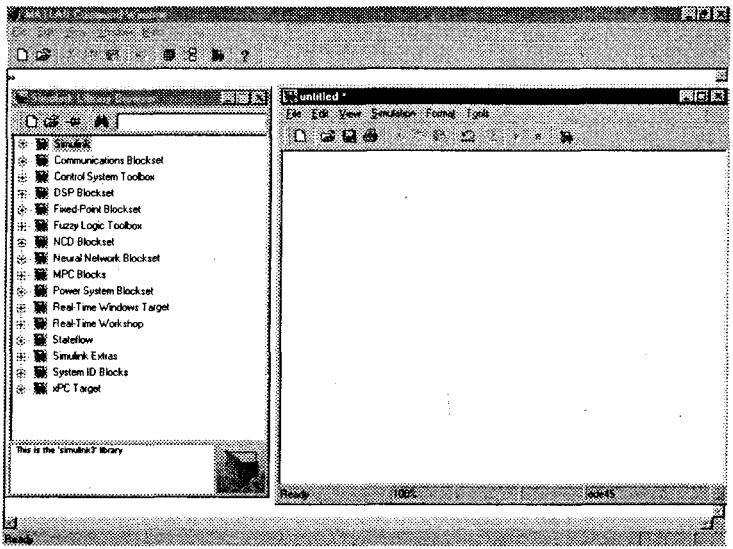


**Рис. 4.16.** Подготовка к созданию модели

В окне броузера библиотеки надо нажать кнопку New (она первая в панели инструментов и имеет пиктограмму в виде чистого листа). Появится пустое окно редактирования модели, показанное на рис. 4.16 справа. В этом окне мы и будем создавать нашу модель.

Следующий этап — выбор источника сигнала. Тут есть две возможности — выбор нужного компонента прямо из дерева компонентов броузера библиотеки или выбор его из открытого окна нужного подраздела библиотеки. Процесс выбора и открытия окон подразделов библиотеки был детально описан ранее, поэтому изберем первую возможность и покажем, как пользоваться новым средством — броузером библиотеки.

Вначале надо удобно разместить окно броузера и окно редактирования модели. Целесообразно располагать их рядом, как на рис. 4.16. Активизировав мышью прямоугольник со знаком «плюс» перед оглавлением основной библиотеки Simulink, можно открыть ветвь с этим разделом. В нем надо аналогичным образом открыть подраздел источников Sources. Затем, активизировав в нем компонент Sine Wave (синусоидальный источник), надо мышью (при нажатой левой кнопке) просто перетащить его в окно редактирования модели — рис. 4.17 показывает результат этого действия.

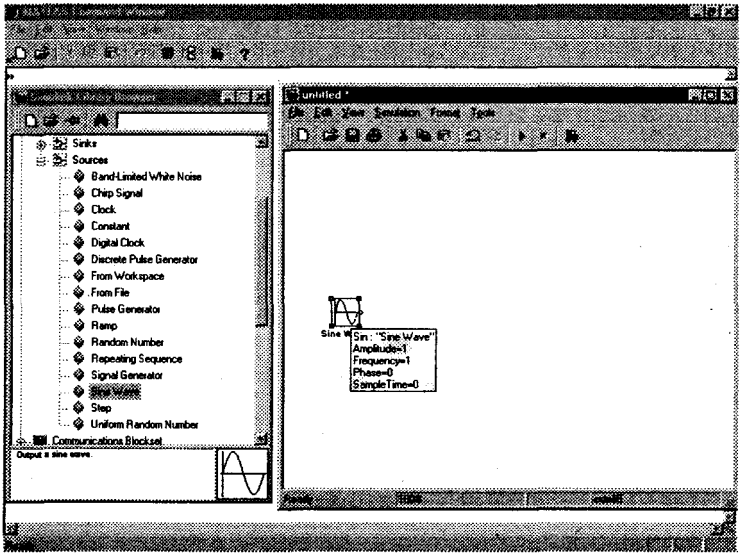


**Рис. 4.17.** Ввод компонента – источника синусоидального сигнала

Обратите внимание на то, что активизация позиции того или иного компонента на соответствующей ветви дерева компонентов ведет к появлению графического отображения компонента в окне контроля, имеющемся в правом нижнем углу окна броузера библиотеки. В нашем случае это отображение представлено синусоидой на фоне координатных осей. Можно просматривать содержимое той или иной ветви в поисках нужного компонента или (как было описано выше) вывести окно со всеми компонентами соответствующей ветви.

Если задержать на пиктограмме компонента указатель мыши, то появится всплывающая подсказка, в которой будут указаны параметры компонента по умолчанию. Полезно сразу установить нужные параметры источника синусоидального сигнала. Для этого сверните панели библиотек и дважды щелкните на только что введенном блоке. Появится окно установки параметров синусоидального сигнала (рис. 4.18). Надо установить заданную амплитуду 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.

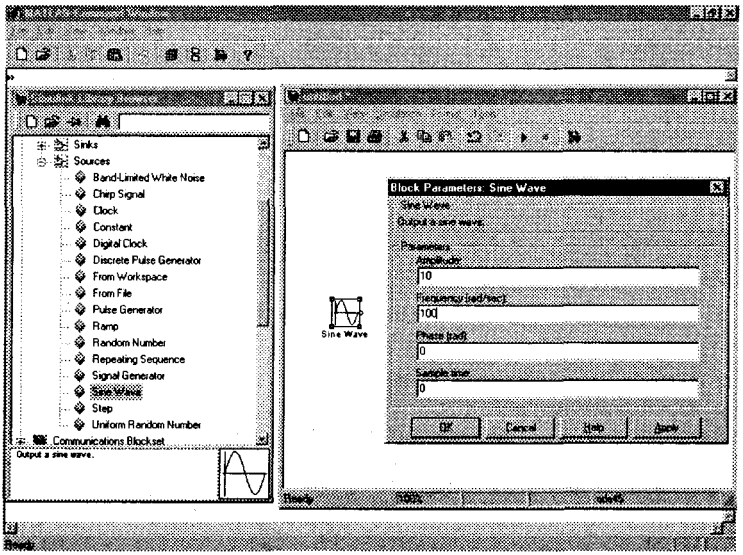


**Рис. 4.18.** Установка параметров входного сигнала

Теперь надо выбрать раздел нелинейных элементов — Nonlinear. В появившемся окне этих элементов надо выбрать блок Saturation (Ограничение) и перетащить его в нужное место окна модели, разместив справа от источника синусоидального сигнала. Этот процесс наглядно показан на рис. 4.19.

Установив указатель мыши на блок ограничения и дважды щелкнув ее *левой кнопкой, можно вывести окно установки параметров* ограничителя — оно видно на рис. 4.19 под блоком ограничителя. В этом окне надо установить верхний порог ограничения 5 В и нижний −5 В.

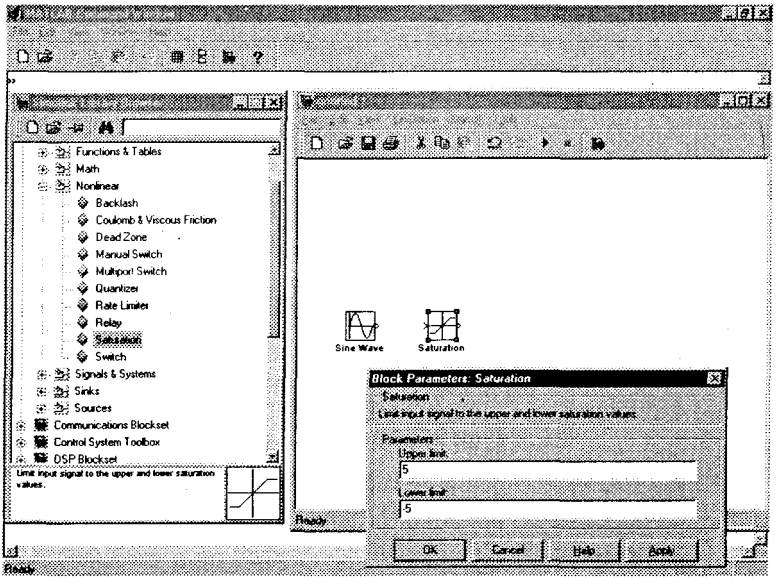


**Рис. 4.19.** Перенос блока ограничения в окно редактирования

Аналогичным образом надо ввести блок осциллографа и соединить блоки друг с другом (рис. 4.20). Для этого, установив указатель мыши на выход источника и нажав левую кнопку мыши, добейтесь, чтобы указатель превратился в крестик из тонких линий. Это означает, что редактор блок-схем готов к проведению отрезка соединительной линии. Отрезок линии проводится при нажатой левой кнопке мыши перемещением указателя до точки входа блока нелинейности. Отпустив левую кнопку мыши, вы получите соединительную линию

между блоками. Аналогичным образом соедините выход блока ограничителя со входом блока осциллографа.

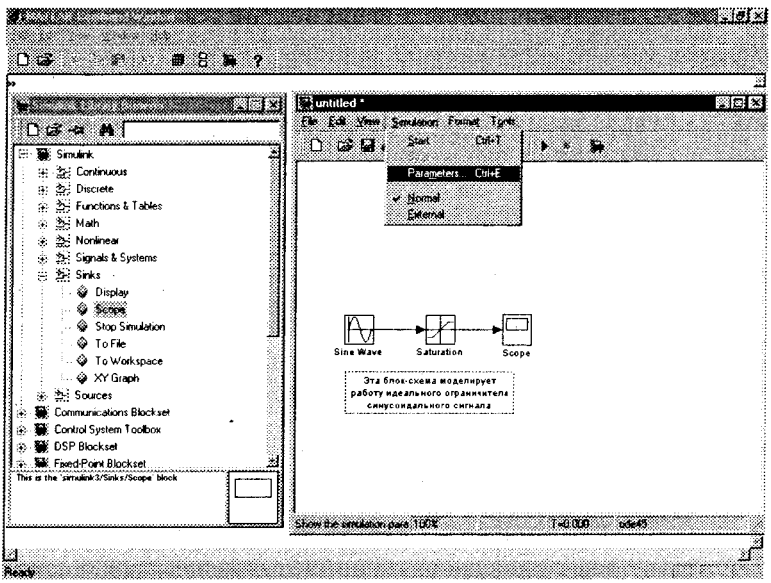


**Рис. 4.20.** Ввод блока осциллографа и создание соединений между блоками

На этом подготовка модели заданной задачи заканчивается. Впрочем, полезно выполнить еще одно действие — разместить под моделью пояснительную надпись. Для этого достаточно установить указатель мыши на свободное место окна модели и дважды щелкнуть левой кнопкой. Должно появиться прямоугольное окно с курсором ввода — мигающей вертикальной чертой. Надпись вводится как обычно, с применением средств строчного редактирования. По завершении ввода надписи надо установить указатель мыши вне поля надписи и щелкнуть левой кнопкой мыши. Блок с надписью затем можно выделить и перетащить в нужное место — под блок-схему созданной модели.

## 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 осцил-

лограф. Нужна некоторая предварительная настройка модели и осциллографа, в частности, выбор просматриваемого интервала времени, установка масштаба регистрируемой величины и т. д. 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 Parameters в меню Simulation пакета Simulink (рис. 4.20). В появившемся окне (рис. 4.21) на вкладке 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 уравнений при моделировании, а также погрешности вычислений. Как правило, однако, вполне удовлетворительны установки этих параметров по умолчанию.



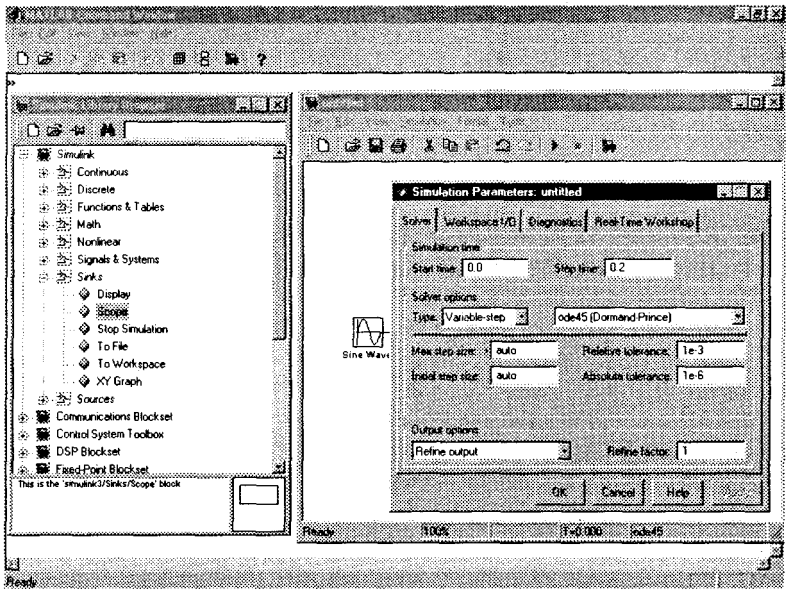


**Рис. 4.21.** Установка параметров запуска модели

Теперь можно запустить модель. Для этого надо нажать кнопку пуска (треугольник) на панели инструментов или исполнить команду Start в меню Simulation. По завершении процесса моделирования (для данной схемы он занимает доли секунды) активизация объекта-осциллографа выводит окно, в котором виден результат моделирова-

ния. Это окно показано на рис. 4.22 и очень напоминает экран реального осциллографа.

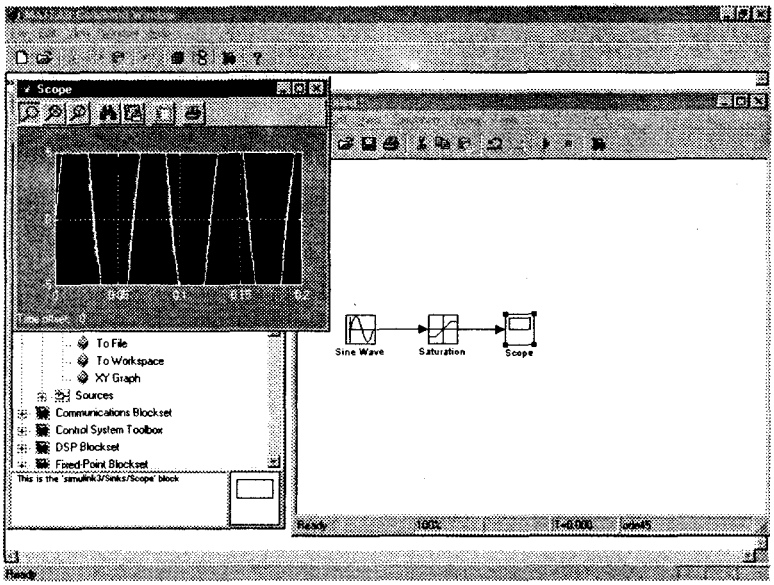


**Рис. 4.22.** Результат моделирования

Если на экране осциллографа наблюдается что-то непонятное, значит, он не настроен. Можно задать автоматическую настройку, нажав кнопку панели инструментов блока осциллографа с изображением бинокля. При этом автоматически выбираются масштабы просмотра по вертикали и горизонтали. Вид окна осциллографа в этом случае соответствует показанному на рис. 4.22.

Для «ручной» настройки осциллографа можно активизировать кнопку Properties (Свойства) на панели инструментов его окна. Окно свойств осциллографа также показано на рис. 4.22 (справа от окна осциллографа). Панель инструментов окна осциллографа имеет еще ряд кнопок вполне очевидного назначения — для изменения масштаба изображения, его печати и т. д. Пользователь может легко разобраться с их действием.

Как и следовало ожидать, в результате моделирования получена синусоида с обрезанными на уровне 5 В вершинами. При этом результат получен мгновенно — см. данные в строке состояния окна

модели (время моделирования — 0). 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». Моделирование сложных моделей на обычном персональном компьютере может занимать многие часы и даже дни, так что для этого лучше использовать MATLAB, установленный на более мощном компьютере.

Можно сохранить созданную модель для последующего применения, показа или модернизации. Для этого используется команда Save или Save As в меню File окна редактора моделей. Процесс записи ничем не отличается от обычного. Модель записывается в виде файла с расширением .mdl.

## Модернизация и расширение модели

Теперь, после создания простой модели, можно попытаться модернизировать ее и дополнить, изменить параметры модели и т. д. На рис. 4.23, к примеру, показана модель, в которой к источнику синусоидального сигнала подключены уже три нелинейных устройства — ограничитель пиков, нелинейность типа «мертвая зона» и квантующее устройство. К каждому из них подключен свой осциллограф.

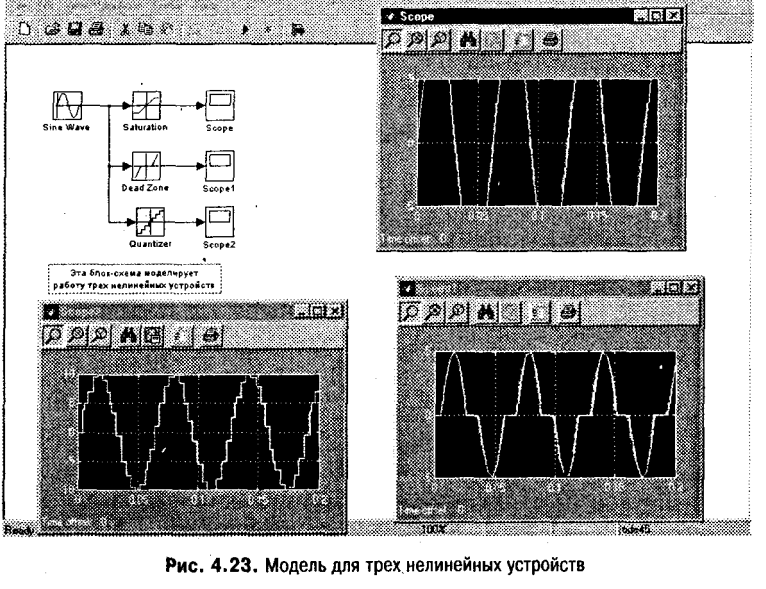


**Рис. 4.23.** Модель для трех нелинейных устройств

Как видно из рис. 4.23, теперь можно наблюдать сигналы с выхода всех трех нелинейных устройств. При этом выходные сигналы с каждого устройства наблюдаются с помощью своего осциллографа. Позже мы покажем, что возможно наблюдение и нескольких сигналов одним осциллографом, если перед ним установить микшер сигналов.

## Некоторые приемы редактирования модели

Иногда бывает нужно 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 панели инструментов с изображением ножниц (Cut), которая помещает выделенный объект в буфер обмена Windows 95/98 и при этом удаляет его со своего места.



Выделение объектов удобнее всего осуществляется мышью. Достаточно установить ее указатель на нужном объекте и один раз щелкнуть левой кнопкой мыши. Объект будет выделен. Двойной щелчок обычно вызывает окно коррекции параметров объекта (блока). Мышью также можно выделить несколько объектов. Для этого надо установить указатель вблизи от них и, нажав и удерживая левую кнопку мыши, начать перемещать мышь. 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 меню Edit.

Для стирания выделенного объекта можно использовать и команду Delete в контекстно-зависимом меню правой кнопки мыши (рис. 4.24). Контекстно-зависимое меню очень удобно тем, что для любого объекта оно выводит перечень команд и операций, которые доступны для заданного контекста (состояния).

Как видно из рис. 4.23, 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 приведенных выше в качестве примеров, и вы убедитесь в том, что блок-схема из десятка элементов может быть составлена за считанные минуты. При этом приемы редактирования очень напоминают работу с популярными графическими редакторами, кото-

рую легко осваивают даже дети. Что, однако, стоит отметить, так это возможность задания наклонных линий соединений при нажатой клавише Shift.

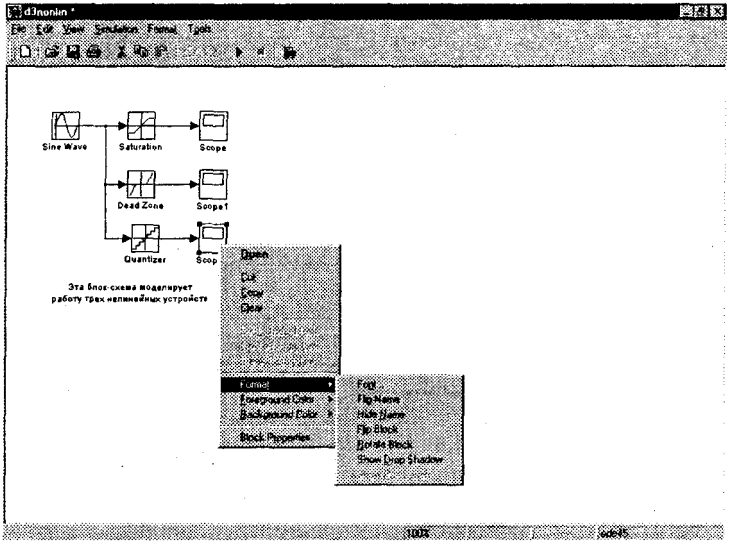


**Рис. 4.24.** Контекстно-зависимое меню

Simulink имеет расширенные возможности редактирования блок-схем. Так, в окне редактирования можно не только перемещать блоки с помощью мыши, но и менять их размеры. Для этого блок выделяется, после чего указатель мыши надо установить на кружки по углам блока. Как только указатель превратится в двунаправленную диагональную стрелку, можно будет при нажатой левой кнопке растягивать блоки по диагонали, увеличивая или уменьшая их размеры. Кроме того, блоки можно помещать в буфер обмена (Clipboard) операционной системы Windows 95/98 и использовать буфер для переноса блоков из одного места в другое.

В меню Format (и в контекстно-зависимом меню) можно найти ряд команд форматирования блоков: замены шрифта пояснительных надписей и их стиля (Font), смены расположения надписей (Flip name), устранения и восстановления надписей (Show Name/Hide Name), поворота блоков вокруг вертикальной оси (Flip Block), поворота на 90° (Rotate Block), включения и отключения тени (Show Drop Shadow/Hide

Drop Shadow). Там же есть очевидные опции по изменению цвета линий блока, закраске блоков и настройке цвета общего фона.

Большую помощь в редактировании оказывает команда Undo — отмена последней операции. Она поддерживает свыше ста операций, включая операции добавления и стирания линий. Эту команду можно реализовать с помощью кнопки в панели инструментов или из меню Edit.

# Примеры работы с Simulink

## Построение фигур Лиссажу



Вначале рассмотрим простой пример применения виртуального графопостроителя для построения фигур Лиссажу. Такие фигуры получаются, если на входы X и Y подать синусоидальные сигналы, отношение которых выражается рациональной дробью, — при этом фигура получается неподвижной. Для построения таких фигур средствами Simulink достаточно создать модель, содержащую два генератора синусоидальных колебаний и XY-графопостроитель. Такая модель показана на рис. 4.25.

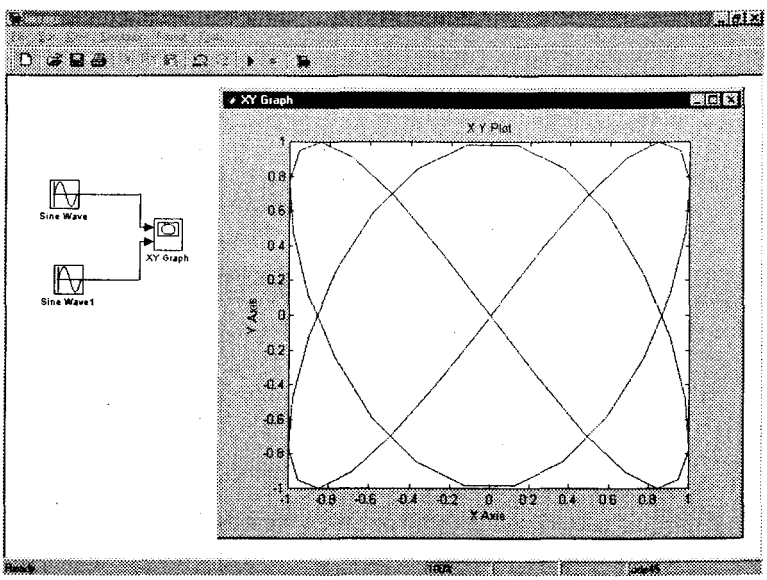


**Рис. 4.25.** Модель для построения фигур Лиссажу

На рис. 4.25 показан также результат моделирования. Частоты генераторов были установлены равными 2 и 3 рад/с, то есть с отношением 2/3. Это отношение и определяет вид наблюдаемой замкнутой кривой.

## 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 квантовых генераторов.



Пример моделирования типовой системы второго порядка есть в наборе демонстрационных примеров пакета Simulink. Модель такой системы представлена на рис. 4.26. Эта система описывается хорошо известным нелинейным дифференциальным уравнением второго порядка Ван-дер-Поля.

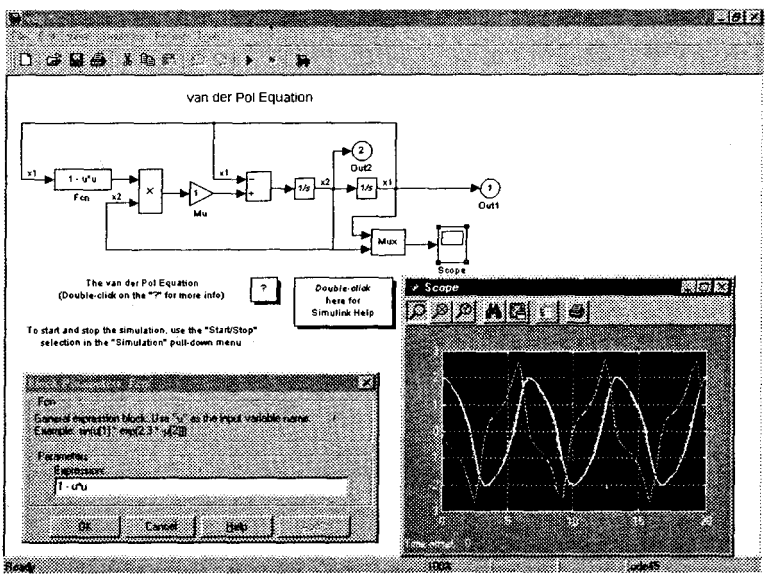


**Рис. 4.26.** Модель колебательной системы второго порядка

Как нетрудно заметить, данная модель представляет собой типичный усилитель с нелинейным элементом Fcn, позволяющим задать

тип нелинейности, с положительной обратной связью и имеющий 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 Fcn.

На рис. 4.26 показан и результат моделирования. Нетрудно заметить, что в данном случае осциллограф отображает две кривые, соответствующие выходным портам Out 1 и Out 2. Для этого перед осциллографом размещен блок микшера Mux с двумя входами. Результат моделирования отображается в виде временных зависимостей выходных сигналов. Этот результат получен при $F(u) = 1 - u^2$. Окно задания нелинейности представлено на рис. 4.26 в левом нижнем углу. Оно появляется при активизации блока нелинейности Fcn.



Допустим, нас интересует поведение системы для иной нелинейности, скажем $F(u) = 1 - \exp(u)$. Для замены нелинейности достаточно сделать двойной щелчок на блоке Fcn. В появившемся окне параметров надо вместо функции по умолчанию ввести новую функцию, отражающую нелинейность модели. Это и показано на рис. 4.27.

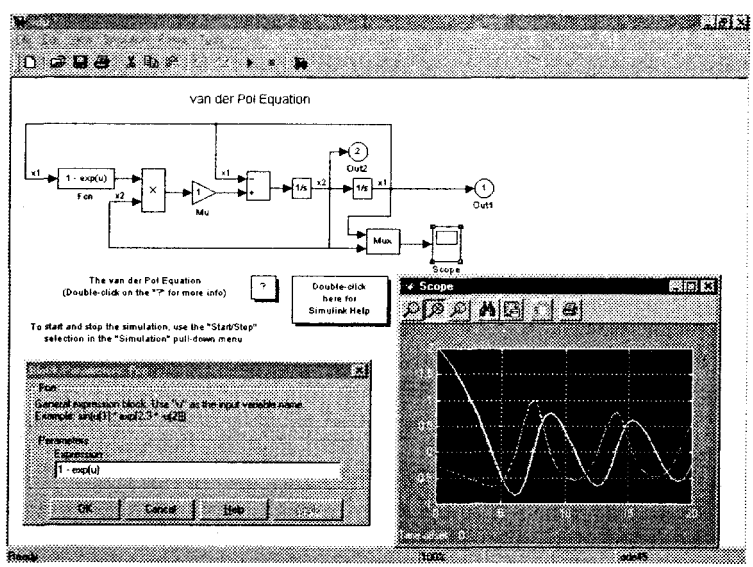


**Рис. 4.27.** Изменение функции нелинейности в модели системы второго порядка

После уточнения нелинейности и закрытия окна параметров можно произвести пуск измененной модели. Результаты также представлены на рис. 4.27. Сравнение временных диаграмм (осциллограмм) выходных сигналов на этом рисунке с приведенными на рис. 4.26 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.

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. 4.28 показана модель системы второго порядка, дополненная графопостроителем. В данном случае $F(u) = 1 - 1{,}1\exp(-u)$.

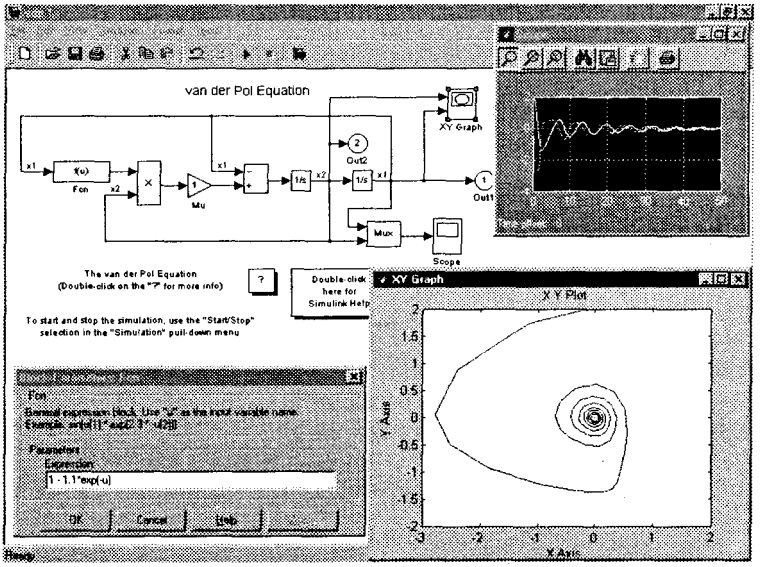


**Рис. 4.28.** Модель системы второго порядка с новыми данными и фазовый портрет ее колебаний

В данном случае параметры нелинейности подобраны таким образом, что колебательный процесс возникает только в начале включения системы. Затем за несколько периодов колебания затухают. Этот 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 — с тем, что аналитическое описание многих моделей можно менять, причем оно выполняется по правилам, принятым в системе MATLAB. Благодаря этому математическая сущность модели оказывается вполне понятной, а результаты моделирования наглядно и адекватно описывают работу сложных моделей при введении в их описание самых различных математических закономерностей.

## 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 уравнениями Ван-дер-Поля. Однако Simulink имеет специальный редактор дифференциальных уравнений, с помощью которого можно задать систему дифференциальных уравнений в форме 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. 4.29 сверху.

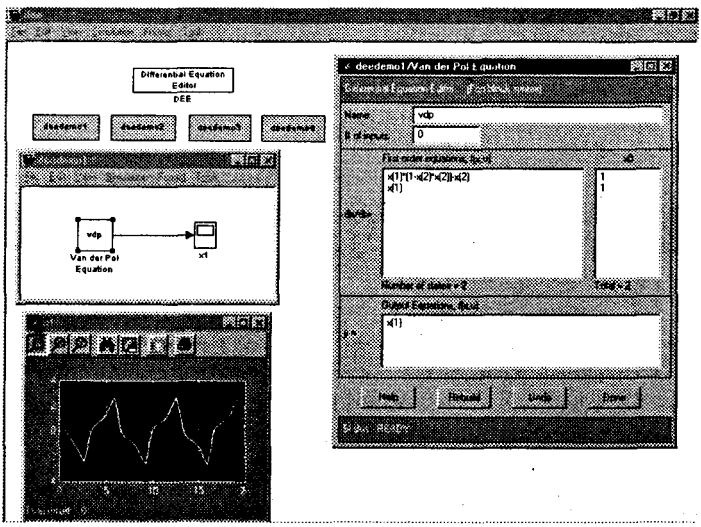


**Рис. 4.29.** Работа с решателем и редактором дифференциальных уравнений

Ограничимся примером с именем ddedemo1. Он выводит окно всего с двумя блоками: блоком vdp и осциллографом 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 на рис. 4.29 в правой части. В этом окне можно модифицировать решаемые уравнения или ввести новые. Для просмотра решения используется осциллограф — его окно также представлено на рис. 4.29.

В системе MATLAB одну и ту же задачу можно решать целым рядом способов. В этом случае получение одинаковых результатов (в том числе при решении средствами Simulink) дает дополнительную гарантию правильности решения и корректности создаваемых моделей.

## Моделирование работы автопилота самолета F14

Рассмотрим еще один пример из набора, входящего в директорию MATLAB/ TOOLBOX/SIMULINK/F14. Этот пример показывает работу одного из вариантов автопилота самолета F14 — устройства, управляющего самолетом таким образом, чтобы он летел в строго заданном направлении. Модель автопилота представлена на рис. 4.30.

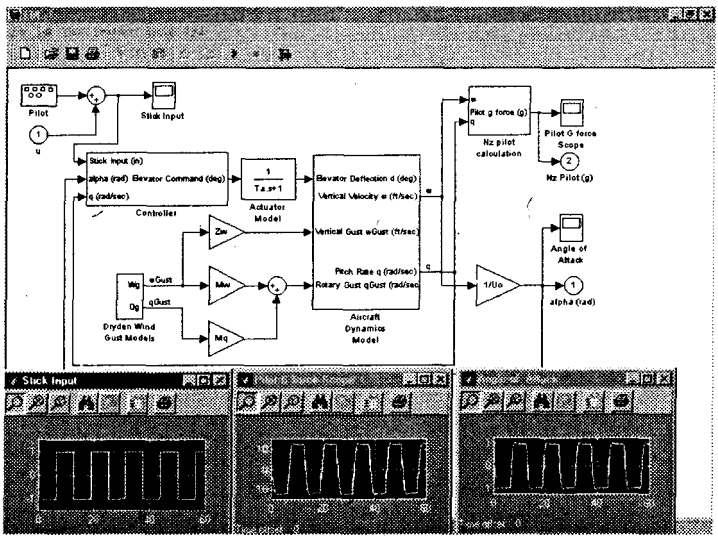


**Рис. 4.30.** Модель аналогового автопилота

На сей раз моделируется весьма сложное и вполне современное устройство, или, скорее, система управления. На приведенном рисунке показаны осциллограммы, характеризующие работу автопилота. Видны небольшие колебания переходной характеристики автопилота, характерные для систем регулирования.

## Применение подсистем

Присмотревшись внимательнее к блок-схеме автопилота, вы заметите, что входящие в нее блоки Controller, Aircraft Dynamic Model и некоторые другие не являются библиотечными компонентами. Это *подсистемы*, то есть ранее отлаженные модели, оформленные в виде блока. Практики часто называют подсистемы «черными ящиками», поскольку их содержимое 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 указатель мыши и дважды щелкнуть левой кнопкой мыши. Если появляется окно установки параметров, значит, блок — библиотечный компонент. Если же вместо этого появится окно с блок-схемой, значит, блок является подсистемой. На рис. 4.31 показана модель одного из указанных блоков — блока Controller.

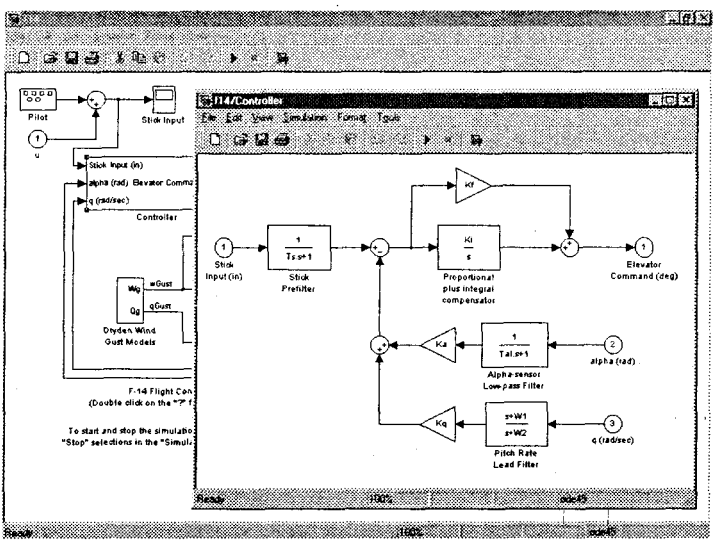


**Рис. 4.31.** Типовой пример подсистемы

Нетрудно заметить, что подсистема отличается от обычной модели только тем, что все ее 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 меню Edit. Можно также использовать библиотечный блок Subsystem, который открывает пустое окно для подготовки подсистемы.

*Использование концепции подсистем 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 различные электронные устройства.

4

## Использование S-функции

Как уже отмечалось, Simulink позволяет определять и использовать системные S-функции. Показанный на рис. 4.32 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
```

*В результате будет выведено 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.
```

**Итак, в данном случае эта функция задает решение системы дифференциальных уравнений. Команда type simom позволяет просмот-**

реть весь файл, что полезно тем, кто решит опробовать свои силы в подготовке собственных S-функций.

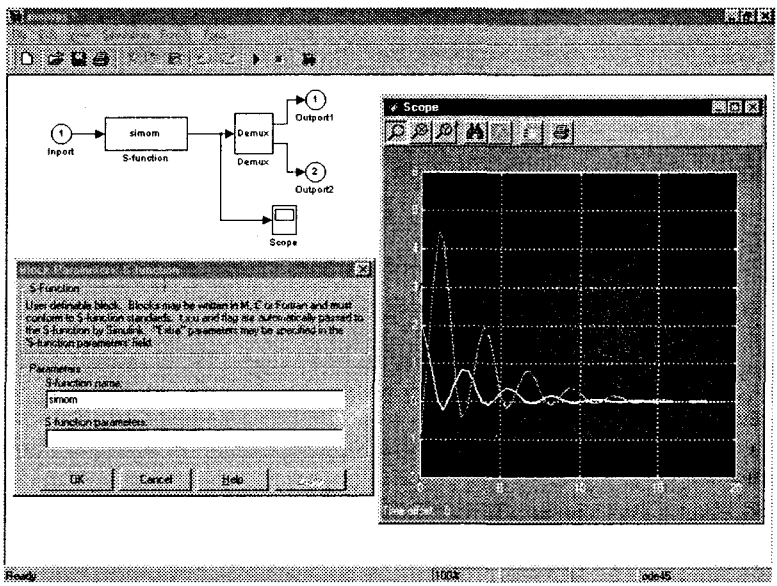


**Рис. 4.32.** Пример применения S-функции

Применение аппарата S-функций способствует математической наглядности решения задач в среде пакета Simulink. Однако практическое освоение этого аппарата требует более детального знакомства с S-функциями, которое выходит за рамки данной книги.

## Применение специальных преобразователей 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 *дискретизацию*. Другой преобразователь определяет значение первой производной сигнала в момент выборки и до следующей выборки вырабатывает линейнонарастающий сигнал. На рис. 4.33 показано действие таких блоков.

Кроме того, этот пример дает наглядное представление о технике просмотра осциллографом сразу трех сигналов — исходного сину-

соидального и двух сигналов от преобразующих блоков. Для объединения сигналов используется блок Mux.

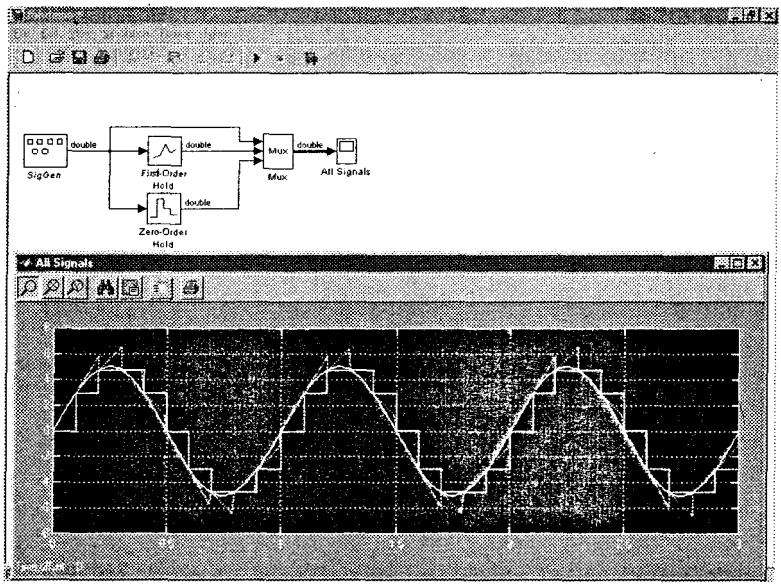


**Рис. 4.33.** Пример просмотра сигналов от трех блоков

## Еще один пример сложной системы

На рис. 4.34 показана модель еще одной сложной системы — Vehicle Suspension. Этот демонстрационный пример, иллюстрирующий работу автомобильной подвески, приведен не столько ради обсуждения конкретной системы, сколько как пример вставки в модель блоков нестандартного вида.

При моделировании используется оригинальное устройство регистрации, позволяющее строить графики, весьма напоминающие графики ленточных самописцев, широко применяемых в промышленности. Вид регистрации для данной модели представлен на рис. 4.35.

В библиотеке компонентов Simulink можно найти и множество других регистрирующих устройств, а также средства для создания анимационных видеоклипов, существенно повышающих наглядность визуализации результатов моделирования различных физических, технических и иных систем и устройств.

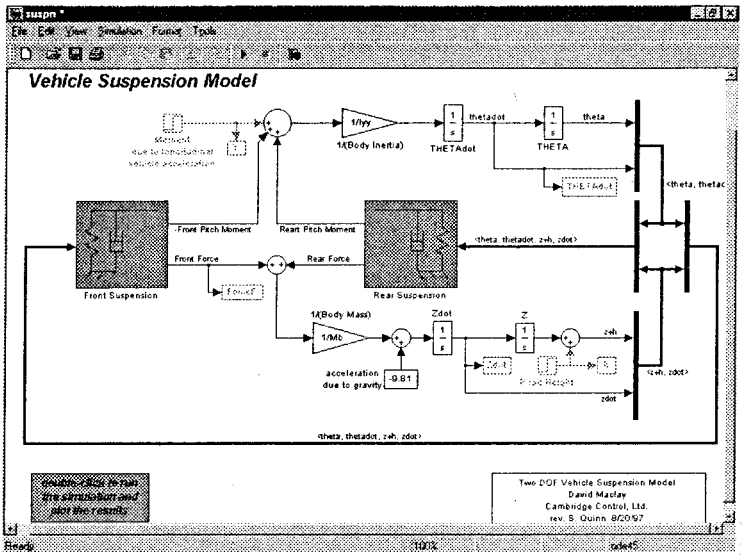


**Рис. 4.34.** Сложная модель Vehicle Suspension

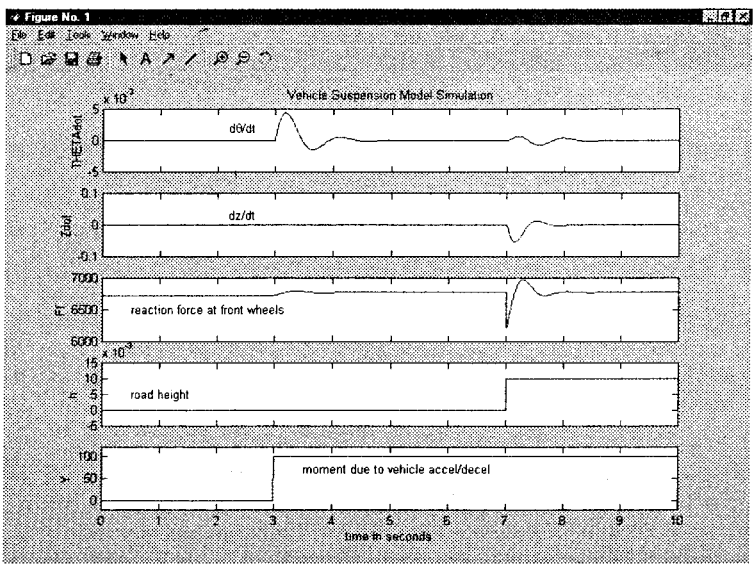


**Рис. 4.35.** Пример регистрации множества графиков

4

## Моделирование работы унитаза

Для тех, кому приведенные выше примеры кажутся слишком далекими от обыденной жизни, приведем еще один пример — моделирование работы столь известного устройства, как унитаз. Модель сливного бачка унитаза представлена на рис. 4.36.

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, буквально воспроизводящим звуки слива воды в унитазе.

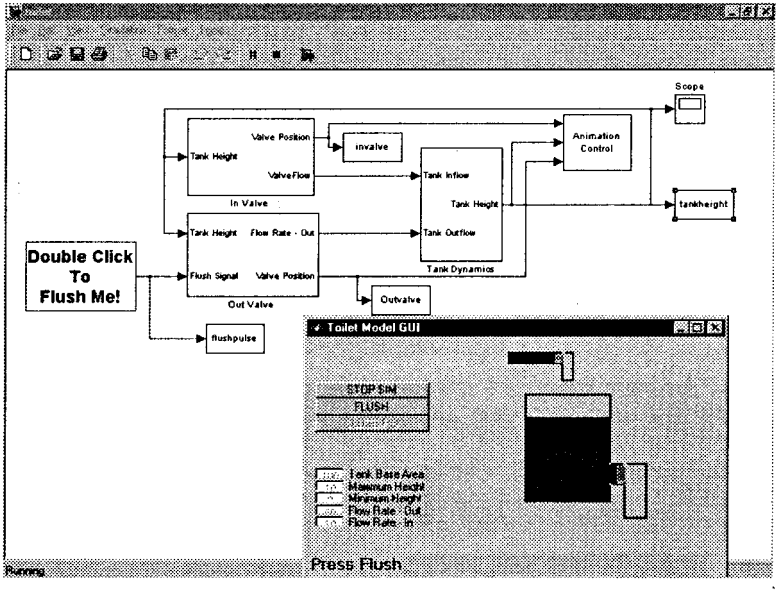


**Рис. 4.36.** Модель сливной системы унитаза

# Глава 5
# Пакет расширения по нейронным сетям

## Назначение пакета Neural Networks Toolbox

Пакет Neural Networks Toolbox (Нейронные сети) содержит средства для проектирования, моделирования, обучения и использования множества известных парадигм аппарата искусственных нейронных сетей (ИНС): от базовых моделей персептрона до самых современных ассоциативных и самоорганизующихся сетей. Пакет может быть использован для решения множества разнообразных задач, таких как обработка сигналов, нелинейное управление, финансовое моделирование и т. п.

Для каждого типа архитектуры и обучающего алгоритма ИНС имеются функции инициализации, обучения, адаптации, создания, моделирования, демонстрации, а также примеры применения. Искусственные многослойные нейронные сети конструируются по принципам построения их биологических аналогов. Они уже сейчас способны решать широкий круг задач распознавания образов, идентификации, управления сложными нелинейными объектами, роботами и т. п. Отметим, что в настоящее время дальнейшее повышение производительности компьютеров связывают с ИНС, в частности, с так называемыми нейрокомпьютерами (НК), основу которых составляет искусственная нейронная сеть.

Термин «нейронные сети» сформировался в 40-х годах XX в. в среде исследователей, изучавших принципы организации и функционирования биологических нейронных сетей. Основные результаты, полученные в этой области, связаны с именами американских исследователей У. Маккалоха, Д. Хебба, Ф. Розенблатта, М. Минского, Дж. Хопфилда и др.

Представим некоторые проблемы, решаемые в контексте ИНС и представляющие интерес для пользователей.

- **Классификация образов.** Задача состоит в указании принадлежности входного образа (например, речевого сигнала или рукописного символа), представленного вектором признаков, к одному или

нескольким предварительно определенным классам. К известным приложениям относятся распознавание букв, распознавание речи, классификация сигнала электрокардиограммы, классификация клеток крови.

- **Кластеризация/категоризация.** При решении задачи кластеризации, которая известна также как классификация образов «без учителя», отсутствует обучающая выборка с метками классов. Алгоритм кластеризации основан на подобии образов и помещает близкие образы в один кластер. Известны случаи применения кластеризации для извлечения знаний, сжатия данных и исследования свойств данных.
- **Аппроксимация функций.** Предположим, что имеется обучающая выборка $((x^1, y^1), (x^2, y^2)..., (x^N, y^N))$ (пары данных «вход–выход»), которая генерируется неизвестной функцией $F(x)$, искаженной шумом. Задача аппроксимации состоит в нахождении оценки неизвестной функции $F(x)$. Аппроксимация функций необходима при решении многочисленных инженерных и научных задач моделирования.
- **Предсказание/прогноз.** Пусть заданы $n$ дискретных отсчетов $\{y(t_1), y(t_2), ..., y(t_k)\}$ в последовательные моменты времени $t_1$, $t_2$, ..., $t_k$. Задача состоит в предсказании значения $y(t_{k+1})$ в некоторый будущий момент времени $t_{k+1}$. Предсказание/прогноз имеют значительное влияние на принятие решений в бизнесе, науке и технике. Предсказание цен на фондовой бирже и прогноз погоды являются типичными приложениями техники предсказания/прогноза.
- **Оптимизация.** Многочисленные проблемы в математике, статистике, технике, науке, медицине и экономике могут рассматриваться как проблемы оптимизации. Задачей алгоритма оптимизации является нахождение такого решения, которое удовлетворяет системе ограничений и максимизирует или минимизирует целевую функцию. Известная задача коммивояжера является классическим примером задачи оптимизации.
- **Память, адресуемая по содержимому.** В модели вычислений фон Неймана обращение к памяти доступно только посредством адреса, который не зависит от содержимого памяти. Более того, если допущена ошибка в вычислении адреса, то может быть найдена совершенно иная информация. Ассоциативная память, или память, адресуемая по содержимому, доступна по указанию заданного содержимого. Содержимое памяти может быть вызвано даже по частичному входу или искаженному содержимому. Ассоциативная

память чрезвычайно желательна при создании мультимедийных информационных баз данных.

- **Управление.** Рассмотрим динамическую систему, заданную совокупностью $\{u(t), y(t)\}$, где $u(t)$ является входным управляющим воздействием, а $y(t)$ — выходом системы в момент времени $t$. В системах управления с эталонной моделью целью управления является расчет такого входного воздействия $u(t)$, при котором система следует по желаемой траектории, диктуемой эталонной моделью. Примером является оптимальное управление двигателем.

Эти и подобные задачи успешно решаются средствами пакета Neural Networks, который входит в состав расширенных версий систем MATLAB. Для дополнительного знакомства с нейронными сетями можно рекомендовать следующие книги:

1. Уоссерман Ф. Нейрокомпьютерная техника: Теория и практика. — М.: Мир, 1992. — 237 с.
2. Змитрович А. И. Интеллектуальные информационные системы/ НТООО «ТетраСистемс», Минск, 1997. — 368 с.



## Биологический нейрон

Под нейронными сетями подразумеваются вычислительные структуры, которые моделируют простые биологические процессы, обычно ассоциируемые с процессами человеческого мозга. Адаптируемые и обучаемые, они представляют собой распараллеленные системы, способные к обучению путем анализа положительных и отрицательных воздействий. Элементарным преобразователем в данных сетях является *искусственный нейрон*, или просто *нейрон*, названный так по аналогии с биологическим прототипом.

К настоящему времени предложено и изучено большое количество моделей нейроподобных элементов и нейронных сетей, некоторые из которых рассмотрены далее.

Нервная система и мозг человека состоят из нейронов, соединенных между собой нервными волокнами. Нервные волокна способны передавать электрические импульсы между нейронами. Все процессы передачи раздражений от нашей кожи, ушей и глаз к мозгу, процессы мышления и управления действиями — все это реализовано в живом организме как передача электрических импульсов между нейронами.

*Нейрон* (нервная клетка) является особой биологической клеткой, которая обрабатывает информацию (рис. 5.1). Он состоит из тела и

отростков нервных волокон двух типов — *дендритов*, по которым принимаются импульсы, и единственного *аксона*, по которому нейрон может передавать импульс. Тело нейрона включает ядро, которое содержит информацию о наследственных свойствах, и плазму, обладающую молекулярными средствами для производства необходимых нейрону материалов. Нейрон получает сигналы (импульсы) от аксонов других нейронов через дендриты (приемники) и передает сигналы, сгенерированные телом клетки, вдоль своего аксона (передатчика), который в конце разветвляется на волокна. На окончаниях этих волокон находятся специальные образования — *синапсы*, которые влияют на силу импульса.

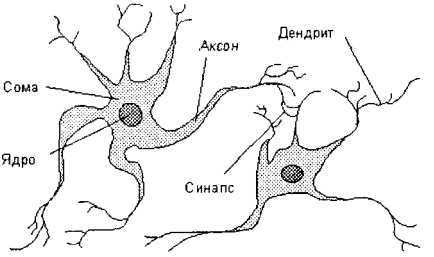


**Рис. 5.1.** Взаимосвязь биологических нейронов

Синапс является элементарной структурой и функциональным узлом между двумя нейронами (волокно аксона одного нейрона и дендрит другого). Когда импульс достигает синаптического окончания, высвобождаются определенные химические вещества, называемые нейротрансмиттерами. Нейротрансмиттеры диффундируют через синаптическую щель, возбуждая или затормаживая, в зависимости от типа синапса, способность нейрона-приемника генерировать электрические импульсы. Результативность синапса может настраиваться проходящими через него сигналами, так что синапсы могут обучаться в зависимости от активности процессов, в которых они участвуют. Эта зависимость от предыстории действует как память, которая, возможно, ответственна за память человека. Важно отметить, что веса синапсов могут изменяться со временем, что изменяет и поведение соответствующего нейрона.

## Структура и свойства искусственного нейрона

Нейрон — это составная часть нейронной сети (НС). На рис. 5.2 показана его структура.

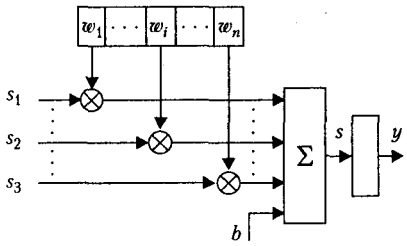


**Рис. 5.2.** Структура искусственного нейрона

В состав нейрона входят умножители (синапсы), сумматор и нелинейный преобразователь. Синапсы осуществляют связь между нейронами и умножают входной сигнал на число, характеризующее силу связи, — вес синапса. Сумматор выполняет сложение сигналов, поступающих по синаптическим связям от других нейронов, и внешних входных сигналов. Нелинейный преобразователь реализует нелинейную функцию одного аргумента — выхода сумматора. Эта функция называется *функцией активации* или *передаточной функцией* нейрона. Нейрон в целом реализует скалярную функцию векторного аргумента. Математическая модель нейрона описывается соотношениями

$$s = \sum_{i=1}^{n} w_i x_i + b;$$

$$y = f(s),$$

где $w_i$ — вес синапса ($i = 1...n$); $b$ — значение смещения; $s$ — результат суммирования; $x_i$ — компонент входного вектора (входной сигнал) ($i = 1...n$); $y$ — выходной сигнал нейрона; $n$ — число входов нейрона; $f$ — нелинейное преобразование (функция активации или передаточная функция).

В общем случае входной сигнал, весовые коэффициенты и значения смещения могут принимать вещественные значения. Выход ($y$) определяется видом функции активации и может быть как действительным, так и целым. Во многих практических задачах входы, веса и смещения могут принимать лишь некоторые фиксированные значения.

Синаптические связи с положительными весами называют *возбуждающими*, с отрицательными весами — *тормозящими*.

Таким образом, нейрон полностью описывается своими весами $w_i$ и передаточной функцией $f(s)$. Получив набор чисел (вектор) $x_i$ в качестве входов, нейрон выдает некоторое число $y$ на выходе.

Описанный вычислительный элемент можно считать упрощенной математической моделью биологических нейронов — клеток, из которых состоит нервная система человека и животных.

Чтобы подчеркнуть различие нейронов биологических и математических, вторые иногда называют *нейроноподобными элементами* или *формальными нейронами*.

На входной сигнал $s$ нелинейный преобразователь отвечает выходным сигналом $f(s)$, который представляет собой выход нейрона $y$. Примеры активационных функций представлены в табл. 5.1 и на рис. 5.3.

**Таблица 5.1.** Перечень функций активации нейронов

| Название | Формула | Область значений |
|---|---|---|
| Пороговая | $f(s)=\begin{cases}0, & s<T,\\ 1, & s\ge T\end{cases}$ | (0, 1) |
| Знаковая (сигнатурная) | $f(s)=\begin{cases}1, & s>0,\\ -1, & s\le 0\end{cases}$ | (−1, 1) |
| Сигмоидальная (логистическая) | $f(x)=\dfrac{1}{1+e^{-s}}$ | (0, 1) |
| Полулинейная | $f(s)=\begin{cases}s, & s>0,\\ 0, & s\le 0\end{cases}$ | (0, ∞) |
| Линейная | $f(s)=s$ | (−∞, ∞) |
| Радиальная базисная (гауссова) | $f(s)=\exp(-s^2)$ | (0, 1) |
| Полулинейная с насыщением | $f(s)=\begin{cases}0, & s\le 0,\\ s, & 0<s<1,\\ 1, & s\ge 1\end{cases}$ | (0, 1) |
| Линейная с насыщением | $f(s)=\begin{cases}-1, & s\le -1,\\ s, & -1<s<1,\\ 1, & s\ge 1\end{cases}$ | (−1, 1) |
| Гиперболический тангенс (сигмоидальная) | $f(s)=\dfrac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$ | (−1, 1) |
| Треугольная | $f(s)=\begin{cases}1-\lvert s\rvert, & \lvert s\rvert\le 1,\\ 0, & \lvert s\rvert>1\end{cases}$ | (0, 1) |

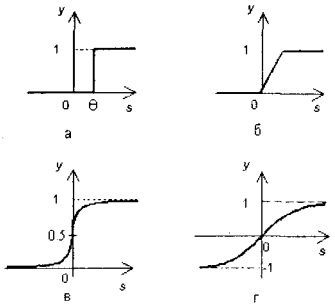


**Рис. 5.3.** Примеры активационных функций: а – пороговая; б – полулинейная с насыщением; в – сигмоид (логистическая функция); г – сигмоид (гиперболический тангенс)

Одной из наиболее распространенных является нелинейная функция с насыщением, так называемая *логистическая функция*, или *сигмоид* (то есть функция S-образного вида):

$$f(s) = \frac{1}{1 + e^{-s}}.$$

Из выражения для сигмоида очевидно, что выходное значение нейрона лежит в диапазоне [0, 1]. Одно из ценных свойств сигмоидальной функции — простое выражение для ее производной:

$$f'(s) = f(s)\,[1 - f(s)],$$

которое используется в некоторых алгоритмах обучения (см. далее). Кроме того, данная функция обладает свойством «усиливать» слабые сигналы лучше, чем большие, и предотвращает насыщение от больших сигналов, так как они соответствуют областям аргументов, где сигмоид имеет пологий наклон.

## Классификация нейронных сетей и их свойства

Искусственная нейронная сеть — это набор нейронов, соединенных между собой. Как правило, передаточные (активационные) функции всех нейронов в сети фиксированы, а веса являются параметрами сети и могут изменяться. Некоторые входы нейронов помечены как внешние входы сети, а некоторые выходы — как внешние выходы сети. Подавая любые числа на входы сети, мы получаем какой-то набор чисел на выходах сети. Таким образом, работа нейросети состоит в преобразовании входного вектора **X** в выходной вектор **Y**, причем это преобразование задается весами сети.

Практически любую задачу можно свести к задаче, решаемой нейросетью. В табл. 5.2 показано, каким образом следует сформулировать в терминах нейросети задачу распознавания рукописных букв.

**Таблица 5.2.** Задача распознавания рукописных букв в терминах нейросети

| **Задача распознавания рукописных букв** | |
|---|---|
| **Дано:** растровое черно-белое изображение буквы размером 30×30 пикселей | **Надо:** определить, какая это буква (в алфавите 33 буквы) |
| **Формулировка для нейросети** | |
| **Дано:** входной вектор из 900 двоичных символов (900 = 30×30) | **Надо:** построить нейросеть с 900 входами и 33 выходами, которые помечены буквами. Если на входе сети — изображение буквы «А», то максимальное значение выходного сигнала достигается на выходе «А». Аналогично сеть работает для всех 33 букв |

Поясним, зачем требуется выбирать выход с максимальным уровнем сигнала. Дело в том, что уровень выходного сигнала, как правило, может принимать любые значения из какого-то диапазона. Однако в данной задаче нас интересует не аналоговый ответ, а всего лишь номер категории (номер буквы в алфавите). Поэтому используется следующий подход: каждой категории сопоставляется свой выход, а ответом сети считается та категория, на чьем выходе уровень сигнала максимален. В определенном смысле уровень сигнала на выходе «А» — это достоверность того, что на вход была подана рукописная буква «А». Задачи, в которых нужно отнести входные данные к одной из известных категорий, называются задачами классификации. Изложенный подход — стандартный способ классификации с помощью нейронных сетей.

Теперь, когда стало ясно, что именно мы хотим построить, мы можем переходить к вопросу «как строить такую сеть». Этот вопрос решается в два этапа:

1. Выбор типа (архитектуры) сети.
2. Подбор весов (обучение) сети.

На первом этапе следует выбрать следующее:

- какие нейроны мы хотим использовать (число входов, передаточные функции);

- каким образом следует соединить их между собой;
- что взять в качестве входов и выходов сети.

Эта задача на первый взгляд кажется необозримой, но, к счастью, нам необязательно придумывать нейросеть «с нуля» — существует несколько десятков различных нейросетевых архитектур, причем эффективность многих из них доказана математически. Наиболее популярные и изученные архитектуры — это многослойный персептрон, нейросеть с общей регрессией, сети Кохонена и др., которые будут рассмотрены далее.

На втором этапе нам следует «обучить» выбранную сеть, то есть подобрать такие значения ее весов, чтобы сеть работала нужным образом. Необученная сеть подобна ребенку — ее можно научить чему угодно. В используемых на практике нейросетях количество весов может составлять несколько десятков тысяч, поэтому обучение — действительно сложный процесс. Для многих архитектур разработаны специальные алгоритмы обучения, которые позволяют настроить веса сети определенным образом.



В зависимости от функций, выполняемых нейронами в сети, можно выделить три их типа:

- **входные нейроны** — это нейроны, на которые подается входной вектор, кодирующий входное воздействие или образ внешней среды; в них обычно не осуществляются вычислительные процедуры, информация передается со входа на выход нейрона путем изменения его активации;
- **выходные нейроны** — это нейроны, выходные значения которых представляют выход сети; преобразования в таких нейронах осуществляются по приведенным выше выражениям;
- **промежуточные нейроны** — это нейроны, составляющие основу искусственных нейронных сетей, преобразования в них выполняются по этим же выражениям.

В большинстве нейронных моделей тип нейрона связан с его расположением в сети. Если нейрон имеет только выходные связи, то это входной нейрон, если наоборот — выходной нейрон. Однако может встретиться случай, когда выход топологически внутреннего нейрона рассматривается как часть выхода сети. В процессе функционирования (эволюции состояния) сети осуществляется преобразование входного вектора в выходной, то есть некоторая переработка информации. Конкретный вид выполняемого сетью преобразования информации обусловливается не только характеристиками нейропо-

добных элементов (функциями активации и т. п.), но и особенностями ее архитектуры. В частности, той или иной топологией межнейронных связей, выбором определенных подмножеств нейроподобных элементов для ввода и вывода информации, способами обучения сети, наличием или отсутствием конкуренции между нейронами, направлением и способами управления и синхронизации передачи информации между нейронами.

## Топология нейронных сетей

*Классифицируя нейронные сети по топологии, можно выделить* три основных типа таких сетей:

- полносвязные сети (рис. 5.4, *а*);
- многослойные, или слоистые сети (рис. 5.4, *б*);
- слабосвязные сети (нейронные сети с локальными связями) (рис. 5.4, *в*).

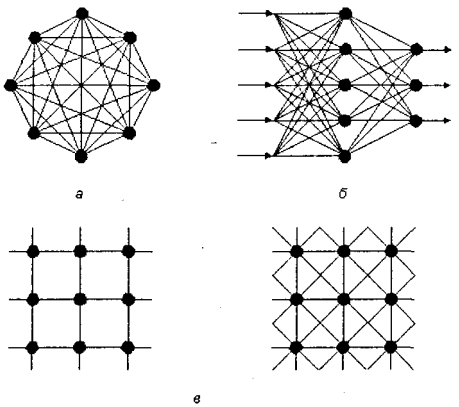


**Рис. 5.4.** Архитектуры нейронных сетей: а – полносвязная сеть, б – многослойная сеть с последовательными связями, в – слабосвязные сети

*Полносвязные сети* представляют собой ИНС, каждый нейрон которой передает свой выходной сигнал остальным нейронам, в том числе и самому себе. *Все* входные сигналы подаются *всем* нейронам. Выходными сигналами сети могут быть все или некоторые выходные сигналы нейронов после нескольких тактов функционирования сети.

В *многослойных сетях* нейроны объединяются в *слои*. Слой содержит совокупность нейронов с едиными входными сигналами. Число нейронов в каждом слое может быть любым и никак заранее не связано с количеством нейронов в других слоях. В общем случае сеть состоит из $Q$ слоев, пронумерованных слева направо. Внешние входные сигналы подаются на входы нейронов первого слоя (входной слой часто нумеруют как нулевой), а выходами сети являются выходные сигналы последнего слоя. Вход нейронной сети можно рассматривать как выход «нулевого слоя» вырожденных нейронов, которые служат лишь в качестве распределительных точек, суммирования и преобразования сигналов здесь не производится. Кроме входного и выходного слоев в многослойной нейронной сети есть один или несколько промежуточных (скрытых) слоев. Связи от выходов нейронов некоторого слоя $q$ ко входам нейронов следующего слоя $(q+1)$ называются последовательными.

В свою очередь среди слоистых сетей выделяют следующие типы:



1. **Монотонные.** Это специальный частный случай слоистых сетей с дополнительными условиями на связи и элементы. Каждый слой, кроме последнего (выходного), разбит на два блока: возбуждающий (В) и тормозящий (Т). Связи между блоками тоже разделяются на тормозящие и возбуждающие. Если от блока $A$ к блоку $C$ ведут только возбуждающие связи, это означает, что любой выходной сигнал блока $C$ является монотонной неубывающей функцией любого выходного сигнала блока $A$. Если же эти связи только тормозящие, то любой выходной сигнал блока $C$ является невозрастающей функцией любого выходного сигнала блока $A$. Для элементов монотонных сетей необходима монотонная зависимость выходного сигнала элемента от параметров входных сигналов.
2. **Сети без обратных связей.** В таких сетях нейроны входного слоя получают входные сигналы, преобразуют их и передают нейронам 1-го скрытого слоя, далее срабатывает 1-й скрытый слой и т. д. до $Q$-го слоя, который выдает выходные сигналы для интерпретатора и пользователя. Если не оговорено противное, то каждый выходной сигнал $q$-го слоя подается на вход всех нейронов $(q+1)$-го слоя; однако возможен вариант соединения $q$-го слоя с произвольным $(q+p)$-м слоем.

   Следует отметить, что классическим вариантом слоистых сетей являются сети прямого распространения (рис. 5.5).
3. **Сети с обратными связями.** Это сети, у которых информация с последующих слоев передается на предыдущие.

В качестве примера сетей с обратными связями на рис. 5.6 представлены так называемые *частично-рекуррентные сети* Элмана и Жордана.

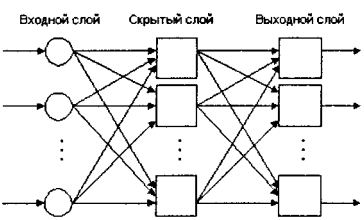


**Рис. 5.5.** Многослойная (двухслойная) сеть прямого распространения

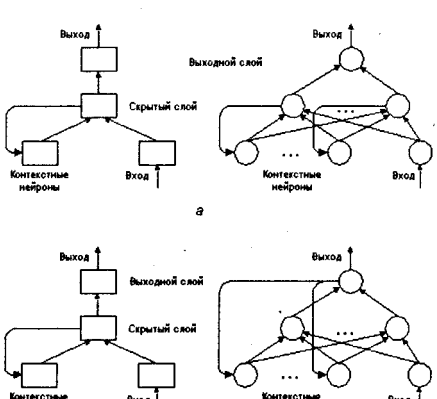


**Рис. 5.6.** Частично-рекуррентные сети: а – Элмана, б – Жордана

Известные сети можно разделить по принципу структуры нейронов на *гомогенные* (или *однородные*) и *гетерогенные*. Гомогенные сети состоят из нейронов одного типа с единой функцией активации. В гетерогенную сеть входят нейроны с различными функциями активации.

Развивая дальше вопрос о возможной классификации НС, важно отметить существование *бинарных* и *аналоговых* сетей. Первые из них оперируют с двоичными сигналами, и выход каждого нейрона может принимать только два значения: логический ноль («заторможенное» состояние) и логическая единица («возбужденное» состояние).

Еще одна классификация делит НС на *асинхронные* и *синхронные*. В первом случае в каждый момент времени свое состояние меняет лишь один нейрон. Во втором — состояние меняется сразу у целой группы нейронов, как правило, у всего слоя. Алгоритмически ход времени в НС задается итерационным выполнением над нейронами однотипных действий. Далее будут рассматриваться только синхронные НС.

Сети можно классифицировать также по числу слоев.

Теоретически число слоев и число нейронов в каждом слое может быть произвольным, однако фактически оно ограничено ресурсами компьютера или специализированной микросхемы, на которых обычно реализуются ИНС. Чем сложнее ИНС, тем масштабнее задачи, подвластные ей.

Выбор структуры ИНС осуществляется в соответствии с особенностями и сложностью задачи. Для решения некоторых отдельных типов задач уже существуют оптимальные (на сегодняшний день) конфигурации, описанные далее. Если же задача не может быть сведена ни к одному из известных типов, разработчику приходится решать сложную проблему синтеза новой конфигурации. При этом он руководствуется несколькими основополагающими принципами:

- возможности сети возрастают с увеличением числа ячеек сети, плотности связей между ними и числом выделенных слоев;
- введение обратных связей наряду с увеличением возможностей сети поднимает вопрос о так называемой *динамической устойчивости* сети;
- сложность алгоритмов функционирования сети (в том числе, например, введение нескольких типов синапсов — возбуждающих, тормозящих и др.) также способствует усилению мощи ИНС.

Вопрос о необходимых и достаточных свойствах сети для решения того или иного рода задач представляет собой целое направление нейрокомпьютерной науки. Так как проблема синтеза ИНС сильно зависит от решаемой задачи, дать общие подробные рекомендации затруднительно. В большинстве случаев оптимальный вариант получается на основе интуитивного подбора, хотя в литературе приведены доказательства того, что для любого алгоритма существует нейронная сеть, которая может его реализовать. Остановимся на этом подробнее.

Многие задачи — распознавания образов (зрительных, речевых), выполнения функциональных преобразований при обработке сигналов,

управления, прогнозирования, идентификации сложных систем — сводятся к следующей математической постановке. Необходимо построить отображение $X$ $Y$ такое, чтобы в ответ на каждый возможный входной сигнал $X$ формировался правильный выходной сигнал $Y$. Отображение задается конечным набором пар (<вход>, <известный выход>). Число таких пар (обучающих примеров) существенно меньше общего числа возможных сочетаний значений входных и выходных сигналов. Совокупность всех обучающих примеров носит название *обучающей выборки*.

В задачах распознавания образов $X$ — некоторое представление образа (изображение, вектор чисел), $Y$ — номер класса, к которому принадлежит входной образ.

В задачах управления $X$ — набор контролируемых параметров управляемого объекта, $Y$ — код, определяющий управляющее воздействие, соответствующее текущим значениям контролируемых параметров.

В задачах прогнозирования в качестве входных сигналов используются временные ряды, представляющие значения контролируемых переменных на некотором интервале времени. Выходной сигнал — множество переменных, которое является подмножеством переменных входного сигнала.

При идентификации $X$ и $Y$ представляют входные и выходные сигналы системы соответственно.

Вообще говоря, большая часть прикладных задач может быть сведена к реализации некоторого сложного многомерного функционального преобразования $X \rightarrow Y$.

В результате построения такого преобразования (отображения) необходимо добиться того, чтобы обеспечивалось формирование правильных выходных сигналов в соответствии:

- со всеми примерами обучающей выборки;
- со всеми возможными входными сигналами, которые не вошли в обучающую выборку.

Второе требование в значительной степени усложняет задачу формирования обучающей выборки. В общем виде эта задача в настоящее время еще не решена, однако во всех известных случаях было найдено частное решение. Дальнейшие рассуждения предполагают, что обучающая выборка уже сформирована.

Отметим, что теоретической основой для построения нейронных сетей является следующее утверждение: для любого множества пар входных-выходных векторов произвольной размерности $\{(\mathbf{X}^k, \mathbf{Y}^k),$

$k = 1...N\}$ существует двухслойная однородная нейронная сеть с последовательными связями, с сигмоидальными передаточными функциями и с конечным числом нейронов, которая для каждого входного вектора $\mathbf{X}^k$ формирует соответствующий ему выходной вектор $\mathbf{Y}^k$.

Таким образом, для представления многомерных функций многих переменных может быть использована однородная нейронная сеть, имеющая всего один скрытый слой, с сигмоидальными передаточными функциями нейронов.

Для оценки числа нейронов в скрытых слоях однородных нейронных сетей можно воспользоваться формулой для оценки необходимого числа синаптических весов $L_w$ (в многослойной сети с сигмоидальными передаточными функциями)

$$\frac{mN}{1+\log_2 N} \le L_w \le m\left(\frac{N}{m}+1\right)(n+m+1)+m,$$

где $n$ — размерность входного сигнала, $m$ — размерность выходного сигнала, $N$ — число элементов обучающей выборки.

Оценив необходимое число весов, можно рассчитать число нейронов в скрытых слоях. Например, число нейронов в двухслойной сети составит

$$L = \frac{L_w}{n+m}.$$

Известны и другие подобные формулы, например, вида

$$2(L+n+m) \le N \le 10(L+n+m),$$

$$\frac{N}{10} - n - m \le L \le \frac{N}{2} - n - m.$$

Точно так же можно рассчитать число нейронов в сетях с большим числом слоев, которые иногда целесообразно использовать: такие многослойные нейронные сети могут иметь меньшие размерности матриц синаптических весов нейронов одного слоя, чем двухслойные сети, реализующие то же самое отображение. К сожалению, строгая методика построения данных сетей пока отсутствует.

Отметим, что отечественному читателю приведенные результаты обычно известны в виде так называемой *теоремы о полноте*.

**Теорема о полноте.** Любая непрерывная функция на замкнутом ограниченном множестве может быть равномерно приближена функциями, вычисляемыми нейронными сетями, если функция активации нейрона дважды непрерывно дифференцируема и непрерывна.

Таким образом, нейронные сети являются универсальными аппроксимирующими системами.

Очевидно, что процесс функционирования ИНС, то есть сущность действий, которые она способна выполнять, зависит от величин синаптических связей, поэтому, задавшись определенной структурой ИНС, отвечающей какой-либо задаче, разработчик сети должен найти оптимальные значения всех переменных весовых коэффициентов (некоторые синаптические связи могут быть постоянными).

Этот этап называется обучением ИНС, и от того, насколько качественно он будет выполнен, зависит способность сети решать поставленные перед ней проблемы во время функционирования.

## Обучение нейронных сетей

Обучить нейросеть — значит, сообщить ей, чего мы от нее добиваемся. Этот процесс очень похож на обучение ребенка алфавиту. Показав ребенку изображение буквы «А», мы спрашиваем его: «Какая это буква?» Если ответ неверен, мы сообщаем ребенку тот ответ, который хотели бы от него получить: «Это буква А». Ребенок запоминает этот пример вместе с верным ответом, то есть в его памяти происходят некоторые изменения в нужном направлении. Мы будем повторять процесс предъявления букв снова и снова до тех пор, когда все буквы будут твердо запомнены. Такой процесс называют «обучение с учителем» (рис. 5.7).

При обучении сети мы действуем совершенно аналогично. У нас имеется некоторая база данных, содержащая примеры (набор рукописных изображений букв). Предъявляя изображение буквы «А» на вход сети, мы получаем от нее некоторый ответ, не обязательно верный. Нам известен и верный (желаемый) ответ — в данном случае нам хотелось бы, чтобы на выходе с меткой «А» уровень сигнала был максимален. Обычно в качестве желаемого выхода в задаче классификации берут набор (1, 0, 0, ...), где 1 стоит на выходе с меткой «А», а 0 — на всех остальных выходах. Вычисляя разность между желаемым ответом и реальным ответом сети, мы получаем (для букв русского алфавита) 33 числа — вектор ошибки. Алгоритм обучения — это набор формул, который позволяет по вектору ошибки вычислить требуемые поправки для весов сети. Одну и ту же букву (а также различные изображения одной и той же буквы) мы можем предъявлять сети много раз. В этом смысле обучение скорее напоминает повторение упражнений в спорте — тренировку.

Оказывается, что после многократного предъявления примеров веса сети стабилизируются, причем сеть дает правильные ответы на все (или почти все) примеры из базы данных. В таком случае говорят, что «сеть выучила все примеры», «сеть обучена» или «сеть натренирована». В программных реализациях можно видеть, что в процессе обучения функция ошибки (например, сумма квадратов ошибок по всем выходам) постепенно уменьшается. Когда функция ошибки достигает нуля или приемлемого малого уровня, тренировку останавливают, а полученную сеть считают натренированной и готовой к применению на новых данных.

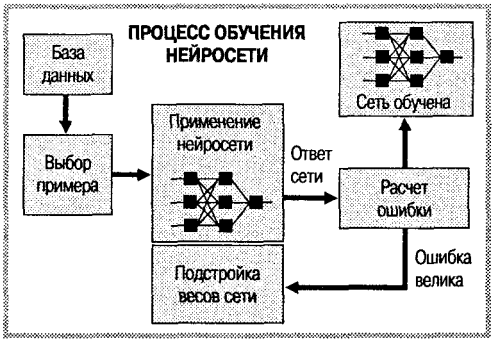


**Рис. 5.7.** Иллюстрация процесса обучения НС

Важно отметить, что вся информация, которую сеть имеет о задаче, содержится в наборе примеров. Поэтому качество обучения сети напрямую зависит от количества примеров в обучающей выборке, а также от того, насколько полно эти примеры описывают данную задачу. Так, например, бессмысленно использовать сеть для предсказания финансового кризиса, если в обучающей выборке кризисов не представлено. Считается, что для полноценной тренировки требуется хотя бы несколько десятков (а лучше сотен) примеров.

Математически процесс обучения можно описать следующим образом.

В процессе функционирования нейронная сеть формирует выходной сигнал $Y$ в соответствии с входным сигналом $X$, реализуя некоторую функцию $Y = G(X)$. Если архитектура сети задана, то вид функции $G$ определяется значениями синаптических весов и смещений сети.

Пусть решением некоторой задачи является функция $Y = F(X)$, заданная парами входных-выходных данных $(X^1, Y^1)$, $(X^2, Y^2)$, ..., $(X^N, Y^N)$, для которых $Y^k = F(X^k)$ $(k = 1, 2, ..., N)$.

Обучение состоит в поиске (синтезе) функции $G$, близкой к $F$ в смысле некоторой функции ошибки $E$ (см. рис. 5.7).

Если выбрано множество обучающих примеров — пар $(X^k, Y^k)$ (где $k = 1, 2, ..., N$) и способ вычисления функции ошибки $E$, то обучение нейронной сети превращается в задачу многомерной оптимизации, имеющую очень большую размерность, при этом, поскольку функция $E$ может иметь произвольный вид, обучение в общем случае — многоэкстремальная невыпуклая задача оптимизации.

Для решения этой задачи могут быть использованы следующие (итерационные) алгоритмы:

- алгоритмы локальной оптимизации с вычислением частных производных первого порядка — градиентный алгоритм (метод скорейшего спуска); методы с одномерной и двумерной оптимизацией целевой функции в направлении антиградиента; метод сопряженных градиентов; методы, учитывающие направление антиградиента на нескольких шагах алгоритма;
- алгоритмы локальной оптимизации с вычислением частных производных первого и второго порядка — метод Ньютона, методы оптимизации с разреженными матрицами Гессе, квазиньютоновские методы, метод Гаусса—Ньютона, метод Левенберга—Марквардта и др.;
- стохастические алгоритмы оптимизации — поиск в случайном направлении, имитация отжига, метод Монте-Карло (численный метод статистических испытаний);
- алгоритмы глобальной оптимизации (задачи глобальной оптимизации решаются с помощью перебора значений переменных, от которых зависит целевая функция).

## Алгоритм обратного распространения

Рассмотрим идею одного из самых распространенных алгоритмов обучения — алгоритма обратного распространения ошибки (back propagation). Это итеративный градиентный алгоритм обучения, который используется с целью минимизации среднеквадратичного отклонения текущего выхода от желаемого выхода в многослойных нейронных сетях.

Алгоритм обратного распространения используется для обучения многослойных нейронных сетей с последовательными связями вида рис. 5.5. Как отмечено выше, нейроны в таких сетях делятся на группы с общим входным сигналом — слои, при этом на каждый нейрон

первого слоя подаются все элементы внешнего входного сигнала, а все выходы нейронов $q$-го слоя подаются на каждый нейрон слоя $(q+1)$. Нейроны выполняют взвешенное (с синаптическими весами) суммирование элементов входных сигналов; к данной сумме прибавляется смещение нейрона. Над полученным результатом затем выполняется нелинейное преобразование с помощью активационной функции. Значение функции активации есть выход нейрона.

В многослойных сетях оптимальные выходные значения нейронов всех слоев, кроме последнего, как правило, неизвестны, и персептрон с тремя и более слоями уже невозможно обучить, руководствуясь только величинами ошибок на выходах НС. Наиболее приемлемым вариантом обучения в таких условиях оказался градиентный метод поиска минимума функции ошибки с рассмотрением сигналов ошибки от выходов НС к ее входам, то есть в направлении, обратном прямому распространению сигналов в обычном режиме работы. Этот алгоритм обучения НС получил название процедуры обратного распространения.



В данном алгоритме функция ошибки представляет собой сумму квадратов рассогласования (ошибки) желаемого выхода сети и реального. При вычислении элементов вектора градиента использован своеобразный вид производных функций активации сигмоидального типа. Алгоритм действует циклически (итеративно), и его циклы принято называть *эпохами*. На каждой эпохе на вход сети поочередно подаются все обучающие наблюдения, выходные значения сети сравниваются с целевыми значениями и вычисляется ошибка. Значения ошибки, а также градиента поверхности ошибок используются для корректировки весов, после чего все действия повторяются. Начальная конфигурация сети выбирается случайным образом, и процесс обучения прекращается либо когда пройдено определенное количество эпох, либо когда ошибка достигнет некоторого определенного уровня малости, либо когда ошибка перестанет уменьшаться (пользователь может сам выбрать нужное условие остановки).

Приведем словесное описание алгоритма.

- **Шаг 1.** Весам сети присваиваются небольшие начальные значения.
- **Шаг 2.** Выбирается очередная обучающая пара ($X$, $Y$) из обучающего множества; вектор **X** подается на вход сети.
- **Шаг 3.** Вычисляется выход сети.
- **Шаг 4.** Вычисляется разность между требуемым (целевым, $Y$) и реальным (вычисленным) выходом сети.

- **Шаг 5.** Веса сети корректируются так, чтобы минимизировать ошибку (сначала веса выходного слоя, затем, с использованием правила дифференцирования сложной функции и отмеченного своеобразного вида производной сигмоидальной функции, — веса предыдущего слоя и т. п.).
- **Шаг 6.** Шаги со 2-го по 5-й повторяются для каждой пары обучающего множества до тех пор, пока ошибка на всем множестве не достигнет приемлемой величины.

Шаги 2 и 3 подобны тем, которые выполняются в уже обученной сети.

Вычисления в сети выполняются послойно. На шаге 3 каждый из выходов сети вычитается из соответствующего компонента целевого вектора с целью получения ошибки. Эта ошибка используется на шаге 5 для коррекции весов сети.

Шаги 2 и 3 можно рассматривать как «проход вперед», так как сигнал распространяется по сети от входа к выходу. Шаги 4 и 5 составляют «обратный проход», поскольку здесь вычисляемый сигнал ошибки распространяется обратно по сети и используется для подстройки весов.

Классический метод обратного распространения относится к алгоритмам с линейной сходимостью. Его известными недостатками являются: невысокая скорость сходимости (большое число итераций, требуемых для достижения минимума функции ошибки), возможность сходиться не к глобальному, а к локальным решениям (локальным минимумам отмеченной функции). Возможен также *паралич сети*, при котором большинство нейронов функционируют при очень больших значениях аргумента функции активации, то есть на ее пологом участке (поскольку ошибка пропорциональна производной, которая на данных участках мала, то процесс обучения практически замирает).

Для устранения этих недостатков были предложены многочисленные модификации алгоритма обратного распространения, которые связаны с использованием различных функций ошибки, различных процедур определения направления и величины шага и т. п.

## Переобучение и обобщение нейронных сетей

Одна из наиболее серьезных трудностей изложенного подхода заключается в том, что таким образом мы минимизируем не ту ошибку, которую на самом деле нужно минимизировать, — ошибку, которую можно ожидать от сети, когда ей будут подаваться совершенно но-

вые наблюдения. Иначе говоря, мы хотели бы, чтобы нейронная сеть обладала способностью обобщать результат на новые наблюдения. В действительности сеть обучается минимизировать ошибку на обучающем множестве, и в отсутствие идеального и бесконечно большого обучающего множества это совсем не то же самое, что минимизировать «настоящую» ошибку на поверхности ошибок в заранее неизвестной модели явления.

Сильнее всего это различие проявляется в проблеме переобучения. При слишком близкой подгонке явление проще будет продемонстрировать не для нейронной сети, а на примере аппроксимации посредством полиномов — при этом суть явления абсолютно та же.

Полином (или многочлен) — это выражение, содержащее только константы и целые степени независимой переменной. Примеры полиномов:

$$y = 2x + 3,$$

$$y = 3x^2 + 4x + 1.$$



Графики полиномов могут иметь различную форму. Чем выше степень многочлена (и, тем самым, чем больше членов в него входит), тем более сложной может быть эта форма. Если у нас есть некоторые данные, мы можем поставить цель подогнать к ним полиномиальную кривую (модель) и получить, таким образом, объяснение для имеющейся зависимости. Однако наши данные могут быть зашумлены, поэтому нельзя считать, что самая лучшая модель задается кривой, которая в точности проходит через все имеющиеся точки. Полином низкого порядка может быть недостаточно гибким средством для аппроксимации данных, в то время как полином высокого порядка может оказаться чересчур гибким и будет точно следовать данным, принимая при этом замысловатую форму, не имеющую никакого отношения к форме настоящей зависимости.

Нейронная сеть сталкивается с точно такой же трудностью. Сети с большим числом весов моделируют более сложные функции и, следовательно, склонны к переобучению. Сеть же с небольшим числом весов может оказаться недостаточно гибкой, чтобы смоделировать имеющуюся зависимость. Например, сеть без промежуточных слоев на самом деле моделирует обычную линейную функцию.

Как же выбрать «правильную» степень сложности для сети? Более сложная сеть почти всегда дает меньшую ошибку, но это может свидетельствовать не о хорошем качестве модели, а о переобучении.

Ответ состоит в том, чтобы использовать механизм контрольной кросс-проверки, при котором часть обучающих наблюдений резер-

вируется и не используется в процессе обучения по алгоритму обратного распространения. Вместо этого по мере работы алгоритма она используются для независимой проверки результата. В самом начале работы ошибка сети на обучающем и проверочном множествах будет одинаковой, но по мере того как сеть обучается, ошибка обучения, естественно, убывает, и пока обучение уменьшает действительную функцию ошибок, ошибка на проверочном множестве также будет убывать. Если же контрольная ошибка перестала убывать или даже стала расти, это указывает на то, что сеть начала слишком близко аппроксимировать данные и обучение следует остановить.

Отмеченное явление чересчур точной аппроксимации в процессе обучения и называется *переобучением*. Если такое случается, то рекомендуется уменьшить число скрытых элементов и/или слоев, ибо сеть является слишком мощной для данной задачи. Если же сеть, наоборот, была взята недостаточно сложной для того, чтобы моделировать имеющуюся зависимость, то переобучения, скорее всего, не произойдет и обе ошибки — обучения и проверки — не достигнут достаточного уровня малости.

Описанные проблемы с локальными минимумами и выбором размера сети приводят к тому, что при практической работе с нейронными сетями, как правило, приходится экспериментировать с большим числом различных сетей, порой обучая каждую из них по несколько раз (чтобы не быть введенным в заблуждение локальными минимумами) и сравнивая полученные результаты. Главным показателем качества результата здесь является контрольная ошибка. При этом, в соответствии с общенаучным принципом, согласно которому при прочих равных следует предпочесть более простую модель, из двух сетей с приблизительно равными ошибками контроля имеет смысл выбрать ту, которая меньше.

Необходимость многократных экспериментов ведет к тому, что проверочное множество начинает играть ключевую роль в выборе модели, то есть становится частью процесса обучения. Тем самым ослабляется его роль как независимого критерия качества модели — при большом числе экспериментов есть риск выбрать «удачную» сеть, дающую хороший результат на проверочном (контрольном) множестве.

Для того чтобы придать окончательной модели должную надежность, часто (по крайней мере, когда объем обучающих данных это позволяет) поступают так: резервируют еще одно — тестовое (или тестирующее) множество наблюдений. Итоговая модель тестируется на данных из этого множества, чтобы убедиться, что результаты, дос-

тигнутые на обучающем и контрольном множествах, реальны. Разумеется, для того чтобы хорошо играть свою роль, тестовое множество должно быть использовано только один раз: если его использовать повторно для корректировки процесса обучения, то оно фактически превратится в контрольное множество.

## Обучение без учителя

Рассмотренный алгоритм обучения нейронной сети с помощью процедуры обратного распространения подразумевает наличие некоего внешнего звена, предоставляющего сети кроме входных также и целевые выходные образы. Алгоритмы, пользующиеся подобной концепцией, называются алгоритмами *обучения с учителем*. Для их успешного функционирования необходимо наличие экспертов, создающих на предварительном этапе для каждого входного образа эталонный выходной. Так как создание искусственного интеллекта движется по пути копирования природных прообразов, ученые не прекращают спор на тему, можно ли считать алгоритмы обучения с учителем натуральными или же они полностью искусственны. Например, обучение человеческого мозга, на первый взгляд, происходит без учителя: на зрительные, слуховые, тактильные и прочие рецепторы поступает информация извне, и внутри нервной системы происходит некая самоорганизация. Однако нельзя отрицать и того, что в жизни человека немало учителей — и в буквальном, и в переносном смысле, — которые координируют внешние воздействия. Вместе с тем чем бы ни закончился спор приверженцев этих двух концепций обучения, они обе имеют право на существование.



Главная черта, делающая обучение без учителя привлекательным, — это его «самостоятельность». Процесс обучения, как и в случае обучения с учителем, заключается в подстройке весов синапсов. Очевидно, что подстройка весов синапсов может проводиться только на основании информации, доступной в нейроне, то есть информации о его состоянии, уже имеющихся весовых коэффициентах и поданном входном векторе **X**. Исходя из этого и, что более важно, по аналогии с известными принципами самоорганизации нервных клеток построены алгоритмы обучения Хебба и Кохонена. Общая идея данных алгоритмов заключается в том, что в процессе самообучения путем соответствующей коррекции весовых коэффициентов усиливаются связи между возбужденными нейронами.

Следует отметить, что вид откликов $Y$ на каждый класс входных образов неизвестен заранее и будет представлять собой произвольное сочетание состояний нейронов выходного слоя, обусловленное

случайным распределением весов на стадии инициализации. Вместе с тем сеть способна обобщать схожие образы, относя их к одному классу. Тестирование обученной сети позволяет определить топологию классов в выходном слое.

# Применение нейросетей

*После того как сеть обучена, мы можем применять ее для решения различных задач* (рис. 5.8).

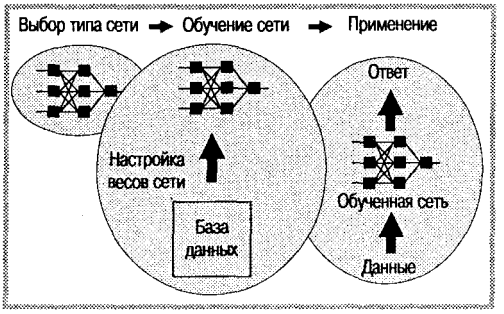


**Рис. 5.8.** Этапы нейросетевого проекта

Важнейшая особенность человеческого мозга состоит в том, что, однажды обучившись определенному процессу, он может верно действовать и в тех ситуациях, в которых он не бывал в процессе обучения. Например, мы можем читать почти любой почерк, даже если видим его первый раз в жизни. Так же и нейросеть, грамотным образом обученная, может с большой вероятностью правильно реагировать на новые, не предъявленные ей ранее данные. Например, мы можем нарисовать букву «А» другим почерком, а затем предложить нашей сети классифицировать новое изображение. Веса обученной сети хранят достаточно много информации о сходстве и различиях букв, поэтому можно рассчитывать на правильное опознание и нового варианта изображения.

## Области применения нейросетей: классификация

Отметим, что задачи классификации (типа распознавания букв) очень плохо алгоритмизируются. Если в случае распознавания букв верный ответ очевиден для нас заранее, то в более сложных практических задачах обученная нейросеть выступает как эксперт, обладающий большим опытом и способный дать ответ на трудный вопрос.

Примером такой задачи служит медицинская диагностика, где сеть может учитывать большое количество числовых параметров (энцефалограмма, давление, вес и т. д.). Конечно, «мнение» сети в этом случае нельзя считать окончательным.

Классификация предприятий по степени их перспективности — это уже привычный способ использования нейросетей в практике западных компаний (деление компаний на перспективные и убыточные). При этом сеть также использует множество экономических показателей, сложным образом связанных между собой.

Нейросетевой подход особенно эффективен в задачах экспертной оценки по той причине, что он сочетает в себе способность компьютера к обработке чисел и способность мозга к обобщению и распознаванию. Говорят, что у хорошего врача способность к распознаванию в своей области столь велика, что он может провести приблизительную диагностику уже по внешнему виду пациента. Можно согласиться также, что опытный трейдер чувствует направление движения рынка по виду графика. Однако в первом случае все факторы наглядны, то есть характеристики пациента мгновенно воспринимаются мозгом как «бледное лицо», «блеск в глазах» и т. д. Во втором же случае учитывается только один фактор, показанный на графике, — курс акций за определенный период времени. Нейросеть позволяет обрабатывать огромное количество факторов (до нескольких тысяч) независимо от их наглядности — это универсальный «хороший врач», который может поставить свой диагноз в любой области.



## Кластеризация и поиск зависимостей

Помимо задач классификации нейросети широко используются для поиска зависимостей в данных и кластеризации.

Например, нейросеть на основе *метода группового учета аргументов* (МГУА) позволяет, базируясь на обучающей выборке, построить зависимость одного параметра от других в виде полинома. Такая сеть может не только мгновенно выучить таблицу умножения, но и найти сложные скрытые зависимости в данных (например, финансовых), которые не обнаруживаются стандартными статистическими методами.

*Кластеризация* — это разбиение набора примеров на несколько компактных областей (кластеров), причем число кластеров заранее неизвестно. Кластеризация позволяет представить неоднородные данные в более наглядном виде и далее использовать различные методы для исследования каждого кластера. Например, таким образом

можно быстро выявить фальсифицированные страховые случаи или недобросовестные предприятия.

### Прогнозирование

Задачи прогнозирования особенно важны для практики, в частности для финансовых приложений, поэтому поясним способы применения нейросетей в этой области более подробно.

Рассмотрим практическую задачу, ответ в которой неочевиден — задачу прогнозирования курса акций на один день вперед.

Пусть у нас имеется база данных, содержащая значения курса за последние 300 дней. Простейший вариант в данном случае — попытаться построить прогноз завтрашней цены на основе курсов за последние несколько дней. Понятно, что прогнозирующая сеть должна иметь всего один выход и столько входов, сколько предыдущих значений мы хотим использовать для прогноза — например, 4 последних значения. Составить обучающий пример очень просто — входными значениями будут курсы за 4 последовательных дня, а желаемым выходом — известный нам курс в следующий за этими четырьмя день. Тогда каждая строка таблицы с обучающей последовательностью (выборкой) представляет собой обучающий пример, где первые 4 числа — входные значения сети, а пятое число — желаемое значение выхода.

Заметим, что объем обучающей выборки зависит от выбранного количества входов. Если сделать 299 входов, то такая сеть потенциально могла бы строить лучший прогноз, чем сеть с 4 входами, однако в этом случае мы имеем дело с огромным массивом данных, что делает обучение и использование сети практически невозможным. При выборе числа входов следует учитывать это, выбирая разумный компромисс между глубиной предсказания (число входов) и качеством обучения (объем тренировочного набора).

## Персептроны

В качестве научного предмета искусственные нейронные сети впервые заявили о себе в 40-е годы. Стремясь воспроизвести функции человеческого мозга, исследователи создали простые аппаратные (а позже программные) модели биологического нейрона и системы его соединений. Когда нейрофизиологи достигли более глубокого понимания нервной системы человека, эти ранние попытки стали восприниматься как весьма грубые аппроксимации. Тем не менее на этом пути были достигнуты впечатляющие результаты, стимулиро-

вавшие дальнейшие исследования, приведшие к созданию более изощренных сетей.

Первое систематическое изучение искусственных нейронных сетей было предпринято Маккалохом и Питтсом в 1943 г. Позднее они исследовали сетевые парадигмы для распознавания изображений, подвергаемых сдвигам и поворотам, используя при этом простую нейронную модель, показанную на рис. 5.9. Элемент $\Sigma$ умножает каждый вход $x_i$ на вес $w_i$ и суммирует взвешенные входы. Если эта сумма больше заданного порогового значения, выход равен единице, в противном случае — нулю. Эти системы (и множество им подобных) получили название *персептронов*. Они состоят из одного слоя искусственных нейронов, соединенных с помощью весовых коэффициентов с множеством входов (см. рис. 5.10), хотя в принципе описываются и более сложные системы.



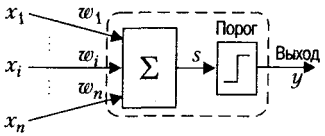


**Рис. 5.9.** Персептронный нейрон

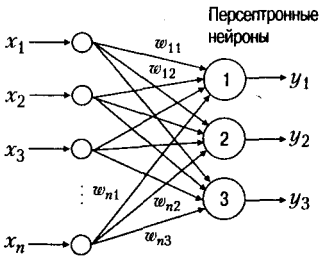


**Рис. 5.10.** Персептрон со многими выходами

В 60-е годы персептроны вызвали большой интерес и оптимизм. Розенблатт доказал замечательные теоремы об обучении персептронов, приводимые ниже. Уидроу представил ряд убедительных демонстраций систем персептронного типа, и исследователи во всем мире стремились изучить возможности этих систем. Первоначальная эйфория сменилась разочарованием, когда оказалось, что персептроны не способны обучиться решению ряда простых задач. Минский строго проанализировал эту проблему и показал, что имеются жесткие ограничения на то, что могут выполнять однослойные пер-

септроны, и, следовательно, на то, чему они могут обучаться. Так как в то время методы обучения многослойных сетей не были известны, исследователи перешли в более многообещающие области, и исследования в области нейронных сетей пришли в упадок. Недавнее открытие методов обучения многослойных сетей повлияло на возрождение интереса и исследовательских усилий в большей степени, чем какой-либо иной фактор.

Работа Минского, возможно, и охладила пыл энтузиастов персептрона, но обеспечила время для необходимой консолидации и развития лежащей в основе теории. Важно отметить, что анализ Минского не был опровергнут. Он остается важным исследованием и должен изучаться, чтобы ошибки 60-х годов не повторились.

Несмотря на свои ограничения, персептроны широко изучались (хотя не слишком широко использовались). Теория персептронов является основой для многих других типов искусственных нейронных сетей, в силу чего они являются логической исходной точкой для изучения искусственных нейронных сетей.

Рассмотрим в качестве примера трехнейронный персептрон (см. рис. 5.10), нейроны которого имеют активационную функцию в виде единичного скачка.

На $n$ входов поступают входные сигналы, проходящие по синапсам на три нейрона, образующие единственный слой этой сети и выдающие три выходных сигнала:

$$y_j = f\left(\sum_{i=1}^{n} x_i w_{ij}\right), \quad j = 1...3.$$

Очевидно, что все весовые коэффициенты синапсов одного слоя нейронов можно свести в матрицу **W**, в которой каждый элемент $w_{ij}$ задает величину $i$-й синаптической связи $j$-го нейрона. Таким образом, процесс, происходящий в нейронной сети, может быть записан в матричной форме:

$$\mathbf{Y} = f(\mathbf{XW}),$$

где **X** и **Y** — соответственно, входной и выходной сигнальные векторы (здесь и далее под вектором понимается вектор-строка), $f(\mathbf{S})$ — активационная функция, применяемая поэлементно к компонентам вектора **S**.

На рис. 5.11 представлен двухслойный персептрон, полученный из персептрона, изображенного на рис. 5.10, путем добавления второго слоя, состоящего из двух нейронов.

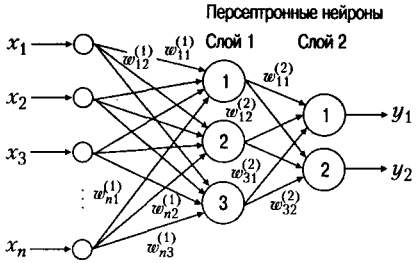


**Рис. 5.11.** Двухслойный персептрон

Здесь уместно отметить важную роль нелинейности активационной функции. Если бы она не обладала данным свойством или не входила в алгоритм работы каждого нейрона, результат функционирования любой $Q$-слойной нейронной сети с весовыми матрицами $\mathbf{W}^{(q)}$ для каждого слоя $q = 1...Q$ сводился бы к перемножению входного вектора сигналов $\mathbf{X}$ на матрицу:

$$\mathbf{W}_{(\Sigma)} = \mathbf{W}^{(1)} ... \mathbf{W}^{(q)} ... \mathbf{W}^{(Q)}.$$

Фактически такая $Q$-слойная нейронная сеть эквивалентна сети с одним скрытым слоем и с весовой матрицей единственного слоя $\mathbf{W}_{(\Sigma)}$:

$$\mathbf{Y} = \mathbf{X}\,\mathbf{W}_{(\Sigma)}.$$

Работа персептрона сводится к классификации (обобщению) входных сигналов, принадлежащих $n$-мерному гиперпространству, по некоторому числу классов. С математической точки зрения это происходит путем разбиения гиперпространства гиперплоскостями. Для случая однослойного персептрона

$$\sum_{i=1}^{n} x_i w_{ij} = \theta_j, j = 1, 2, ..., m.$$

Каждая полученная область является областью определения отдельного класса. Число таких классов для персептрона не превышает $2^n$, где $n$ — число его входов. Однако не все из классов могут быть выделены данной нейронной сетью.

Например, однослойный персептрон, состоящий из одного нейрона с двумя входами, не может реализовать логическую функцию «исключающее ИЛИ», то есть не способен разделить плоскость (двумерное гиперпространство) на две полуплоскости так, чтобы осуществить классификацию входных сигналов по классам $A$ и $B$ (табл. 5.3).

Уравнение сети для этого случая

$$x_1 w_1 + x_2 w_2 = \theta$$

является уравнением прямой (одномерной гиперплоскости), которая ни при каких условиях не может разделить плоскость так, чтобы точки из множества входных сигналов, принадлежащие разным классам, оказались по разные стороны от прямой (рис. 5.12). Невозможность реализации однослойным персептроном этой функции получила название *проблемы исключающего ИЛИ*.

**Таблица 5.3.** Логическая функция «исключающее ИЛИ»

| $x_2$ | $x_1$ | |
|---|---|---|
| | **0** | **1** |
| **0** | *B* | *A* |
| **1** | *A* | *B* |

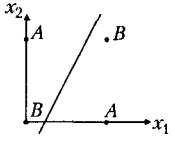


**Рис. 5.12.** Линейная неразделимость функции «исключающее ИЛИ»

Отметим, что функции, которые не реализуются однослойным персептроном, называются *линейно неразделимыми*. Решение задач, подпадающих под это ограничение, заключается в применении сетей с двумя и более слоями или сетей с нелинейными синапсами, однако и тогда существует вероятность, что корректное разделение некоторых входных сигналов на классы невозможно.

Обучение персептрона сводится к формированию весов связей между первым и вторым (см. рис. 5.11) слоями в соответствии со следующим алгоритмом.

- **Шаг 1.** Проинициализировать элементы весовой матрицы (обычно небольшими случайными значениями).
- **Шаг 2.** Подать на входы один из входных векторов, которые сеть должна научиться различать, и вычислить ее выход.
- **Шаг 3.** Если выход правильный, перейти на шаг 4. Иначе — вычислить разницу между идеальным $d$ и полученным $Y$ значениями выхода:

  $\delta = d - Y$.

  Модифицировать веса в соответствии с формулой

$w_{ij}(t + 1) = w_{ij}(t) + \eta\delta x_i$,

где $t$ и $(t+1)$ — номера, соответственно, текущей и следующей итераций; $\eta$ — коэффициент скорости обучения, $0 < \eta < 1$; $i$ — номер входа; $j$ — номер нейрона в слое.

Очевидно, что если $d > Y$, то весовые коэффициенты будут увеличены и тем самым уменьшат ошибку. В противном случае они будут уменьшены, и $Y$ тоже уменьшится, приближаясь к $d$.

- **Шаг 4.** Цикл с шага 2, пока сеть не перестанет ошибаться.

На втором шаге на разных итерациях поочередно в случайном порядке предъявляются все возможные входные векторы. К сожалению, нельзя заранее определить число итераций, которое потребуется выполнить, а в некоторых случаях невозможно и гарантировать полный успех.

Сходимость рассмотренной процедуры устанавливается теоремами, утверждающими, что:

- для любой классификации обучающей последовательности можно подобрать такой набор (из бесконечного набора) элементарных нейронов, в котором будет осуществлено разделение обучающей последовательности при помощи линейного решающего правила;
- если относительно задуманной классификации можно найти набор элементов, в котором существует решение, то в рамках этого набора оно будет достигнуто за конечный промежуток времени.

## Нейронные сети встречного распространения

Объединение разнотипных нейронных структур в единой архитектуре зачастую приводит к свойствам, которых нет у них по отдельности. Причем именно каскадные соединения нейронных структур, специализирующихся на решении различных задач, позволяют решить проблему комплексно.

Нейронные сети встречного распространения (двунаправленные нейронные сети, в системе MATLAB именуемые Learning Vector Quantization Network, или LVQ-network), состоящие из входного слоя нейронов и так называемых слоев нейронов Кохонена и Гроссберга, по своим характеристикам существенно превосходят возможности сетей с одним скрытым слоем нейронов. Так, время их обучения задачам распознавания и кластеризации более чем в сто раз меньше времени обучения аналогичным задачам сетей с обратным распро-

странением. Это может быть полезно в тех приложениях, где долгая обучающая процедура невозможна.

Важными свойствами сети встречного распространения являются ее хорошие способности к обобщению, позволяющие получать правильный выход даже при неполном или зашумленном входном векторе. Это позволяет эффективно использовать данную сеть для распознавания и восстановления образов, а также для усиления сигналов.

В процессе обучения сети встречного распространения входные векторы ассоциируются с соответствующими выходными векторами. Эти векторы могут быть двоичными или непрерывными. После обучения сеть формирует выходные сигналы, соответствующие входным сигналам. Обобщающая способность сети дает возможность получать правильный выход, когда входной вектор неполон или искажен.

Сеть встречного распространения имеет два слоя с последовательными связями. Первый слой — слой Кохонена, второй — слой Гроссберга. Каждый элемент входного сигнала подается на все нейроны слоя Кохонена. Каждый нейрон слоя Кохонена соединен со всеми нейронами слоя Гроссберга. Отличие сети встречного распространения от других многослойных сетей с последовательными связями состоит в операциях, выполняемых нейронами Кохонена и Гроссберга.

В режиме функционирования сети предъявляется входной сигнал **X** и формируется выходной сигнал **Y**. В режиме обучения на вход сети подается входной сигнал и веса корректируются, чтобы сеть выдавала требуемый выходной сигнал.

## Функционирование сети

**Слой Кохонена.** В своей простейшей форме слой Кохонена функционирует по правилу «победитель получает все». Для данного входного вектора один и только один нейрон Кохонена выдает логическую единицу, все остальные выдают ноль.

Выход каждого нейрона Кохонена является просто суммой взвешенных элементов входных сигналов:

$$s_j = w_{1j}x_1 + w_{2j}x_2 + \ldots + w_{nj}x_n = \sum_i x_i w_{ij},$$

где $s_j$ — выход $j$-го нейрона Кохонена; $\mathbf{W}_j = (w_{1j}, w_{2j}, \ldots w_{nj})$ — вектор весов $j$-го нейрона Кохонена; $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ — вектор входного сигнала, или в векторно-матричной форме:

$$\mathbf{S} = \mathbf{XW},$$

где **S** — вектор выходов слоя Кохонена.

Нейрон Кохонена с максимальным значением $s_j$ является «победителем». Его выход равен единице, у остальных он равен нулю.

**Слой Гроссберга.** Слой Гроссберга функционирует в сходной манере. Его выход является взвешенной суммой выходов слоя Кохонена (то есть он является слоем нейронов с линейными активационными функциями).

*Если слой Кохонена функционирует таким образом, что лишь один* выход равен единице, а остальные равны нулю, то каждый нейрон слоя Гроссберга выдает величину веса, который связывает этот нейрон с единственным нейроном Кохонена, чей выход отличен от нуля.

**Предварительная обработка входных сигналов.** Рассматриваемая НС требует предварительной обработки входных векторов путем их нормализации. Такая нормализация выполняется путем деления каждого компонента входного вектора на длину вектора (квадратный корень из суммы квадратов компонентов вектора). Это превращает входной вектор в единичный вектор с тем же направлением, то есть в вектор единичной длины в $n$-мерном пространстве.



## Обучение слоя Кохонена

Слой Кохонена классифицирует входные векторы в группы схожих векторов. Это достигается с помощью такой подстройки весов, что близкие входные векторы активизируют один и тот же нейрон данного слоя (затем задачей слоя Гроссберга является получение требуемых выходов).

Слой Кохонена обучается без учителя (самообучается). В результате обучения слой приобретает способность разделять несхожие входные векторы. Какой именно нейрон будет активизироваться при предъявлении конкретного входного сигнала, заранее трудно предсказать.

При обучении слоя Кохонена на вход подается входной вектор и вычисляются его скалярные произведения с векторами весов всех нейронов. Скалярное произведение является мерой сходства между входным вектором и вектором весов. Нейрон с максимальным значением скалярного произведения объявляется «победителем», и его веса подстраиваются (весовой вектор приближается к входному).

Уравнение, описывающее процесс обучения, имеет вид

$$w_н = w_с + \eta(x - w_с),$$

где $w_н$ — новое значение веса, соединяющего входной компонент $x$ с выигравшим нейроном, $w_с$ — предыдущее значение этого веса, $\eta$ — коэффициент скорости обучения.

Каждый вес, связанный с выигравшим нейроном Кохонена, изменяется пропорционально разности между его величиной и величиной входа, к которому он присоединен. Направление изменения минимизирует разность между весом и соответствующим элементом входного сигнала.

Коэффициент скорости обучения $\eta$ вначале обычно полагается равным 0,7 и может затем постепенно уменьшаться в процессе обучения. Это позволяет делать большие начальные шаги для быстрого грубого обучения и меньшие шаги при подходе к окончательной величине.

Если бы с каждым нейроном Кохонена ассоциировался один входной вектор, то слой Кохонена мог бы быть обучен с помощью одной коррекции на вес ($\eta = 1$). Как правило обучающее множество включает много сходных между собой входных векторов, и сеть должна быть обучена активизировать один и тот же нейрон Кохонена для каждого из них. Веса этого нейрона должны получаться усреднением входных векторов, которые должны его активизировать.

## Обучение слоя Гроссберга

Выходы слоя Кохонена подаются на входы нейронов слоя Гроссберга. Выходы нейронов вычисляются, как при обычном функционировании. Далее каждый вес корректируется лишь в том случае, если он соединен с нейроном Кохонена, имеющим ненулевой выход. Величина коррекции веса пропорциональна разности между весом и требуемым выходом нейрона Гроссберга.

Обучение слоя Гроссберга — это обучение с учителем, алгоритм использует заданные желаемые выходы.

В полной модели сети встречного распространения имеется возможность получать выходные сигналы по входным и наоборот. Этим двум действиям соответствуют прямое и обратное распространение сигналов.

**Области применения:** распознавание образов, восстановление образов (ассоциативная память), сжатие данных (с потерями).

**Недостатки:** сеть не дает возможности строить точные аппроксимации (точные отображения). В этом сеть значительно уступает сетям с обратным распространением ошибки. К недостаткам модели также следует отнести слабую теоретическую проработку модификаций сети встречного распространения.

**Преимущества:** сеть встречного распространения проста. Она дает возможность извлекать статистические свойства из множеств вход-

ных сигналов. Кохоненом показано, что для полностью обученной сети вероятность того, что случайно выбранный входной вектор (в соответствии с функцией плотности вероятности входного множества) будет ближайшим к любому заданному весовому вектору, равна $1/p$; $p$ — число нейронов Кохонена.

Сеть быстро обучается. Время обучения по сравнению с сетью с обратным распространением может быть в 100 раз меньше.

По своим возможностям строить отображения сеть встречного распространения значительно превосходит однослойные перцептроны.

Сеть полезна для приложений, в которых требуется быстрая начальная аппроксимация.

Сеть дает возможность строить функцию и обратную к ней функцию, что находит применение при решении практических задач.

### Модификации

Сети встречного распространения могут различаться способами определения начальных значений синаптических весов. Так, кроме традиционных случайных значений из заданного диапазона могут быть использованы значения в соответствии с известным методом выпуклой комбинации.

Для повышения эффективности обучения применяется добавление шума к входным векторам.

Еще один метод повышения эффективности обучения — наделение каждого нейрона «чувством справедливости». Если нейрон становится победителем чаще, чем $1/p$ ($p$ — число нейронов Кохонена), то ему временно увеличивают порог, давая тем самым обучаться и другим нейронам.

Кроме *метода аккредитации*, при котором для каждого входного вектора активизируется лишь один нейрон Кохонена, может быть использован *метод интерполяции*, при использовании которого целая группа нейронов Кохонена, имеющих наибольшие выходы, может передавать свои выходные сигналы в слой Гроссберга. Этот метод повышает точность отображений, реализуемых сетью, и реализован в так называемых *самоорганизующихся картах*.

## Нейронные сети Хопфилда и Хэмминга

Среди различных конфигураций искусственных нейронных сетей встречаются такие, при классификации которых по принципу обучения, строго говоря, не подходят ни обучение с учителем, ни обу-

чение без учителя. В таких сетях весовые коэффициенты синапсов рассчитываются только однажды перед началом функционирования сети на основе информации об обрабатываемых данных, и все обучение сети сводится именно к этому расчету. С одной стороны, предъявление априорной информации можно расценивать как помощь учителя, но с другой — сеть фактически просто запоминает образцы до того, как на ее вход поступают реальные данные, и не может изменять свое поведение, поэтому говорить о звене обратной связи с внешним миром (учителем) не приходится. Из сетей с подобной логикой работы наиболее известны *сеть Хопфилда* и *сеть Хэмминга* (представляющие собой разновидности сетей с обратными связями), которые обычно используются для организации ассоциативной памяти. Далее речь пойдет именно о них.

Структурная схема *сети Хопфилда* приведена на рис. 5.13. Она состоит из единственного слоя нейронов, число которых является одновременно числом входов и выходов сети. Каждый нейрон связан синапсами со всеми остальными нейронами, а также имеет один входной синапс, через который осуществляется ввод сигнала. Выходные сигналы, как обычно, образуются на аксонах.

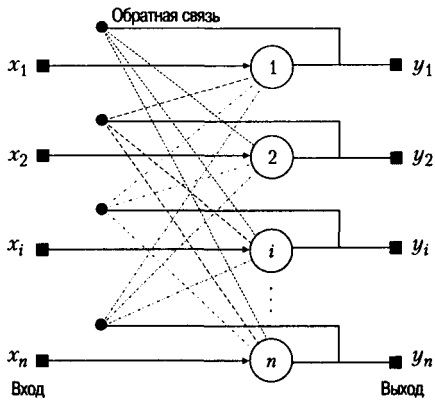


**Рис. 5.13.** Структурная схема сети Хопфилда

Задача, решаемая данной сетью в качестве ассоциативной памяти, как правило, формулируется следующим образом. Известен некоторый набор двоичных сигналов (изображений, звуковых оцифровок, прочих данных, описывающих некие объекты или характеристики процессов), которые считаются образцовыми. Сеть должна уметь из произвольного неидеального сигнала, поданного на ее вход, выде-

лить («вспомнить» по частичной информации) соответствующий образец (если такой есть) или «дать заключение» о том, что входные данные не соответствуют ни одному из образцов. В общем случае любой сигнал может быть описан вектором $\mathbf{X} = \{x_i: i = 1, 2, ..., n\}$, $n$ — число нейронов в сети и размерность входных и выходных векторов. Каждый элемент $x_i$ равен либо +1, либо −1. Обозначим вектор, описывающий $k$-й образец, через $\mathbf{X}^k$, а его компоненты, соответственно, как $x_i^k$, $k = 1, 2, ..., m$, где $m$ — в данном случае число образцов. Когда сеть распознает (или «вспомнит») какой-либо образец на основе предъявленных ей данных, ее выходы будут содержать именно его, то есть $\mathbf{Y} = \mathbf{X}^k$, где $\mathbf{Y}$ — вектор выходных значений сети: $\mathbf{Y} = \{y_i: i = 1, 2, ..., n\}$. В противном случае выходной вектор не совпадет ни с одним образцовым.

Если, например, сигналы представляют собой некие изображения, то, отобразив в графическом виде данные с выхода сети, можно будет увидеть картинку, полностью совпадающую с одной из образцовых (в случае успеха) или же «вольную импровизацию» сети (в случае неудачи).



На стадии инициализации сети весовые коэффициенты синапсов устанавливаются следующим образом:

$$w_{ij} = \begin{cases} \sum_{k=1}^{m} x_i^k x_j^k, & i \neq j, \\ 0, & i = j \end{cases}$$

Здесь $i$ и $j$ — индексы, соответственно, предсинаптического и постсинаптического нейронов; $x_i^k$, $x_j^k$ — $i$-й и $j$-й элементы вектора $k$-го образца.

Алгоритм функционирования сети следующий ($t$ — номер итерации):

1. На входы сети подается неизвестный сигнал. Фактически его ввод осуществляется непосредственной установкой значений аксонов:

   $y_i(0) = x_i$, $i = 1, 2, ..., n$,

   поэтому обозначение на схеме сети входных синапсов в явном виде носит чисто условный характер. Ноль в скобке справа от $y_i$ означает нулевую итерацию в цикле работы сети.

2. Рассчитываются новое состояние нейронов

   $$s_j(t+1) = \sum_{i=1}^{n} w_{ij} y_i(t), \quad j = 1, 2, ..., n$$

   и новые значения аксонов

   $$y_j(t+1) = f[s_j(t+1)],$$

где $f$ — пороговая активационная функция с порогом $\theta = 0$ (см. табл. 5.1).

3. Проверка, изменились ли выходные значения аксонов за последнюю итерацию. Если да — переход к п. 2, иначе (если выходы стабилизировались) — конец работы. При этом выходной вектор представляет собой образец, наилучшим образом сочетающийся с входными данными.

Таким образом, когда подается новый вектор, сеть переходит из вершины в вершину, пока не стабилизируется. Устойчивая вершина определяется сетевыми весами и текущими входами. Если входной вектор частично неправилен или неполон, сеть стабилизируется в вершине, ближайшей к желаемой.

Показано, что достаточным условием устойчивой работы такой сети является выполнение условий

$$w_{ij} = w_{ji}, \quad w_{ii} = 0.$$

Как говорилось выше, иногда сеть не может провести распознавание и выдает на выходе несуществующий образ. Это связано с проблемой ограниченности возможностей сети. Для сети Хопфилда число запоминаемых образов $N$ не должно превышать величины, примерно равной $0{,}15n$. Кроме того, если два образа $A$ и $B$ очень похожи, они, возможно, будут вызывать у сети перекрестные ассоциации, то есть предъявление на входы сети вектора **A** приведет к появлению на ее выходах вектора **B** и наоборот.

Когда нет необходимости, чтобы сеть в явном виде выдавала образец, то есть достаточно, скажем, получать номер образца, ассоциативную память успешно реализует *сеть Хэмминга*. Данная сеть характеризуется, по сравнению с сетью Хопфилда, меньшими затратами памяти и объемом вычислений, что становится очевидным из ее структуры (рис. 5.14).

Сеть состоит из двух слоев. Первый и второй слои имеют по $m$ нейронов, где $m$ — число образцов. Нейроны первого слоя имеют по $n$ синапсов, соединенных со входами сети (образующими фиктивный нулевой слой). Нейроны второго слоя связаны между собой ингибиторными (отрицательными обратными) синаптическими связями. Единственный синапс с положительной обратной связью для каждого нейрона соединен с его же аксоном.

Идея работы сети состоит в нахождении расстояния Хэмминга от тестируемого образа до всех образцов (расстоянием Хэмминга называется число различающихся битов в двух бинарных векторах).

Сеть должна выбрать образец с минимальным расстоянием Хэмминга до неизвестного входного сигнала, в результате чего будет активизирован только один выход сети, соответствующий этому образцу.

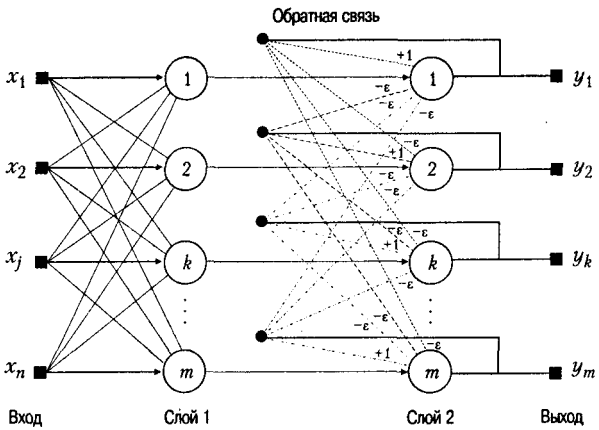


**Рис. 5.14.** Структурная схема сети Хэмминга

На стадии инициализации весовым коэффициентам первого слоя и порогу активационной функции присваиваются следующие значения:

$$w_{ik} = \frac{x_i^k}{2}, \quad \theta_k = n/2, \quad i = 1, 2, ..., n, \quad k = 1, 2, ..., m.$$

Здесь $x_i^k$ — $i$-й элемент $k$-го образца.

Весовые коэффициенты тормозящих синапсов во втором слое берут равными некоторой величине $0 < \varepsilon < 1/m$. Синапс нейрона, связанный с его же аксоном, имеет вес +1.

Алгоритм функционирования сети Хэмминга следующий:

1. На входы сети подается неизвестный вектор $\mathbf{X} = \{x_i: i = 1...n\}$, исходя из которого рассчитываются состояния нейронов первого слоя (верхний индекс в скобках указывает номер слоя):

   $$y_j^{(1)} = s_j^{(1)} = \sum_{i=1}^{n} w_{ij} x_i + \theta_j, j = 1, 2, ..., m.$$

   После этого полученными значениями инициализируются значения аксонов второго слоя:

   $$y_j^{(2)} = y_j^{(1)}, \ j = 1, 2, ..., m.$$

2. Вычислить новые состояния нейронов второго слоя:

$$s_j^{(2)}(t+1) = y_j - \varepsilon \sum_{k=1}^{m} y_k^{(2)}(t), \quad k \neq j, \quad j = 1, 2, \ldots, m$$

и значения их аксонов:

$$y_j^{(2)}(t+1) = f\left[s_j^{(2)}(t+1)\right], \quad j = 1, 2, \ldots, m.$$

3. Проверить, изменились ли выходы нейронов второго слоя за последнюю итерацию. Если да — перейти к шагу 2. Иначе — конец работы.

Из оценки алгоритма видно, что роль первого слоя весьма условна: воспользовавшись один раз на шаге 1 значениями его весовых коэффициентов, сеть больше не обращается к нему, поэтому первый слой может быть вообще исключен из сети (заменен на матрицу весовых коэффициентов).

В заключение можно сделать следующее обобщение. Сети Хопфилда и Хэмминга позволяют просто и эффективно разрешить задачу воссоздания образов по неполной и искаженной информации. Невысокая емкость сетей (число запоминаемых образов) объясняется тем, что сети не просто запоминают образы, а позволяют проводить их обобщение; например, с помощью сети Хэмминга возможна классификация по критерию максимального правдоподобия. Вместе с тем легкость построения программных и аппаратных моделей делают эти сети привлекательными для многих применений.

## Сеть с радиальными базисными элементами

В общем случае под термином Radial Basis Function Network (сеть с радиальными базисными элементами — RBF) понимается двухслойная сеть без обратных связей, которая содержит скрытый слой радиально симметричных скрытых нейронов (шаблонный слой). Для того чтобы шаблонный слой был радиально-симметричным, необходимо выполнение следующих условий:

- наличие центра, представленного в виде вектора во входном пространстве; обычно этот вектор сохраняется в пространстве весов от входного слоя к слою шаблонов;
- наличие способа измерения расстояния входного вектора от центра; обычно это стандартное евклидово расстояние;
- наличие специальной функции прохождения от одного аргумента, которая определяет выходной сигнал нейрона путем отображения функции расстояния; обычно используется функция Гаусса: $\varphi(s) = \exp(-s^2/2)$.

Другими словами, выходной сигнал шаблонного нейрона — это функция только от расстояния между входным вектором **X** и сохраненным центром **C**:

$$f(\mathbf{X}) = \varphi\left(\frac{\|\mathbf{X} - \mathbf{C}\|}{\sigma}\right).$$

Выходной слой сети является линейным, так что выходы сети определяются выражением

$$y_j = \sum_{i=1}^{K} w_{ij}\varphi\left(\frac{\|X - C_i\|}{\sigma_i}\right), \quad j = 1, 2, ..., m,$$

где $C_i$ — центры, $\sigma_i$ — отклонения радиальных элементов.

Обучение RBF-сети происходит в несколько этапов. Сначала определяются центры и отклонения для радиальных элементов, после этого оптимизируются параметры $w_{ij}$ линейного выходного слоя.



Расположение центров должно соответствовать кластерам, реально присутствующим в исходных данных. Рассмотрим два наиболее часто используемых метода.

- **Выборка из выборки**. В качестве центров радиальных элементов берутся несколько случайно выбранных точек обучающего множества. В силу случайности выбора они «представляют» распределение обучающих данных в статистическом смысле. Однако если число радиальных элементов невелико, такое представление может быть неудовлетворительным.
- **Алгоритм K-средних**. Этот алгоритм стремится выбрать оптимальное множество точек, являющихся центроидами кластеров в обучающих данных. При $K$ радиальных элементах их центры располагаются таким образом:
  - чтобы каждая обучающая точка «относилась» к одному центру кластера и лежала к нему ближе, чем к любому другому центру;
  - чтобы каждый центр кластера был центроидом множества обучающих точек, относящихся к этому кластеру.

После того как определено расположение центров, нужно найти отклонения. Величина отклонения (ее также называют сглаживающим фактором) определяет, насколько «острой» будет гауссова функция. Если эти функции выбраны слишком острыми, сеть не будет интерполировать данные между известными точками и потеряет способность к обобщению. Если же гауссовы функции взяты чересчур широкими, сеть не будет воспринимать мелкие детали (на самом деле

сказанное — еще одна форма проявления дилеммы пере/недообучения). Как правило отклонения выбираются таким образом, чтобы «колпак» каждой гауссовой функции захватывал несколько соседних центров. Для этого имеется несколько методов:

- **Явный**. Отклонения задаются пользователем.
- **Изотропный**. Отклонение берется одинаковым для всех элементов и определяется эвристически с учетом количества радиальных элементов и объема покрываемого пространства.
- ***K* ближайших соседей**. Отклонение каждого элемента устанавливается (индивидуально) равным среднему расстоянию до его *K* ближайших соседей. Тем самым отклонения будут меньше в тех частях пространства, где точки расположены густо, — здесь будут хорошо учитываться детали, — а там, где точек мало, отклонения будут большими (и будет производиться интерполяция).

После того как выбраны центры и отклонения, параметры выходного слоя оптимизируются с помощью стандартного метода линейной оптимизации — алгоритма псевдообратных матриц (сингулярного разложения).

Могут быть построены различные гибридные разновидности сетей с радиальными базисными функциями. Например, выходной слой может иметь нелинейные функции активации, и тогда для его обучения используется какой-либо из алгоритмов обучения многослойных сетей, например метод обратного распространения. Можно также обучать радиальный (скрытый) слой с помощью алгоритма обучения сети Кохонена — это еще один способ разместить центры так, чтобы они отражали расположение данных.

Сети RBF имеют ряд преимуществ перед рассмотренными многослойными сетями прямого распространения (хотя их структура и соответствует приведенной на рис. 5.5). Во-первых, они моделируют произвольную нелинейную функцию с помощью всего одного промежуточного слоя, тем самым избавляя нас от необходимости решать вопрос о числе слоев. Во-вторых, параметры линейной комбинации в выходном слое можно полностью оптимизировать с помощью хорошо известных методов линейной оптимизации, которые работают быстро и не испытывают трудностей с локальными минимумами, так мешающими при обучении с использованием алгоритма обратного распространения ошибки. Поэтому сеть RBF обучается очень быстро (на порядок быстрее, чем с использованием алгоритма обратного распространения).

Недостатки сетей RBF: данные сети обладают плохими экстраполирующими свойствами и получаются весьма громоздкими при большой размерности вектора входов.

## Вероятностная нейронная сеть

Задача оценки плотности вероятности по имеющимся данным имеет давнюю историю в математической статистике.

Обычно при этом предполагается, что плотность имеет некоторый определенный вид (чаще всего — что она имеет нормальное распределение). После этого оцениваются параметры модели. Нормальное распределение часто используется потому, что тогда параметры модели (среднее значение и стандартное отклонение) можно оценить аналитически.

Заметим, что предположение о нормальности далеко не всегда оправдано.



Другой подход к оценке плотности вероятности основан на так называемых *ядерных оценках*. Можно рассуждать так: тот факт, что наблюдаемое значение соответствует данной точке пространства, свидетельствует о том, что в этой точке имеется некоторая плотность вероятности. Кластеры из близко лежащих точек указывают на то, что в этом месте плотность вероятности значимо отличается от нуля. Вблизи наблюдаемых значений имеется большее доверие к уровню плотности, а по мере отдаления от них доверие убывает и стремится к нулю. В методе ядерных оценок в точке, соответствующей каждому наблюдению, помещается некоторая простая функция, затем все они складываются и в результате получается оценка для общей плотности вероятности. Чаще всего в качестве ядерных функций берутся гауссовы функции (в форме колокола). Если обучающих примеров достаточное количество, то такой метод дает достаточно хорошее приближение к истинной плотности вероятности.

Метод аппроксимации плотности вероятности с помощью ядерных функций во многом похож на метод радиальных базисных функций, и таким образом мы естественно приходим к понятиям *вероятностной нейронной сети* (PNN) и *обобщенно-регрессионной нейронной сети* (GRNN). PNN-сети предназначены для задач классификации, а GRNN-сети — для задач регрессии. Сети этих двух типов представляют собой реализацию методов ядерной аппроксимации, оформленных в виде нейронной сети.

Сеть PNN имеет по меньшей мере три слоя: входной, радиальный и выходной. Радиальные элементы берутся по одному на каждое обу-

чающее наблюдение. Каждый из них представляет гауссову функцию с центром в этом наблюдении. Каждому классу соответствует один выходной элемент. Каждый такой элемент соединен со всеми радиальными элементами, относящимися к его классу, а со всеми остальными радиальными элементами он имеет нулевое соединение. Таким образом, выходной элемент просто складывает отклики всех элементов, принадлежащих к его классу. Значения выходных сигналов получаются пропорциональными ядерным оценкам вероятности принадлежности соответствующим классам, и, учтя условие нормировки (сумма вероятностей полной группы событий равна единице), мы получаем окончательные оценки вероятности принадлежности классам.

Выход такой сети, соответствующий какому-либо классу, описывается выражением

$$y = \frac{1}{N\sigma^n}\sum_{k=1}^{N}\varphi\left(\frac{\|\mathbf{X}-\mathbf{X}^k\|}{\sigma}\right),$$

где $n$ — размерность входного вектора, $N$ — объем обучающей выборки, $\mathbf{X}^k$ — элемент (вектор) этой выборки, соответствующий отмеченному классу.

Базовая модель PNN-сети может иметь две модификации.

Вероятностная нейронная сеть имеет единственный управляющий параметр обучения, значение которого должно выбираться пользователем, — отклонение гауссовой функции σ (параметр сглаживания). Как и в случае RBF-сетей, этот параметр выбирается из тех соображений, чтобы «шапки» определенное число раз перекрывались: выбор слишком маленьких отклонений приведет к «острым» аппроксимирующим функциям и неспособности сети к обобщению, а при слишком больших отклонениях будут теряться детали. Требуемое значение несложно найти опытным путем, подбирая его так, чтобы контрольная ошибка была как можно меньше. К счастью, PNN-сети не очень чувствительны к выбору параметра сглаживания.

Наиболее важные преимущества PNN-сетей состоят в том, что выходное значение имеет вероятностный смысл (и поэтому его легче интерпретировать), и в том, что сеть быстро обучается. При обучении такой сети время тратится практически только на то, чтобы подавать ей на вход обучающие наблюдения, и сеть работает настолько быстро, насколько это вообще возможно.

Существенным недостатком таких сетей является их объем. PNN-сеть фактически вмещает в себя все обучающие данные, поэтому она требует много памяти и может медленно работать.

PNN-сети особенно полезны при пробных экспериментах (например, когда нужно решить, какие из входных переменных использовать), так как благодаря короткому времени обучения можно быстро проделать большое количество пробных тестов.

## Обобщенно-регрессионная нейронная сеть

Данная сеть устроена аналогично вероятностной нейронной сети, но она предназначена для решения задач регрессии, а не классификации. Как и в случае PNN-сети, в точку расположения каждого обучающего наблюдения помещается гауссова ядерная функция. Мы считаем, что каждое наблюдение свидетельствует о некоторой степени уверенности в том, что поверхность отклика в данной точке имеет определенную высоту, и эта уверенность убывает при отходе в сторону от точки. GRNN-сеть копирует внутрь себя все обучающие наблюдения и использует их для оценки отклика в произвольной точке. Окончательная выходная оценка сети получается как взвешенное среднее выходов по всем обучающим наблюдениям:

$$y = \frac{\sum_{k=1}^{N} y^k \varphi\left(\frac{\|X - X^k\|}{\sigma}\right)}{\sum_{k=1}^{N} \varphi\left(\frac{\|X - X^k\|}{\sigma}\right)},$$

где $X^k$, $y^k$ — точки обучающей выборки.

Первый промежуточный слой сети GRNN состоит из радиальных элементов. Второй промежуточный слой содержит элементы, которые помогают оценить взвешенное среднее. Каждый выход имеет в этом слое свой элемент, формирующий для него взвешенную сумму. Чтобы получить из взвешенной суммы взвешенное среднее, эту сумму нужно поделить на сумму весовых коэффициентов. Последнюю сумму вычисляет специальный элемент второго слоя. После этого в выходном слое производится собственно деление (с помощью специальных элементов «деления»). Таким образом, число элементов во втором промежуточном слое на единицу больше, чем в выходном слое. Как правило, в задачах регрессии требуется оценить одно выходное значение, и, соответственно, второй промежуточный слой содержит два элемента.

Можно модифицировать GRNN-сеть таким образом, чтобы радиальные элементы соответствовали не отдельным обучающим случаям,

а их кластерам. Это уменьшает размеры сети и увеличивает скорость обучения. Центры для таких элементов можно выбирать с помощью любого предназначенного для этой цели алгоритма (выборки из выборки, *K*-средних или Кохонена).

Достоинства и недостатки у сетей GRNN в основном такие же, как и у сетей PNN, — единственное различие состоит в том, что GRNN используются в задачах регрессии, а PNN — в задачах классификации. GRNN-сеть обучается почти мгновенно, но может получиться большой и медленной (хотя здесь, в отличие от PNN, не обязательно иметь по одному радиальному элементу на каждый обучающий пример, их число все равно будет большим). Как и сеть RBF, сеть GRNN не обладает способностью экстраполировать данные.

## Линейные НС

Согласно общепринятому в науке принципу, если более сложная модель не дает лучших результатов, чем более простая, то из них следует предпочесть вторую. В терминах аппроксимации отображений самой простой моделью будет линейная, в которой аппроксимирующая (подгоночная) функция определяется гиперплоскостью. В задаче классификации гиперплоскость размещается таким образом, чтобы она разделяла собой два класса (линейная дискриминантная функция); в задаче регрессии гиперплоскость должна проходить через заданные точки. Линейная модель обычно задается уравнением

$$\mathbf{Y} = \mathbf{XW} + \mathbf{B},$$

где **W** — матрица весов сети, **B** — вектор смещений.

На языке нейронных сетей линейная модель представляется сетью без промежуточных слоев, которая в выходном слое содержит только линейные элементы (то есть элементы с линейной функцией активации). Веса соответствуют элементам матрицы, а пороги — компонентам вектора смещения. Во время работы сеть фактически умножает вектор входов на матрицу весов, а затем к полученному вектору прибавляет вектор смещения.

# Функции пакета Neural Networks Toolbox

## Обзор функций пакета Neural Networks Toolbox

В состав пакета Neural Networks входят более 150 различных функций, образуя собой своеобразный макроязык программирования и позволяя пользователю создавать, обучать и использовать самые различные НС. С помощью команды

```
» help nnet
```

можно получить перечень входящих в пакет функций. Для получения справки по любой функции можно использовать команду

```
» help имя_функции
```

Данные функции по своему назначению делятся на ряд групп. Рассмотрим основные из них.

## Функции активации (передаточные функции) и связанные с ними функции

- `compet(X)` — функция конкуренции — в качестве аргумента использует матрицу `X`, столбцы которой ассоциируются с векторами входов. Возвращает разреженную матрицу с единичными элементами, индексы которых соответствуют индексам наибольших элементов каждого столбца.

  Пример:

```
» X = [0.9 -0.6;0.1 0.4;0.2 -0.5;0 0.5];
» compet(X)
ans =
   (1,1)        1
   (4,2)        1
```

  При записи в форме `compet(code)` возвращается некоторая служебная информация. Переменная `code` может принимать значения `'deriv'` (имя производной функции), `'name'` (полное имя), `'output'` (диапазон выхода), `'active'` (возможный диапазон входов).

  Примеры:

```
» compet('deriv')
ans =
     ''
» compet('name')
ans =
Competitive
» compet('output')
ans =
     0     1
» compet('active')
ans =
  -Inf   Inf
```

  Данная функция используется при создании НС со слоем «соревнующихся» нейронов (как, например, в сетях встречного распространения).

- `hardlim(X)` — пороговая функция активации с порогом $\theta = 0$; аргумент имеет тот же смысл, что и для предыдущей команды. Возвращает матрицу, размер которой равен размеру матрицы `X`, а элементы имеют значения 0 или 1 — в зависимости от знака соответствующего элемента `X`.

  Пример:

```
» X = [0.9 -0.6;0.1 0.4;0.2 -0.5;0 0.5];
» hardlim(X)

ans =
     1     0
     1     1
     1     0
     1     1
```

  В форме `hardlim(code)` функция возвращает информацию, аналогичную рассмотренной для функции `compet`.

- `hardlims(X)` — знаковая или сигнатурная функция активации (см. табл. 5.1); действует так же, как функция `hardlim(X)`, но возвращает значения −1 или +1.
- `logsig(X)` — сигмоидальная логистическая функция. Возвращает матрицу, элементы которой являются значениями логистической функции (см. табл. 5.1) от аргументов — элементов матрицы `X`.
- `poslin(X)` — возвращает матрицу значений полулинейной (см. табл. 5.1) функции.
- `purelin(X)` — возвращает матрицу значений линейной функции активации (см. табл. 5.1).
- `radbas(X)` — возвращает матрицу значений радиальной базисной функции (см. табл. 5.1).
- `satlin(X)` — возвращает матрицу значений полулинейной функции с насыщением (см. табл. 5.1).
- `satlins(X)` — возвращает матрицу значений линейной функции с насыщением (см. табл. 5.1).
- `softmax(X)` — возвращает матрицу, элементы которой вычисляются по формуле

$$\frac{\exp(x_{ij})}{\sum_{i=1}^{N}\exp(x_{ij})},$$

  где $N$ — число строк матрицы-аргумента `X`.

- `tansig(X)` — возвращает матрицу значений сигмоидальной (гиперболический тангенс) функции (см. табл. 5.1).
- `tribas(X)` — возвращает матрицу значений треугольной функции принадлежности (см. табл. 5.1).
- `dhardlim(X,Y)` — производная пороговой функции активации. Аргументами являются матрица входов `X` и матрица выходов `Y`; матрицы имеют одинаковый размер. Возвращается матрица того же размера с нулевыми элементами.
- `dhardlms(X,Y)` — производная знаковой функции активации (см. табл. 5.1). Возвращается матрица с нулевыми элементами.
- `dlogsig(X,Y)` — производная сигмоидальной логистической функции. Возвращается матрица с элементами $y_{ij}(1 - y_{ij})$.

  Примеры:

```
» X = [0.1; 0.8; -0.7];
» Y = logsig(X)
Y =
    0.5250
    0.6900
    0.3318
» dY_dX = dlogsig(X,Y)
dY_dX =
    0.2494
    0.2139
    0.2217
```

- `dposlin(X,Y)` — производная полулинейной функции. Возвращается матрица с элементами, равными единице для соответственных положительных элементов матрицы-аргумента `Y` и равными нулю в противоположном случае.
- `dpurelin(X,Y)` — производная линейной функции активации. Возвращается матрица с единичными элементами.
- `dradbas(X,Y)` — производная радиальной базисной функции (см. табл. 5.1). Возвращается матрица с элементами $-2x_{ij}y_{ij}$.
- `dsatlin(X,Y)` — возвращает матрицу значений производной полулинейной функции с насыщением (см. табл. 5.1). Элементы такой матрицы — единицы, если соответственные элементы матрицы `Y` принадлежат интервалу (0, 1), и нули в противоположном случае.
- `dsatlins(X,Y)` — возвращает матрицу значений производной линейной функции с насыщением (см. табл. 5.1). Элементы такой

матрицы — единицы, если соответственные элементы матрицы Y принадлежат интервалу (−1, 1), и нули в противоположном случае.

- `dtansig(X,Y)` — возвращает матрицу значений производной сигмоидальной функции — гиперболического тангенса (см. табл. 5.1). Элементы этой матрицы определяются выражением $1 - y_{ij}^2$.
- `dtribas(X,Y)` — возвращает матрицу значений производной треугольной функции активации (см. табл. 5.1). Элементы этой матрицы определяются следующими выражениями: 1, если $-1 < y_{ij} < 0$; $-1$, если $0 \le y_{ij} < 1$; 0 в оставшихся случаях.

## Функции обучения нейронных сетей

Эти функции позволяют устанавливать алгоритм и параметры обучения НС заданной конфигурации по желанию пользователя. В группу входят следующие функции.

- `[net,tr] = trainbfg(net,Pd,Tl,Ai,Q,TS,VV,TV)` — функция обучения, реализующая разновидность квазиньютоновского алгоритма обратного распространения ошибки (BFGS). Аргументы функции:
  - `net` — имя обучаемой НС;
  - `Pd` — наименование массива задержанных входов обучающей выборки;
  - `Tl` — массив целевых значений выходов;
  - `Ai` — матрица начальных условий входных задержек;
  - `Q` — количество обучающих пар в одном цикле обучения (размер «пачки»);
  - `TS` — вектор временных интервалов;
  - `VV` — пустой (`[ ]`) массив или массив проверочных данных;
  - `TV` — пустой (`[ ]`) массив или массив тестовых данных.

  Функция возвращает обученную нейронную сеть `net` и набор записей `tr` для каждого цикла обучения (`tr.epoch` — номер цикла, `tr.perf` — текущая ошибка обучения, `tr.vperf` — текущая ошибка для проверочной выборки, `tr.tperf` — текущая ошибка для тестовой выборки).

Процесс обучения происходит в соответствии со значениями следующих параметров (в скобках приведены значения по умолчанию):

- `net.trainParam.epochs` (100) — заданное количество циклов обучения;

- net.trainParam.show (25) — количество циклов для показа промежуточных результатов;
- net.trainParam.goal (0) — целевая ошибка обучения;
- net.trainParam.time (∞) — максимальное время обучения в секундах;
- net.trainParam.min_grad ($10^{-6}$) — целевое значение градиента;
- net.trainParam.max_fail (5) — максимально допустимая кратность превышения ошибки проверочной выборки по сравнению с достигнутым минимальным значением;
- net.trainParam.searchFcn ('srchcha') — имя используемого одномерного алгоритма оптимизации.

Структуры и размеры массивов:

- Pd — No × Ni × TS — массив ячеек, каждый элемент которого P{i,j,ts} есть матрица Dij×Q;
- Tl — Nl × TS — массив ячеек, каждый элемент которого P{i,ts} есть матрица Vi×Q;
- Ai — Nl × LD — массив ячеек, каждый элемент которого Ai{i,k} есть матрица Si×Q,

  где

  Ni = net.numInputs (количество входов сети);

  Nl = net.numLayers (количество ее слоев);

  LD = net.numLayerDelays (количество слоев задержки);

  Ri = net inputs{i}.size (размер *i*-го входа);

  Si = net.layers{i}.size (размер *i*-го слоя);;

  Vi = net targets{i}.size (размер целевого вектора);

  Dij = Ri * length(net.inputWeights{i,j}.delays) (вспомогательная вычисляемая величина).

Если массив VV не пустой, то он должен иметь структуру, определяемую следующими компонентами:

- VV.PD — задержанные значения входов проверочной выборки;
- VV.Tl — целевые значения;
- VV.Ai — начальные входные условия;
- VV.Q — количество проверочных пар в одном цикле обучения;
- VV.TS — временные интервалы проверочной выборки.

Эти параметры используются для задания останова процесса обучения в случаях, когда ошибка для проверочной выборки не уменьшается или начинает возрастать.

Структуру, аналогичную структуре массива VV, имеет массив TV.

Рассматриваемая функция, заданная в форме trainbfg(code), возвращает для значений аргумента 'pnames' и 'pdefaults', соответственно, имена параметров обучения и их значения по умолчанию.

Для использования функции в НС, определяемых пользователем, необходимо:

1. Установить параметр net.trainFcn = 'trainbfg' (при этом параметры алгоритма будут заданы по умолчанию).
2. Установить требуемые значения параметров (net.trainParam).

Процесс обучения останавливается в случае выполнения любого из следующих условий:

- превышено заданное количество циклов обучения (net.trainParam.epochs);
- превышено заданное время обучения (net.trainParam.time);
- ошибка обучения стала меньше заданной (net.trainParam.goal);
- градиент стал меньше заданного (net.trainParam.min_grad);
- возрастание ошибки проверочной выборки по сравнению с достигнутым минимальным превысило заданное значение (net.trainParam.max_fail).

Пример:

```
» P = [0 1 2 3 4 5];% Задание входного вектора
» T = [0 0 0 1 1 1];% Задание целевого вектора
» % Создание и тестирование сети
» net = newff([0 5],[2 1],{'tansig','logsig'},'traincgf');
» a = sim(net,P)
a =
    0.0586    0.0772    0.0822    0.0870    0.1326    0.5901
» % Обучение с новыми параметрами и повторное тестирование
» net.trainParam.searchFcn = 'srchcha';
» net.trainParam.epochs = 50;
» net.trainParam.show = 10;
» net.trainParam.goal = 0.1;
» net = train(net,P,T);
TRAINCGF-srchcha, Epoch 0/50, MSE 0.295008/0.1, Gradient 0.623241/1e-006
TRAINCGF-srchcha, Epoch 1/50, MSE 0.00824365/0.1, Gradient 0.0173555/1e-006
```

```
TRAINCGF, Performance goal met.
» a = sim(net,P)
a =
    0.0706    0.1024    0.1474    0.9009    0.9647    0.9655
```

В данном примере созданная многослойная НС, обученная с установками по умолчанию, вначале показала плохой результат отображения обучающей выборки, но после изменения параметров и повторного обучения сети результат стал вполне приемлемым.

- `[net,tr] = trainbr(net,Pd,Tl,Ai,Q,TS,VV)` — функция, реализующая так называемый байесовский метод обучения, сущность которого заключается в подстройке весов и смещений сети на основе алгоритма Левенберга—Марквардта. Данный алгоритм минимизирует комбинацию квадратов ошибок и весов с выбором наилучшего варианта (для получения наилучших обобщающих свойств сети). Эта процедура известна как байесовская регуляризация, откуда следует название метода.

  Аргументы, параметры, возвращаемые величины и использование — такие же, как у предыдущей функции. Сказанное остается в силе для всех остальных функций данной группы.

- `[net,tr] = traincgb(net,Pd,Tl,Ai,Q,TS,VV)` — функция обучения НС, реализующая разновидность алгоритма сопряженных градиентов (так называемый метод Powell—Beale).

- `[net,tr] = traincgf(net,Pd,Tl,Ai,Q,TS,VV)` — функция обучения НС, реализующая разновидность алгоритма обратного распространения ошибки в сочетании с методом оптимизации Флетчера—Поуэлла.

- `[net,tr] = traincgp(net,Pd,Tl,Ai,Q,TS,VV)` — то же, что в предыдущем случае, но с использованием метода Polak—Ribiere.

- `[net,tr] = traingd(net,Pd,Tl,Ai,Q,TS,VV)` — функция, реализующая «классический» алгоритм обратного распространения ошибки.

- `[net,tr] = traingda(net,Pd,Tl,Ai,Q,TS,VV)` — то же, что в предыдущем случае, но с адаптацией коэффициента скорости обучения.

- `[net,tr] = traingdm(net,Pd,Tl,Ai,Q,TS,VV)` — функция, реализующая модифицированный алгоритм обратного распространения ошибки с введенной «инерционностью» коррекции весов и смещений.

- `[net,tr] = traingdx(net,Pd,Tl,Ai,Q,TS,VV)` — функция, реализующая комбинированный алгоритм обучения, объединяющий особенности двух вышеприведенных.

- `[net,tr] = trainlm(net,Pd,Tl,Ai,Q,TS,VV)` — данная функция возвращает веса и смещения НС, используя алгоритм оптимизации Левенберга—Марквардта.
- `[net,tr] = trainoss(net,Pd,Tl,Ai,Q,TS,VV)` — функция, реализующая разновидность алгоритма обратного распространения ошибки с использованием метода секущих.
- `[net,tr] = trainrp(net,Pd,Tl,Ai,Q,TS,VV)` — функция, реализующая разновидность алгоритма обратного распространения ошибки, так называемый *упругий* алгоритм обратного распространения (resilient backpropagation algorithm, RPROP).
- `[net,tr] = trainscg(net,Pd,Tl,Ai,Q,TS,VV)` — данная функция возвращает веса и смещения НС, используя алгоритм масштабируемых сопряженных градиентов.
- `[net,tr] = trainwb(net,Pd,Tl,Ai,Q,TS,VV)` — данная функция корректирует веса и смещения сети в соответствии с заданной функцией обучения нейронов.
- `[net,tr] = trainwb1(net,Pd,Tl,Ai,Q,TS,VV)` — то же, что и предыдущая функция, но одновременно на вход сети предъявляется только один вектор входа.
- `[net,Ac,El] = adaptwb(net,Pd,Tl,Ai,Q,TS)` — функция адаптации весов и смещений НС. Используется совместно с функциями `newp` и `newlin` (см. ниже). Возвращает массив выходов слоев `Ac` и массив ошибок слоев `El`.

## Функции настройки слоев нейронов

Функции данной группы являются вспомогательными при работе с некоторыми рассмотренными функциями обучения НС (например, `trainwb`, `trainwb1`, `adaptwb`), а также используются при настройках однослойных нейросетевых структур (персептронов, слоев Кохонена и т. п.):

- `[dB,LS] = learncon(B,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функция настройки весов с введением «чувства справедливости» (см. выше). Аргументы:
  - `B` — S × 1 — вектор смещений;
  - `P` — 1 × Q — входной вектор;
  - `Z` — S × Q — матрица взвешенных входов;
  - `N` — S × Q — матрица входов;
  - `A` — S × Q — матрица выходных векторов;
  - `T` — S × Q — матрица целевых векторов слоя;

- E — S × Q — матрица ошибок;
- gW — S × R — градиент критерия эффективности по отношению к вектору весов;
- gA — S × Q — градиент критерия эффективности по отношению к вектору выхода;
- D — S × S — матрица расстояний между нейронами;
- LP — параметр обучения, LP = [ ];
- LS — состояние обучения, в начале — [ ].

Возвращаемые величины:

- dB — S × 1 — вектор изменений весов (или смещений);
- LS — новое состояние обучения.

Функция в форме learncon(code) возвращает следующую информацию:

- при аргументе 'pnames' — имена параметров обучения;
- при 'pdefaults' — значения параметров по умолчанию;
- при 'needg' — 1, если эта функция использует gW или gA.

Алгоритм выполнения функции сначала вычисляет «чувство справедливости» нейрона по выражению c = (1 - lr)*c + lr*a, а уже затем корректирует вес в соответствии с формулой b = exp(1 - - log(c)) - b.

Пример:

```
» a = rand(3,1);
» b = rand(3,1);
» lp.lr = 0.5; % Задание параметра обучения
» dW = learncon(b,[],[],[],a,[],[],[],[],[],lp,[])
dW =
    0.3449
    0.7657
    0.5405
```

- learngd — функция коррекции весов и смещений, реализующая градиентный алгоритм оптимизации.

Запись:

```
[dW,LS] = learngd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
[db,LS] = learngd(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
info = learngd(code)
```

Описание. Аргументы функции: W — матрица весов или вектор смещения, остальные аргументы — как у предыдущей функции.

Возвращаемые параметры — как у предыдущей функции.

- `learngdm` — функция практически аналогична предыдущей, но используемый алгоритм оптимизации — градиентный метод с инерционной составляющей.
- `[dW,LS] = learnh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функция коррекции весов, использующая правило Хебба, в соответствии с которым веса корректируются по выражению `dw = lr*a*p'`.
- `[dW,LS] = learnhd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функция реализует модификацию правила Хебба, при котором корректировка весов осуществляется по соотношению `dw = lr*a*p' - dr*w`.
- `[dW,LS] = learnis(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функция подстройки весов «входной звезды» (нейрона слоя Гроссберга), реализующая выражение `dw = lr*a*(p'-w)`.
- `[dW,LS] = learnk(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функция подстройки весов слоя Кохонена, реализующая выражение `dw = lr*(p' - w)`, если $a \neq 0$, и 0 в противном случае.
- `[dW,LS] = learnlv*(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функции настройки сетей встречного распространения (вместо «*» может быть «1» или «2»).
- `[dW,LS] = learnos(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функция настройки нейрона типа «выходная звезда», реализующая выражение `dw = lr*(a - w)*p'`.
- `[dW,LS] = learnp(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функция, реализующая алгоритм обучения персептрона.
- `[dW,LS] = learnpn(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — то же, что и предыдущая функция, но с нормализацией входов. Более эффективна при больших изменениях входных сигналов.
- `[dW,LS] = learnsom(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функция обучения самоорганизующихся карт.
- `[dW,LS] = learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` — функция обучения, реализующая так называемый алгоритм Уидроу—Хоффа (Widrow—Hoff), основанный на соотношении `dw = lr*e*pn'` и известный также как дельта-правило или правило наименьших квадратов.

### Функции одномерной оптимизации

Функции данной группы можно рассматривать как вспомогательные для функций обучения нейронных сетей. Они реализуют различные алгоритмы одномерного поиска.

- srchbac — функция реализует так называемый алгоритм перебора с возвратами (backtracking).
- srchbre — функция реализует комбинированный метод оптимизации, объединяющий метод золотого сечения и квадратичную интерполяцию.
- srchcha — функция реализует разновидность метода оптимизации с применением кубической интерполяции.
- srchgol — функция реализует метод золотого сечения.
- srchhyb — функция реализует комбинированный метод оптимизации, объединяющий метод дихотомии и кубическую интерполяцию.

## Функции инициализации слоев и смещений

Для многих нейронных сетей этапом, предваряющим процедуру их обучения, является этап инициализации (задания некоторых, обычно выбираемых случайным образом) весов и смещений сети. Такая инициализация выполняется с помощью функций данной группы.



- initcon(s,pr) — функция, устанавливающая смещения нейронов в зависимости от среднего выхода нейрона. Аргументы: s — количество нейронов, pr = [Pmin Pmax] — матрица (с двумя столбцами) минимальных и максимальных значений входов, по умолчанию [1 1]. Возвращает вектор смещений. Используется совместно с командой learncon.

  Пример:

```
» b = initcon(3)
b =
    8.1548
    8.1548
    8.1548
```

- initzero — функция задания нулевых начальных значений весам или смещениям. Аргументы те же, что и у предыдущей команды.
- midpoint(S,PR) — функция инициализации, устанавливающая веса в соответствии со средними значениями входов.
- randnc(S,R) — функция задания матрицы весов. Возвращает матрицу размером $S \times R$ со случайными элементами, нормализованную по столбцам (векторы-столбцы имеют единичную длину).
- randnr(S,R) — то же, что предыдущая функция, но возвращает матрицу весов, нормализованную по строкам.

- `rands` — функция инициализации весов/смещений заданием их случайных значений из диапазона [−1, 1]. Запись:

```
W = rands(S,PR)
M = rands(S,R)
v = rands(S)
```

Описание. Аргументы — те же, что и для функции `initcon`; значение `R` по умолчанию — 1. Возвращается матрица соответствующего размера.

## Функции создания нейронных сетей

- `network` — функция создания нейронной сети пользователя. Запись:

```
net = network
net =
network(numInputs,numLayers,biasConnect,inputConnect,layerConnect,
outputConnect,targetConnect)
```

Описание. Функция возвращает созданную нейронную сеть с именем `net` и со следующими характеристиками (в скобках даны значения по умолчанию):

  - `numInputs` — количество входов (0);
  - `numLayers` — количество слоев (0);
  - `biasConnect` — булевский вектор с числом элементов, равным количеству слоев (нули);
  - `inputConnect` — булевская матрица с числом строк, равным количеству слоев, и числом столбцов, равным количеству входов (нули);
  - `layerConnect` — булевская матрица с числом строк и столбцов, равным количеству слоев (нули);
  - `outputConnect` — булевский вектор-строка с числом элементов, равным количеству слоев (нули);
  - `targetConnect` — вектор-строка такая же, как предыдущая (нули).

- `net = newc(PR,S,KLR,CLR)` — функция создания слоя Кохонена. Функция использует следующие аргументы:
  - `PR` — R × 2 — матрица минимальных и максимальных значений для R входных элементов;
  - `S` — число нейронов;
  - `KLR` — коэффициент обучения Кохонена (по умолчанию 0,01);
  - `CLR` — коэффициент «справедливости» (по умолчанию 0,001).

Функция возвращает слой Кохонена с заданным именем.

- `net = newcf(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)` — функция создания разновидности многослойной НС с обратным распространением ошибки — так называемой каскадной НС. Такая сеть содержит `Nl` скрытых слоев, использует входные функции типа `dotprod` и `netsum`, инициализация сети осуществляется функцией `initnw`. Аргументы функции:
  - `PR` — R × 2 — матрица минимальных и максимальных значений R входных элементов;
  - `Si` — размер *i*-го скрытого слоя, для `Nl` слоев;
  - `TFi` — функция активации нейронов *i*-го слоя, по умолчанию `'tansig'`;
  - `BTF` — функция обучения сети, по умолчанию `'traingd'`;
  - `BLF` — функция настройки весов и смещений, по умолчанию `'learngdm'`;
  - `PF` — функция ошибки, по умолчанию `'mse'`.

Пример:

```
» P = [0 1 2 3 4 5 6 7 8 9 10];
» T = [0 1 2 3 4 3 2 1 2 3 4];
» net = newcf([0 10],[5 1],{'tansig' 'purelin'}); % Создание новой
сети
» net.trainParam.epochs = 50; % Задание количества циклов обучения
» net = train(net,P,T); % Обучение НС
TRAINLM, Epoch 0/50, MSE 7.77493/0, Gradient 138.282/1e-010
TRAINLM, Epoch 25/50, MSE 4.01014e-010/0, Gradient 0.00028557/1e-010
TRAINLM, Epoch 50/50, MSE 1.13636e-011/0, Gradient 1.76513e-006/1e-010
TRAINLM, Maximum epoch reached, performance goal was not met.
» Y = sim(net,P); % Использование НС
» plot(P,T,P,Y,'o') % Графическая иллюстрация работы сети
```

На рис. 5.15 точками отображены элементы обучающей выборки, линией — выход сети.

- `net = newelm(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)` — функция создания сети Элмана. Аргументы — такие же, как и у предыдущей функции.
- `net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)` — функция создания «классической» многослойной НС с обучением по методу обратного распространения ошибки.

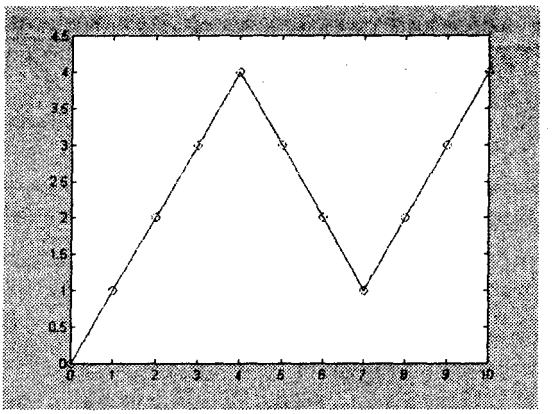


**Рис. 5.15.** Иллюстрация работы сети

- `net = newfftd(PR,ID,[S1 S2…SNl],{TF1 TF2…TFNl},BTF,BLF,PF)` — то же, что и предыдущая функция, но с наличием задержек по входам. Дополнительный аргумент `ID` — вектор входных задержек.
- `net = newgrnn(P,T,spread)` — функция создания обобщенно-регрессионной сети. Аргументы:
  - `P` — R × Q — матрица Q входных векторов;
  - `T` — S × Q — матрица Q целевых векторов;
  - `spread` — отклонение (по умолчанию 1,0).
- `net = newhop(T)` — функция создания сети Хопфилда. Использует только один аргумент:
  - `T` — R × Q — матрица Q целевых векторов (значения элементов должны быть +1 или −1).
- `net = newlin(PR,S,ID,LR)` — функция создания слоя линейных нейронов. Аргументы:
  - `PR` — R × 2 — матрица минимальных и максимальных значений для R входных элементов;
  - `S` — число элементов в выходном векторе;
  - `ID` — вектор входной задержки (по умолчанию `[0]`);
  - `LR` — коэффициент обучения (по умолчанию 0.01).

  Возвращается новый линейный слой. При записи в форме `net = newlin(PR,S,0,P)` используется аргумент `P` — матрица входных век-

торов, при этом возвращается линейный слой с максимально возможным коэффициентом обучения при заданной матрице P.

- net = newlind(P,T) — функция проектирования нового линейного слоя. Данная функция по матрицам входных и выходных векторов методом наименьших квадратов определяет веса и смещения линейной НС.
- net = newlvq(PR,S1,PC,LR,LF) — функция создания сети встречного распространения. Аргументы:
  - PR — R × 2 — матрица минимальных и максимальных значений R входных элементов;
  - S1 — число скрытых нейронов;
  - PC — S2 элементов вектора, задающих доли принадлежности к различным классам;
  - LR — коэффициент обучения, по умолчанию 0,01;
  - LF — функция обучения, по умолчанию 'learnlv2'.
- net = newp(PR,S,TF,LF) — функция создания персептрона. Аргументы:
  - PR — R × 2 — матрица минимальных и максимальных значений R входных элементов,
  - S — число нейронов;
  - TF — функция активации, по умолчанию 'hardlim';
  - LF — функция обучения, по умолчанию 'learnp'.
- net = newpnn(P,T,spread) — функция создания вероятностной НС. Аргументы — как у функции newgrnn.
- net = newrb(P,T,goal,spread) — функция создания сети с радиальными базисными элементами. Аргументы P, T, spread — такие же, как у функции newgrnn; аргумент goal — заданная среднеквадратичная ошибка.
- net = newrbe(P,T,spread) — функция создания сети с радиальными базисными элементами с нулевой ошибкой на обучающей выборке.
- net = newsom (PR,[D1,D2,...],TFCN,DFCN,OLR,OSTEPS,TLR,TND) — функция создания самообучающейся карты со следующими аргументами:
  - PR — R × 2 — матрица минимальных и максимальных значений R входных элементов;
  - I — размеры *i*-го слоя, по умолчанию [5 8];
  - TFCN — топологическая функция, по умолчанию 'hextop';
  - DFCN — функция расстояния, по умолчанию 'linkdist';

- OLR — коэффициент обучения фазы упорядочивания, по умолчанию 0,9;
- OSTEPS — число шагов фазы упорядочивания, по умолчанию 1000;
- TLR — коэффициент обучения фазы настройки, по умолчанию 0,02;
- TND — расстояние для фазы настройки, по умолчанию 1.

## Функции преобразования входов сети

Функции данной группы преобразуют значения входов с использованием операций умножения или суммирования.

- `netprod(Z1,Z2,...)` — возвращает матрицу, элементы которой определяются как произведения элементов входных векторов и смещений. Аргументы `Z1, Z2,...` — матрицы, чьи столбцы ассоциированы с входами или смещениями.

  Примеры:

```
» z1 = [1 2 4;3 4 1];
» z2 = [-1 2 2; -5 -6 1];
» n = netprod(z1,z2)
n =
    -1     4     8
   -15   -24     1
» b = [0; -1];
» % Функция concur(b,3) создает 3 копии вектора смещения
» n = netprod(z1,z2,concur(b,3))
n =
     0     0     0
    15    24    -1
```

- `netsum(Z1,Z2,...)` — то же, что в предыдущем случае, но вместо умножения используется суммирование.
- `dnetprod(Z,N)` — возвращает матрицу значений первой производной входов, преобразованных функцией `N = netprod(Z1,Z2,...)`.

  Пример:

```
» Z1 = [0; 1; -1];
» Z2 = [1; 0.5; 1.2];
» N = netprod(Z1,Z2)
N =
         0
    0.5000
   -1.2000
```

```
» dN_dZ2 = dnetprod(Z2,N)
dN_dZ2 =
     0
     1
    -1
```

- `dnetsum(Z,N)` — то же, что и в предыдущем случае, но по отношению к функции `netsum(Z1,Z2,...)`.

## Функции весов и расстояний

- `boxdist(pos)` — функция определения box-расстояния между нейронами в слое. Имеет один аргумент `pos` — матрицу размером $N \times S$, элементы которой определяют координаты нейронов, возвращает матрицу расстояний размером $S \times S$. Расстояния (элементы возвращаемой матрицы) вычисляются по выражению $D_{ij} = \max(\text{abs}(\mathbf{P}_i - \mathbf{P}_j))$, где $\mathbf{P}_i$ и $\mathbf{P}_j$ — векторы, содержащие координаты нейронов $i$ и $j$.

  Пример:

```
% Случайное размещение 4-х нейронов в 3-х мерном пространстве
» pos = rand(3,4) % (генерация случайной матрицы 3x4)
pos =
    0.8600    0.4966    0.6449    0.3420
    0.8537    0.8998    0.8180    0.2897
    0.5936    0.8216    0.6602    0.3412
» d = boxdist(pos)
d =
         0    0.3635    0.2151    0.5639
    0.3635         0    0.1614    0.6100
    0.2151    0.1614         0    0.5282
    0.5639    0.6100    0.5282         0
```

`dist` — функция вычисления евклидова расстояния. Запись:

- `Z = dist(W,P)` — возвращает матрицу, элементы которой являются евклидовыми расстояниями между строками (векторами) матрицы `W` и столбцами матрицы `P` (матрицы должны иметь соответствующие размеры).
- `D = dist(pos)` — в такой форме функция аналогична функции `boxdist(pos)` за тем исключением, что возвращается матрица евклидовых расстояний.
- `negdist(W,P)` — функция идентична предыдущей, но элементы возвращаемой матрицы являются евклидовыми расстояниями, взятыми со знаком «минус».

- `mandist(W,P)` — функция аналогична предыдущей, но элементы возвращаемой матрицы являются расстояниями по Манхэттену, которое для векторов `x` и `y` определяется соотношением `D = sum(abs(x-y))`.
- `linkdist(pos)` — функция определения линейного расстояния между нейронами в слое. Аналогична функции `boxdist`, отличаясь *от последней алгоритмом определения расстояния*:
  - $D_{ij} = 0$, если $i = j$;
  - $D_{ij} = 1$, если евклидово расстояние между $\mathbf{P}_i$ и $\mathbf{P}_j$ меньше или равно 1;
  - $D_{ij} = 2$, если существует $k$ такое, что $D_{ik} = D_{kj} = 1$;
  - $D_{ij} = 3$, если существуют $k_1$ и $k_2$ такие, что $D_{ik_1} = D_{k_1k_2} = D_{k_2j} = 1$;
  - $D_{ij} = N$, если существуют $k_1$, $k_2$, ..., $k_N$ такие, что $D_{ik_1} = D_{k_1k_2} = ... = D_{k_Nj} = 1$;
  - $D_{ij} = S$, если не выполнено ни одно из предыдущих условий.
- `dotprod(W,P)` — функция придания входам `P` некоторых весов `W`. Возвращает матрицу `Z = W*P`.
- `normprod(W,P)` — функция аналогична предыдущей, но каждый элемент возвращаемой матрицы дополнительно делится на сумму элементов соответствующего столбца-сомножителя матрицы `P`.

## Функции размещения нейронов (топологические функции)

Функции данной группы используются при создании самоорганизующихся карт.

- `gridtop(dim1,dim2,...,dimN)` — функция размещения `N` слоев нейронов *в узлах регулярной прямоугольной N-мерной решетки*. `dim1`, `dim2`, ..., `dimN` — число нейронов в слоях. Возвращает матрицу, содержащую $N$ строк и (`dim1` × `dim2` × ··· × `dimN`) столбцов с координатами нейронов.

  Пример:

```
» pos = gridtop(2,3)
pos =

     0     1     0     1     0     1
     0     0     1     1     2     2
```

- `hextop(dim1,dim2,...,dimN)` — функция аналогична предыдущей, но размещение нейронов производится в узлах гексагональной (шестиугольной) решетки.

Пример (рис. 5.16):

```
» pos = hextop(8,5); plotsom(pos)
```

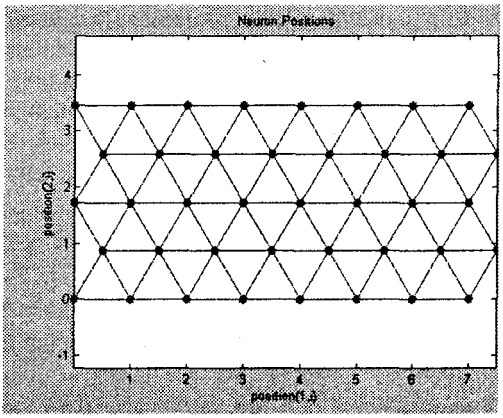


**Рис. 5.16.** Результат выполнения команды hextop(8,5)



- `randtop(dim1,dim2,…,dimN)` — аналогична функции `gridtop(dim1, dim2,…,dimN)`, но координаты нейронов выбираются случайным образом.

Пример (рис. 5.17):

```
» pos = randtop(16,12); plotsom(pos)
```

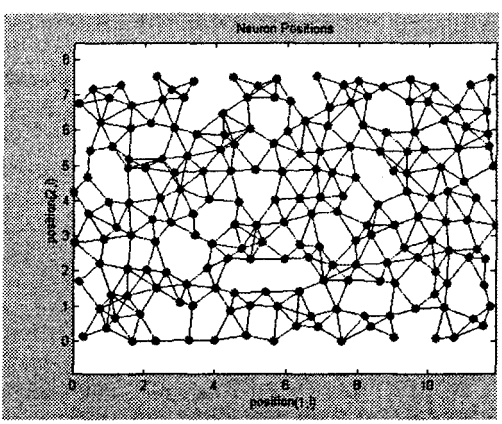


**Рис. 5.17.** Результат выполнения команды randtop(16,12)

## Функции использования нейронных сетей

- `[Y,Pf,Af] = sim(net,P,Pi,Ai)` — функция, моделирующая работу нейронной сети. Аргументы: `net` — имя сети, `P` — ее входы, `Pi` — массив начальных условий входных задержек (по умолчанию они нулевые), `Ai` — массив начальных условий задержек слоя нейронов (по умолчанию они нулевые). Функция возвращает значения выходов `Y` и массивы конечных условий задержек.

  Аргументы `Pi`, `Ai`, `Pf`, `Af` используются только в случаях, когда сеть имеет задержки по входам или по слоям нейронов. Структура данных аргументов:

  - `P` — массив размером Ni × TS, каждый элемент которого `P{i,ts}` является матрицей размером Ri × Q;
  - `Pi` — массив размером Ni × ID, каждый элемент которого `Pi{i,k}` (*i*-й вход в момент `ts=k-ID`) является матрицей размером Ri × Q (по умолчанию — ноль);
  - `Ai` — массив размером Nl × LD, каждый элемент которого `Ai{i,k}` (выход *i*-го слоя в момент `ts=k-LD`) является матрицей размером Si × Q (по умолчанию — ноль);
  - `Y` — массив размером NO × TS, каждый элемент которого `Y{i,ts}` является матрицей размером Ui × Q;
  - `Pf` — массив размером Ni × ID, каждый элемент которого `Pf{i,k}` (*i*-й вход в момент `ts =TS+k-ID`) является матрицей размером Ri × Q;
  - `Af` — массив размером Nl × LD, каждый элемент которого `Af{i,k}` (выход *i*-го слоя в момент `ts=TS+k-LD`) является матрицей размером Si × Q.

  При этом:

  - `Ni = net.numInputs` — количество входов сети;
  - `Nl = net.numLayers` — количество ее слоев;
  - `No = net.numOutputs` — количество выходов сети;
  - `ID = net.numInputDelays` — входные задержки;
  - `LD = net.numLayerDelays` — задержки слоя;
  - `TS` — число временных интервалов;
  - `Q` — размер набора подаваемых векторов;
  - `Ri = net.inputs{ i }.size` — размер *i*-го вектора входа;
  - `Si = net.layers{ i }.size` — размер *i*-го слоя;
  - `Ui = net.outputs{ i }.size` — размер *i*-го вектора выхода.

- net = init(net) — функция инициализирует нейронную сеть с именем net, устанавливая веса и смещения сети в соответствии с установками net.initFcn и net.initParam.
- [net,Y,E,Pf,Af] = adapt(net,P,T,Pi,Ai) — функция адаптации НС. Выполняет адаптацию сети в соответствии с установками net.adaptFcn и net.adaptParam. Здесь E — ошибки сети, T — целевые значения выходов (по умолчанию — ноль); остальные аргументы — как у команды sim.
- [net,tr] = train(net,P,T,Pi,Ai) — функция осуществляет обучение НС в соответствии с установками net.trainFcn и net.trainParam. Здесь tr — информация о выполнении процесса обучения (количество циклов и соответствующая ошибка обучения).
- disp(net) — функция возвращает развернутую информацию о структуре и свойствах НС.

Пример:

```
» net = newp([-1 1; 0 2],3); % Создание НС типа персептрон
» disp(net)
    Neural Network object:
    architecture:
         numInputs: 1
         numLayers: 1
       biasConnect: [1]
      inputConnect: [1]
      layerConnect: [0]
     outputConnect: [1]
     targetConnect: [1]

        numOutputs: 1  (read-only)
        numTargets: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

            inputs: {1x1 cell} of inputs
            layers: {1x1 cell} of layers
           outputs: {1x1 cell} containing 1 output
           targets: {1x1 cell} containing 1 target
            biases: {1x1 cell} containing 1 bias
      inputWeights: {1x1 cell} containing 1 input weight
      layerWeights: {1x1 cell} containing no layer weights
```

```
functions:

      adaptFcn: 'adaptwb'
       initFcn: 'initlay'
    performFcn: 'mae'
     trainFcn: 'trainwb'

parameters:
    adaptParam: .passes
     initParam: (none)
  performParam: (none)
    trainParam: .epochs, .goal, .max_fail, .show, .time

weight and bias values:

  IW: {1x1 cell} containing 1 input weight matrix
  LW: {1x1 cell} containing no layer weight matrices
   b: {1x1 cell} containing 1 bias vector

other:
      userdata: (user stuff)
```

- `display(net)` — то же, что предыдущая команда, но дополнительно возвращает имя нейронной сети.

## Графические функции

- `hintonw(W,maxw,minw)` — функция возвращает так называемый хинтоновский график матрицы весов, при котором каждый весовой коэффициент отображается квадратом площадью, пропорциональной величине данного коэффициента. Знак отображается цветом квадрата. Аргументы: `W` — матрица весов, `maxw` и `minw` — минимальное и максимальное значения ее коэффициентов (могут не задаваться).

  Пример (рис. 5.18):

```
» W = rands(2,3)
W =
   -0.8842    0.6263   -0.7222
   -0.2943   -0.9803   -0.5945
» hintonw(W)
```

- `hintonwbw(W,b,maxw,minw)` — то же, что и предыдущая функция, но на графике отображаются не только веса, но и смещения.

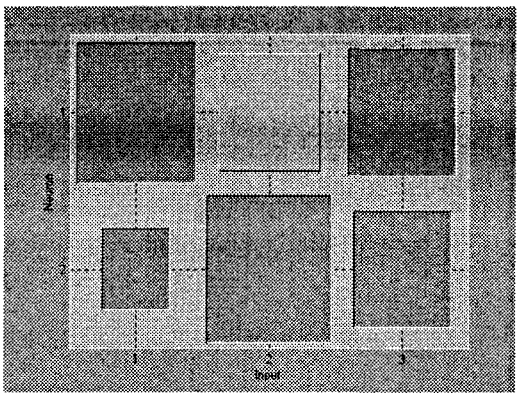

**Рис. 5.18.** Иллюстрация к выполнению функции hintonw



- `plotbr(tr,name,epoch)` — функция возвращает графики изменения критерия качества НС в процессе обучения при использовании байесовского метода (см. выше). Аргументы: `tr` — запись процесса обучения, `name` — имя НС, `epoch` — количество циклов обучения (по умолчанию — длина записи обучения).

  Пример (рис. 5.19):

```
» p = [-1:.05:1];
» t = sin(2*pi*p)+0.1*randn(size(p));
» % Создание новой сети
» net=newff([-1 1],[20,1],{'tansig','purelin'},'trainbr');
» [net,tr] = train(net,p,t); % Обучение сети
TRAINBR, Epoch 0/100, SSE 228.933/0, SSW 21461.7, Grad
2.33e+002/1.00e-010, #Par 6.10e+001/61
TRAINBR, Epoch 25/100, SSE 0.235423/0, SSW 211.044, Grad 9.43e-
002/1.00e-010, #Par 1.35e+001/61
TRAINBR, Epoch 50/100, SSE 0.240881/0, SSW 121.647, Grad 1.87e-
001/1.00e-010, #Par 1.23e+001/61
TRAINBR, Epoch 75/100, SSE 0.239867/0, SSW 116.884, Grad 1.62e-
002/1.00e-010, #Par 1.22e+001/61
TRAINBR, Epoch 100/100, SSE 0.239762/0, SSW 116.871, Grad 9.60e-
003/1.00e-010, #Par 1.22e+001/61
TRAINBR, Maximum epoch reached.
» plotbr(tr)
```

- `plotep(w,b,e)` — функция отображает позиции весов и смещений на поверхности ошибки НС. Аргументы: `w`, `b`, `e` — соответственно,

матрицы весов, смещений и ошибок. Возвращается вектор, используемый для продолжения графика, созданного функцией plotes (см. ниже).

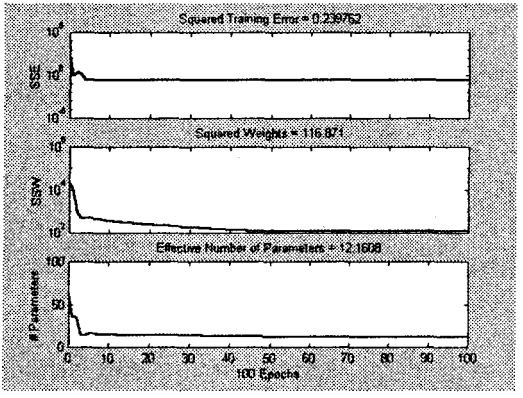


**Рис. 5.19.** Результат выполнения функции plotbr(tr)

- `plotes(wv,bv,e,v)` — функция возвращает график поверхности ошибки одновходового нейрона. Аргументы: `wv`, `bv` — соответственно, наборы значений веса и смещения нейрона, `e` — матрица значений ошибки, `v` — опция вида изображения (по умолчанию `[-37.5, 30]`).
- `plotpc(W,b)` — функция возвращает график линии решения для персептрона. Аргументы: `W` — матрица весов, `b` — вектор смещений. Используется совместно с функцией `plotpv` (см. ниже).
- `plotperf(tr,goal,name,epoch)` — возвращает график изменения критерия качества НС в процессе обучения. Аргументы: `tr` — запись процесса обучения, `goal` — целевое значение критерия, `name` — имя НС, `epoch` — количество циклов обучения.
- `plotpv(p,t)` — функция возвращает графическое отображение входных `p` и целевых `t` векторов персептрона.

  Пример (рис. 5.20):

```
» p = [0 0 1 1; 0 1 0 1];
» t = [0 0 0 1];
» plotpv(p,t)
```

- `plotsom(pos)` — функция возвращает графическое представление расположения нейронов в самоорганизующихся картах (см. раздел «Функции размещения нейронов», рис. 5.16 и 5.17).

- `plotv(M,t)` — функция графического изображения векторов. Аргументы: `M` — матрица с двумя строками, столбцы которой ассоциированы с векторами, `t` — опция, задающая тип линии.

  Пример (рис. 5.21):

  ```
  » plotv([-.4 0.7 .2; -0.5 .1 0.5],'-')
  ```

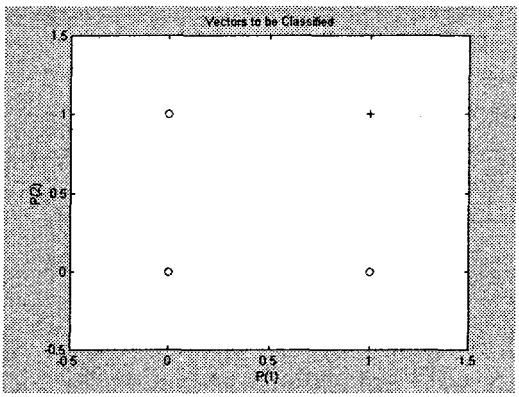


**Рис. 5.20.** Результат использования функции plotpv(p,t)

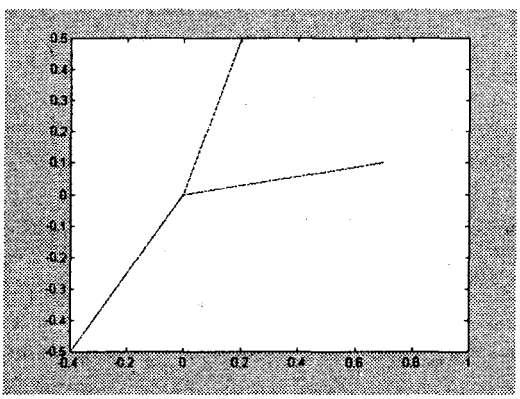

**Рис. 5.21.** Результат использования функции plotv(M,t)

- `plotvec(M,C,m)` — функция графического изображения векторов разными цветами. Аргументы: `M` — матрица с двумя строками, столбцы которой ассоциированы с векторами, `C` — строка задания цветов, `m` — тип точек, указывающих концы векторов (по умолчанию '+').

## Прочие функции

- `errsurf(p,t,wv,bv,f)` — функция, возвращающая матрицу значений поверхности ошибок нейрона с одним входом и одним выходом в зависимости от значений веса и смещения. Аргументы:
  - `p` — вектор значений входа;
  - `t` — вектор значений выхода;
  - `wv` — набор значений веса нейрона;
  - `bv` — набор значений смещения;
  - `f` — название реализуемой функции активации (строка).

  Размер возвращаемой матрицы — (количество значений `bv`) × (количество значений `wv`). Пример (рис. 5.22):

```
» p = [-6.0 -6.1 -4.1 -4.0 +4.0 +4.1 +6.0 +6.1];
» t = [+0.0 +0.0 +.97 +.99 +.01 +.03 +1.0 +1.0];
» wv = -1:.1:1; bv = -2.5:.25:2.5;
» es = errsurf(p,t,wv,bv,'logsig');
» plotes(wv,bv,es,[60 30])
```

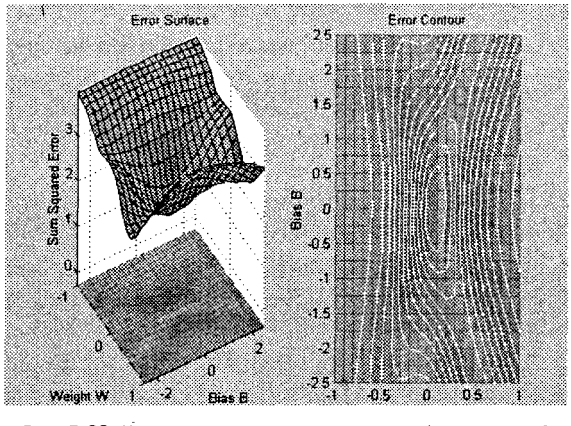


**Рис. 5.22.** Иллюстрация к примеру выполнения функции errsurf

- `maxlinlr(P)` — возвращает максимальную величину коэффициента обучения для линейного слоя нейронов. Здесь `P` — матрица входов.

  При записи в форме `maxlinlr(P,'bias')` функция возвращает максимальную величину коэффициента обучения для линейного слоя нейронов со смещением.

- `gensim(net,st)` — функция генерирует нейросетевой блок Simulink (рис. 5.23) для последующего моделирования НС средствами этого пакета. Пример:

```
» net = newff([0 1],[5 1]); % Создание новой НС
» gensim(net)
```

- `initlay(net)` — функция инициализации слоев нейронной сети. В качестве аргумента использует имя (идентификатор) сети `net`. Возвращает нейронную сеть, слои нейронов в которой инициализированы в соответствии с функцией `net.layers{ i }.initFcn`. В форме `initlay(code)`, где строковая переменная `code` может принимать значения `'pnames'` или `'pdefaults'`, функция возвращает информацию о именах или о значениях по умолчанию параметров инициализации.

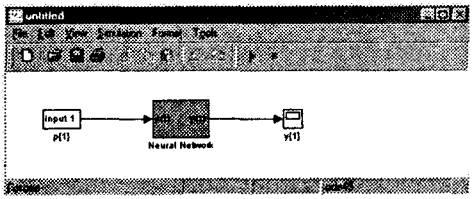


**Рис. 5.23.** Результат выполнения функции gensim

- `initnw(net,i)` — функция инициализации слоя `i`. Возвращает НС, веса и смещения в *i*-м слое которой обновлены в соответствии с алгоритмом инициализации Nguyen—Widrow (так, что зоны «влияния» каждого нейрона в слое распределены равномерно).
- `initwb(net,i)` — почти то же, что в предыдущем случае, но веса и смещения *i*-го слоя инициализируются в соответствии с их собственными функциями инициализации.
- `ddotprod` — функция определения производной от результата `Z` умножения матрицы весов `W` на матрицу входов `P`.

Запись:

```
dZ_dP = ddotprod('p',W,P,Z)
dZ_dW = ddotprod('w',W,P,Z)
```

Примеры:

```
» W = [0 -1 0.2; -1.1 1 0];
» P = [0.1; 0.6; -0.2];
» % Вычисление произведения Z=W*P
» Z = dotprod(W,P)
```

```
Z =
   -0.6400
    0.4900
» dZ_dP = ddotprod('p',W,P,Z)
dZ_dP =
         0   -1.0000    0.2000
   -1.1000    1.0000         0
» dZ_dW = ddotprod('w',W,P,Z)
dZ_dW =
    0.1000
    0.6000
   -0.2000
```

# Примеры создания и использования нейронных сетей

5

## Нейронные сети для аппроксимации функций

Создадим обобщенно-регрессионную НС с именем a для аппроксимации функции вида $y = x^2$ на отрезке [−1, 1], используя следующие экспериментальные данные:

```
x = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1],
y = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1].
```

Процедура создания и использования данной НС описывается следующим образом:

```
» % Задание входных значений
» P = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1];
» % Задание выходных значений
» T = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1];
» a=newgrnn(P,T,0.01); % Создание НС с отклонением 0.01
» Y = sim(a,[-0.9 -0.7 -0.3 0.4 0.8]) % Опрос НС
Y =
    0.8200    0.6400    0.0400    0.0900    0.8100
```

Как видно, точность аппроксимации в данном случае получилась не очень высокой.

Можно попытаться улучшить качество аппроксимации за счет подбора величины отклонения, но в условиях примера приемлемый результат легко достигается при использовании сети с радиальными базисными элементами:

```
» a=newrbe(P,T);
» Y = sim(a,[-0.9 -0.7 -0.3 0.4 0.8]) % Опрос НС
Y =
    0.8100    0.4900    0.0900    0.1600    0.6400
```

Созданную сеть можно сохранить для последующего использования набором в командной строке команды `save('a')`; при этом будет создан файл `a.mat`, то есть файл с именем НС и расширением `mat`. В последующих сеансах работы сохраненную сеть можно загрузить, используя функцию `load('a')`. Естественно, допустимы все другие формы записи операторов `save` и `load`.

Рассмотрим теперь аналогичную задачу, но с использованием линейной НС.

Пусть экспериментальная информация задана значениями

```
x = [+1.0 +1.5 +3.0 -1.2],
y = [+0.5 +1.1 +3.0 -1.0].
```

Процесс создания, обучения и использования линейной НС с именем `b` иллюстрируется приведенным ниже листингом и рис. 5.24.

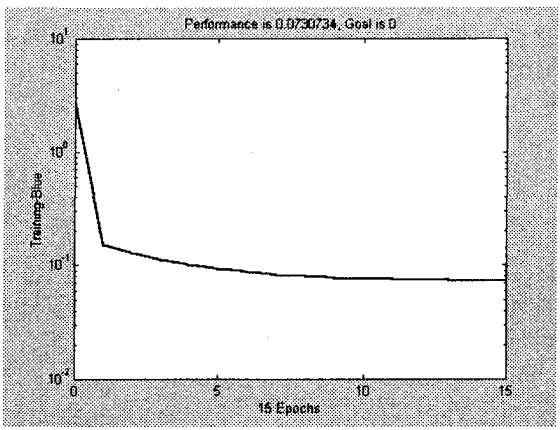


**Рис. 5.24.** Изменение ошибки сети в процессе ее обучения

```
» P = [+1.0 +1.5 +3.0 -1.2];
» T = [+0.5 +1.1 +3.0 -1.0];
» % Определение величины коэффициента обучения
» maxlr=maxlinlr(P,'bias');
» b=newlin([-2 2],1,[0],maxlr);  % Создание линейной НС с именем b
```

```
» b.trainParam.epochs=15;  % Задание количества циклов обучения
» b = train(b,P,T); % Обучение НС
TRAINWB, Epoch 0/15, MSE 2.865/0.
TRAINWB, Epoch 15/15, MSE 0.0730734/0.
TRAINWB, Maximum epoch reached.
» p = -1.2;
» y = sim(b, p) % Опрос сети
y =
   -1.1803
```

## Прогнозирование значений процесса

Рассмотрим теперь такой пример. Предположим, что имеется сигнал (функция времени), описываемый соотношением

$$x(t) = \sin(4\pi t),$$

который подвергается дискретизации с интервалом 0,025 с.

Построим линейную нейронную сеть, позволяющую прогнозировать будущее значение подобного сигнала по 5 предыдущим. Решение данной задачи иллюстрируется ниже.

```
» t = 0:0.025:5; % Задание диапазона времени от 0 до 5 секунд
» x = sin(t*4*pi); % Предсказываемый сигнал
» Q = length(x);
» % Создание входных векторов
» P = zeros(5,Q); % Создание нулевой матрицы P
» P(1,2:Q) = x(1,1:(Q-1));
» P(2,3:Q) = x(1,1:(Q-2));
» P(3,4:Q) = x(1,1:(Q-3));
» P(4,5:Q) = x(1,1:(Q-4));
» P(5,6:Q) =x(1,1:(Q-5));
» s = newlind(P,x);  % Создание новой НС с именем s
» y = sim(s,P);  % Расчет прогнозируемых значений
» % Создание графиков исходного сигнала и прогноза
» plot(t,y,t,x,'*')
» xlabel('Время');
» ylabel('Прогноз -  Сигнал +');
» title('Выход сети и действительные значения');
» % Расчет и создание графика ошибки прогноза
» e = x - y;
» plot(t,e)
» hold on
```

```
» plot([min(t) max(t)],[0 0],':r')
» hold off
» xlabel('Время');
» ylabel('Ошибка');
» title('Сигнал ошибки');
```

В данном случае сеть создавалась с помощью функции newlind, при которой не требуется дополнительного обучения. Судя по графикам результатов, приведенным на рис. 5.25 и 5.26, точность прогноза с использованием линейной НС можно считать хорошей.

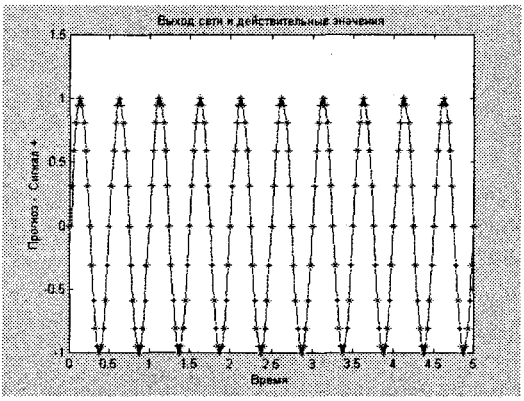


**Рис. 5.25.** Исходный сигнал и прогноз

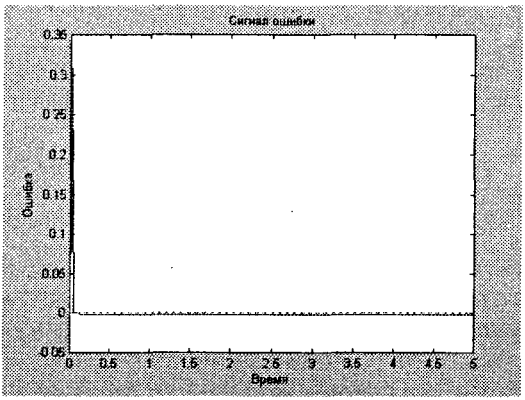


**Рис. 5.26.** Ошибка прогноза

## Использование слоя Кохонена

Рассмотрим задачу автоматического выявления (в режиме обучения без учителя) центров кластеров входов для двумерного случая с использованием слоя Кохонена (слоя «соревнующихся» нейронов). Решение данной задачи приведено ниже.

```
» % Задание диапазонов возможного положения центров кластеров
» X = [0 1; 0 1];
» % Задание параметров для моделирования исходных данных,
» % принадлежащих 8 классам (кластерам)
» clusters = 8;
» points = 10;
» std_dev = 0.05;
» % Моделирование входных данных
» P = nngenc(X,clusters,points,std_dev);
» % Создание слоя Кохонена
» h=newc([0 1;0 1],8,.1);
» % Задание количества циклов обучения
» h.trainParam.epochs=500;
» % Инициализация сети
» h=init(h);
» % Обучение сети
» h=train(h,P);
» w=h.IW{1};
» % Вывод графика исходных данных и выявленных центров кластеров
» plot(P(1,:),P(2,:),'+r');
» hold on;
» plot(w(:,1),w(:,2),'ob');
» xlabel('p(1)');
» ylabel('p(2)');
» % Задание нового входного вектора
» p = [0; 0.2];
» y = sim(h,p) % Опрос сети
y =
   (3,1)        1
```

Работу обученной сети иллюстрируют рис. 5.27 и результат ее опроса (матрица y в конце приведенного листинга, которая выдается в форме разреженной матрицы). В условиях примера предъявленный вектор отнесен к третьему классу (кластеру).

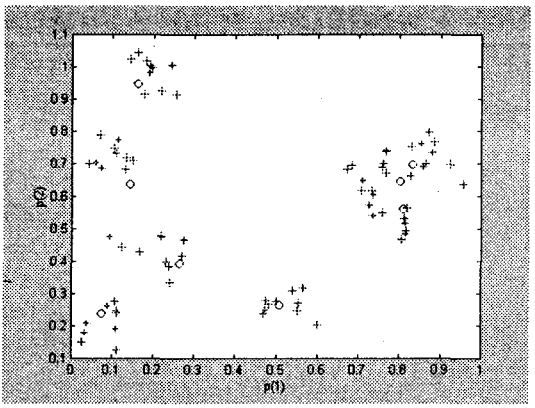


**Рис. 5.27.** Исходные данные и выявленные центры кластеров



## Сеть Хопфилда с двумя нейронами

Рассмотрим сеть Хопфилда, имеющую два нейрона и два устойчивых состояния, отображаемых векторами [1 -1] и [-1 1]. Представим эти векторы с помощью рис. 5.28, выводимого следующей программой:

```
» T = [+1 -1; -1 +1];
» plot(T(1,:),T(2,:),'r*')
» axis([-1.1 1.1 -1.1 1.1]);
» title('Пространство векторов НС Хопфилда');
» xlabel('a(1)');
» ylabel('a(2)');
```

Создадим сеть Хопфилда (с именем H) и проверим ее работу, подав на вход векторы, соответствующие устойчивым точкам. Если сеть работает правильно, она должна выдать эти же векторы без каких-либо изменений.

```
» H=newhop(T); % Создание НС Хопфилда
» [Y,Pf,Af]=sim(H,2,[],T);Y % Опрос сети Хопфилда
Y =
     1    -1
    -1     1
```

Как видно из результата опроса, сеть работает правильно. Подадим теперь на ее вход произвольный вектор.

```
» a = {randss(2,1)}; % Задание случайного вектора
» [y,Pf,Af] = sim(H,{1 50},{},a);
```

```
» plot(T(1,:),T(2,:),'r*')
» axis([-1.1 1.1 -1.1 1.1]);
» record=[cell2mat(a) cell2mat(y)];
» start=cell2mat(a);
» hold on;
» plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:))
» xlabel('a(1)'); ylabel('a(2)');
» title('Результат работы сети Хопфилда');
```

Результат иллюстрируется рис. 5.29.

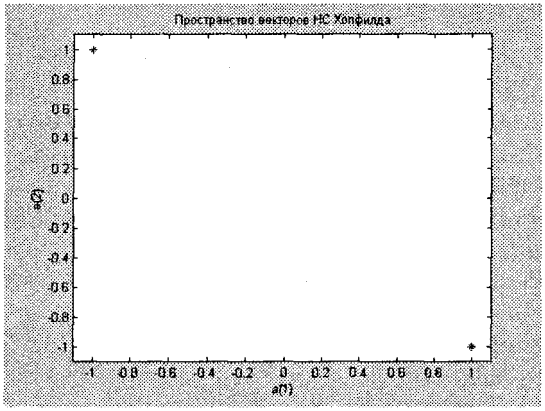


**Рис. 5.28.** Устойчивые точки сети Хопфилда

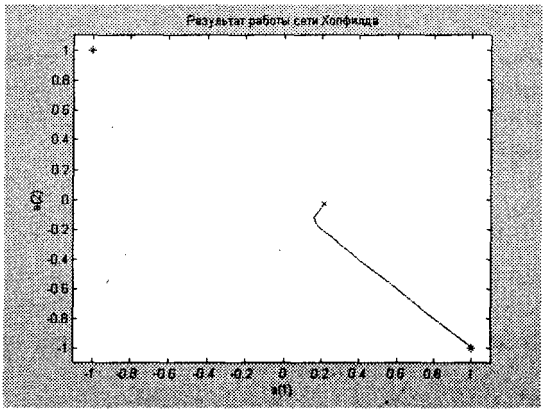


**Рис. 5.29.** Результат работы сети Хопфилда

## Классификация с помощью персептрона

Следующий пример иллюстрирует решение задачи классификации с помощью персептрона. Исходные входные векторы (с указанием их принадлежности к одному из двух классов) и результат настройки персептрона (с именем My_net) представлены на рис. 5.30.

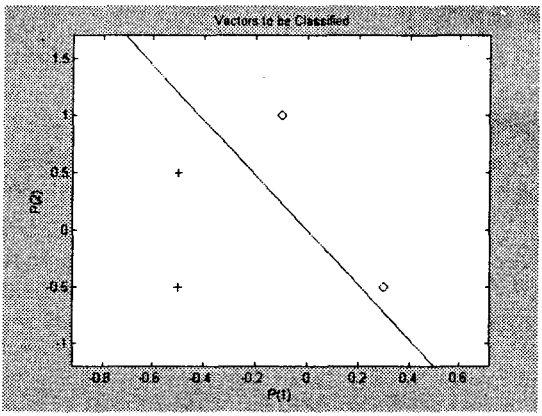


**Рис. 5.30.** Исходные входные векторы и разделяющая линия

```
» % Задание входных векторов с указанием их принадлежности
» % одному из двух классов
» P=[-0.5 -0.5 +0.3 -0.1;-0.5 +0.5 -0.5 +1.0];
» T=[1 1 0 0];
» % Графическое представление исходных векторов
» plotpv(P,T);
» % Создание персептрона с указанием границ изменений входов
» % и 1 нейроном
» My_net=newp([-1 1;-1 1],1);
» E=1;
» % Инициализация персептрона
» My_net=init(My_net);
» % Организация цикла адаптивной настройки персептрона
» % с выводом графика разделяющей линии
» while (sse(E))
      [My_net,Y,E]=adapt(My_net,P,T);
      linehandle=plotpc(My_net.IW{1},My_net.b{1});
      drawnow;
  end;
```

## Адаптивный линейный прогноз

В предыдущем примере настройка НС производилась адаптивно. Отличие такой настройки от выполняемой, например, с помощью метода обратного распространения ошибки заключается в том, что векторы обучающей выборки поступают на вход сети не все «одновременно», а последовательно по одному. При этом после предъявления очередного вектора производится корректировка весов и смещений и может быть произведен опрос сети, затем все повторяется. Адаптивная настройка особенно удобна при работе НС в «реальном» масштабе времени.

Рассмотрим пример задачи прогнозирования значений сигнала (по 5 предыдущим значениям) с использованием указанной настройки.

Предположим, что исходный сигнал определен на интервале времени от 0 до 6 секунд, при этом при $0 \le t \le 4$ с он описывается соотношением $x(t) = \sin(4\pi)$, а при $4 \le t \le 6$ с — соотношением $x(t) = \sin(8\pi)$. График такого сигнала приведен на рис. 5.31.

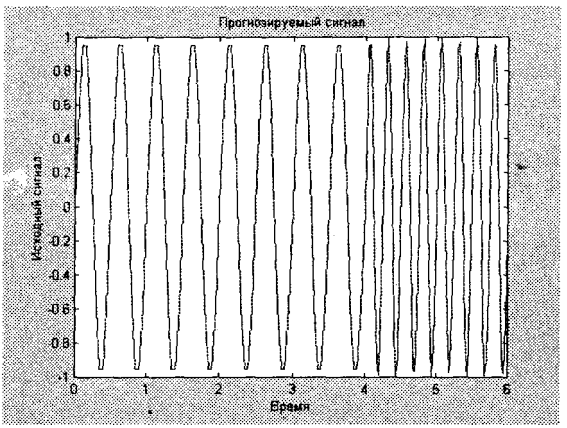


**Рис. 5.31.** График прогнозируемого сигнала

```
» time1 = 0:0.05:4;       % от 0 до 4 секунд
» time2 = 4.05:0.024:6;   % от 4 до 6 секунд
» time = [time1 time2];
» % T определяет исходный сигнал:
» T = con2seq([sin(time1*4*pi) sin(time2*8*pi)]);
» % График исходного сигнала:
```

```
» plot(time,cat(2,T{:}))
» xlabel('Время');
» ylabel('Исходный сигнал');
» title('Прогнозируемый сигнал');
```

Для прогноза значений сигнала создадим линейную НС.

```
» % Входной и целевой прогнозируемый сигнал одинаковы:
» P = T;
» % Задание коэффициента обучения
» lr = 0.1;
» % Для прогноза используются 5 предыдущих значений
» delays = [1 2 3 4 5];
» % Создание и настройка линейной НС
» net = newlin(minmax(cat(2,P{:})),1,delays,lr); % Создание НС
» [net,y,e]=adapt(net,P,T); % Адаптивная настройка сети
» % Графики исходного сигнала и прогноза
» plot(time,cat(2,y{:}),time,cat(2,T{:}),'--')
» xlabel('Время');
» ylabel('Прогнозируемый сигнал');
» title('Исходный сигнал и прогноз');
```

Исходный сигнал и прогноз для этого примера приведены на рис. 5.32.

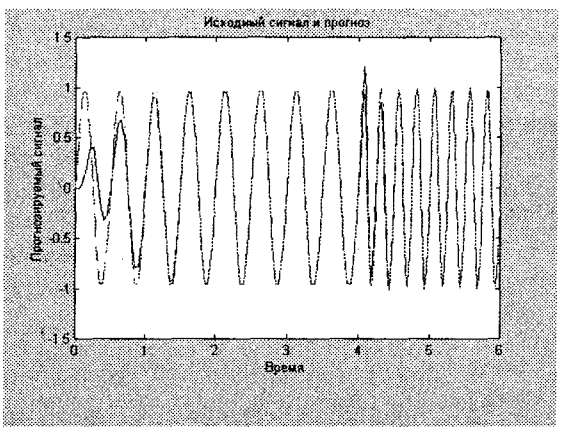


**Рис. 5.32.** Исходный сигнал и прогноз

Как видно из рис. 5.32, полученный результат можно считать удовлетворительным.

## Использование сети Элмана

Рассмотрим задачу восстановления формы сигнала с использованием рекуррентной сети Элмана. Пусть имеются два синусоидальных сигнала — один с единичной амплитудой, другой — с амплитудой, равной двум:

```
p1 = sin(1:20);
p2 = sin(1:20)*2;
```

Пусть целевым сигналом будет сигнал, составленный из их амплитудных значений:

```
t1 = ones(1,20);
t2 = ones(1,20)*2;
```

при этом данные амплитуды чередуются, так что входные и целевые значения могут быть представлены в форме

```
p = [p1 p2 p1 p2];
t = [t1  t2  t1  t2];
```

Преобразуем эти значения в последовательности:

```
Pseq = con2seq(p);
Tseq = con2seq(t);
```

После этого можно непосредственно перейти к проектированию НС. В рассматриваемом случае имеем, очевидно, один вход и один выход, то есть в сети должны присутствовать один входной элемент и один выходной нейрон. Число нейронов в скрытом слое может быть, вообще говоря, любым (оно зависит от сложности задачи); примем, что этот слой содержит 10 нейронов. Дальнейшее решение задачи иллюстрируется ниже, при этом на рис. 5.33 приведен график изменения ошибки сети в процессе ее обучения, а на рис. 5.34 — результаты тестирования сети.

```
» % Задание исходных данных
» p1 = sin(1:20);
» t1 = ones(1,20);
» p2 = sin(1:20)*2;
» t2 = ones(1,20)*2;
» p = [p1 p2 p1 p2];
» t = [t1 t2 t1 t2];
» Pseq = con2seq(p);
» Tseq = con2seq(t);
» % Создание сети Элмана с диапазоном входа [-2, 2], 10 нейронами
» % скрытого слоя, одним выходным нейроном,
```

```
» % функцией активации в виде гиперболического тангенса
» % для нейронов скрытого слоя, линейной функцией активации
» % для выходного нейрона, функцией обучения с адаптацией
» % коэффициента обучения
» net = newelm([-2 2],[10 1],{'tansig','purelin'},'traingdx');
» % Задание параметров обучения:
» % Целевое значение функции ошибки
» net.trainParam.epochs = 500; % Число циклов обучения
» net.trainParam.goal = 0.01;
» net.performFcn = 'sse';  % Задание вида функции ошибки
» % Обучение сети. По умолчанию промежуточные результаты обучения
» % выводятся через 25 циклов
» [net,tr] = train(net,Pseq,Tseq);

TRAINGDX, Epoch 0/500, SSE 443.798/0.01, Gradient 387.439/1e-006
TRAINGDX, Epoch 25/500, SSE 22.1356/0.01, Gradient 7.76533/1e-006
TRAINGDX, Epoch 50/500, SSE 20.0141/0.01, Gradient 2.17503/1e-006
TRAINGDX, Epoch 75/500, SSE 18.9825/0.01, Gradient 1.26454/1e-006
TRAINGDX, Epoch 100/500, SSE 15.8484/0.01, Gradient 9.11439/1e-006
TRAINGDX, Epoch 125/500, SSE 15.1932/0.01, Gradient 2.92454/1e-006
TRAINGDX, Epoch 150/500, SSE 15.1337/0.01, Gradient 2.72534/1e-006
TRAINGDX, Epoch 175/500, SSE 14.9634/0.01, Gradient 2.75817/1e-006
TRAINGDX, Epoch 200/500, SSE 14.3391/0.01, Gradient 3.06463/1e-006
TRAINGDX, Epoch 225/500, SSE 14.2136/0.01, Gradient 3.24916/1e-006
TRAINGDX, Epoch 250/500, SSE 14.1567/0.01, Gradient 3.26074/1e-006
TRAINGDX, Epoch 275/500, SSE 14.0799/0.01, Gradient 3.1542/1e-006
TRAINGDX, Epoch 300/500, SSE 13.7704/0.01, Gradient 3.27526/1e-006
TRAINGDX, Epoch 325/500, SSE 12.6771/0.01, Gradient 3.72479/1e-006
TRAINGDX, Epoch 350/500, SSE 12.1381/0.01, Gradient 5.86736/1e-006
TRAINGDX, Epoch 375/500, SSE 12.315/0.01, Gradient 3.92575/1e-006
TRAINGDX, Epoch 400/500, SSE 12.1414/0.01, Gradient 3.89053/1e-006
TRAINGDX, Epoch 425/500, SSE 11.1606/0.01, Gradient 3.93421/1e-006
TRAINGDX, Epoch 450/500, SSE 8.91345/0.01, Gradient 5.55002/1e-006
TRAINGDX, Epoch 475/500, SSE 8.56053/0.01, Gradient 3.87387/1e-006
TRAINGDX, Epoch 500/500, SSE 8.34089/0.01, Gradient 3.7055/1e-006
TRAINGDX, Maximum epoch reached, performance goal was not met.

» % Построение графика функции ошибки
» semilogy(tr.epoch,tr.perf);
» title('Сумма квадратов ошибок сети Элмана');
» xlabel('Циклы');
```

```
» ylabel('Сумма квадратов ошибок');
» % Тестирование сети
» a = sim(net,Pseq);
» time = 1:length(p);
» time = 1:length(p);
» plot(time,t,'--',time,cat(2,a{:}))
» title('Результаты тестирования сети');
» xlabel('Время');
» ylabel('Заданные значения - -  Выход сети ---');
```

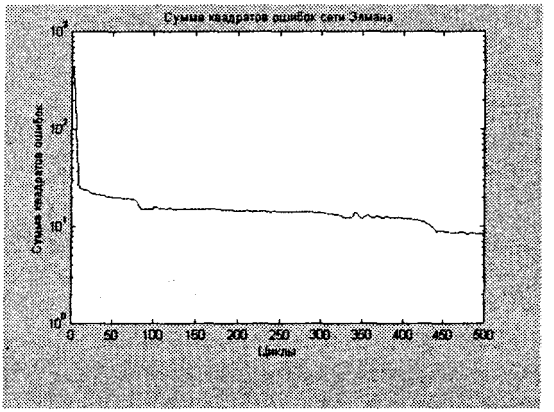


**Рис. 5.33.** Изменение ошибки сети в процессе обучения

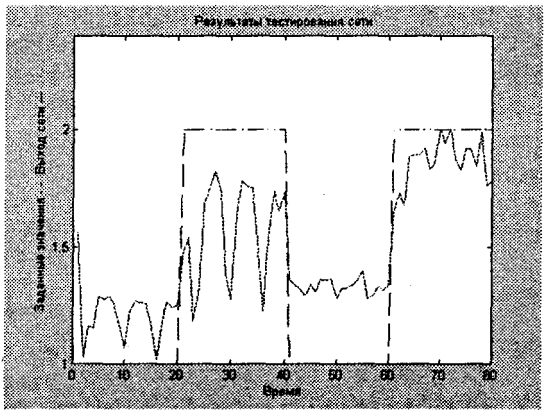


**Рис. 5.34.** Результаты тестирования сети

## Задача классификации: применение сети встречного распространения

Предположим, поставлена следующая задача классификации: задан набор из 10 векторов, представленных в виде столбцов матрицы

$$\mathbf{P} = \begin{bmatrix} -3 & -2 & -2 & 0 & 0 & 0 & 0 & 2 & 2 & 3 \\ 0 & 1 & -1 & 2 & 1 & -1 & -2 & 1 & -1 & 0 \end{bmatrix},$$

а также задан вектор-строка, указывающий принадлежность каждого вектора к одному из двух классов:

```
Tc = [1 1 1 2 2 2 2 1 1 1]
```

Требуется: построить автоматический классификатор подобных векторов, используя приведенные данные как обучающую выборку.

Решение подобной задачи проведем с применением сети встречного распространения так, как это показано ниже.

```
» P=[-3 -2 -2  0  0  0  0 +2 +2 +3; 0 +1 -1 +2 +1 -1 -2 +1 -1  0];
» C = [1 1 1 2 2 2 2 1 1 1];
» % Преобразование вектора C в матрицу T с двумя строками
» T = ind2vec(C);
» % Создание новой сети встречного распространения требует четыре
» % параметра:
» % 1) матрицы минимальных и максимальных значений входных элементов,
» % 2) числа скрытых нейронов,
» % 3) вектор с элементами, указывающими долю каждого из классов,
» % 4) величины коэффициента обучения
» net = newlvq(minmax(P),4,[.6 .4],0.1); % Создание сети
» net.trainParam.epochs=150; % Задание числа циклов обучения
» net.trainParam.show=Inf; % Запрет на выдачу промежуточных результатов
» net=train(net,P,T); % Обучение НС

TRAINWB1, Epoch 150/150
TRAINWB1, Maximum epoch reached.

» Y = sim(net,P) % Тестирование сети
Y =
     1     1     1     0     0     0     0     1     1     1
     0     0     0     1     1     1     1     0     0     0
```

Как видно из результатов тестирования, классификация элементов обучающей выборки произведена точно (три первых и три последних вектора отнесены к первому классу, остальные — ко второму).

## Создание и использование самоорганизующейся карты

Как уже отмечалось, самоорганизующиеся карты можно рассматривать как усовершенствованную модификацию слоя конкурирующих нейронов (слоя Кохонена). От последнего данный вид НС отличается следующим:

1. Нейроны распределяются некоторым пространственным образом (по одному из трех возможных вариантов: в узлах прямоугольной решетки, гексагональной решетки или случайно).
2. На этапе самообучения корректируются веса не только нейрона-«победителя», но и группы нейронов в его некоторой пространственной окрестности.

Назначение самоорганизующихся карт такое же, как у слоя Кохонена — выявление в режиме самообучения центров кластеров входных векторов.

Создание и использование самоорганизующейся карты рассмотрим на примере кластеризации двумерных векторов (исходные данные приведены на рис. 5.35).

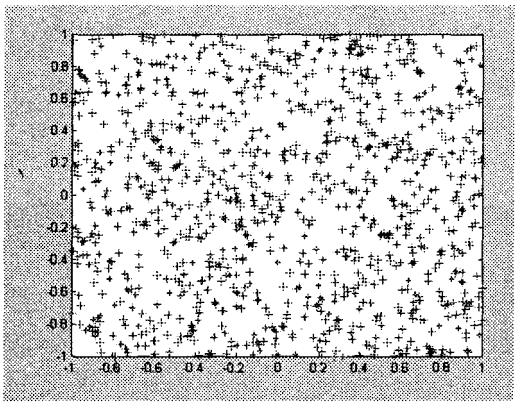


**Рис. 5.35.** Исходные данные

```
» P = rands(2,1000); % Задание случайных двумерных входных векторов
» plot(P(1,:),P(2,:),'+r') % Визуальное изображение входных векторов
» Создание НС с 5x6 = 30 нейронами; все установки — по умолчанию
» net=newsom([0 1; 0 1],[5 6]);
» net.trainParam.epochs=1000; % Задание числа циклов настройки
» net.trainParam.show=200; % Задание периодичности вывода информации
```

```
» net=train(net,P); % Настройка сети
TRAINWB1, Epoch 0/1000
TRAINWB1, Epoch 200/1000
TRAINWB1, Epoch 400/1000
TRAINWB1, Epoch 600/1000
TRAINWB1, Epoch 800/1000
TRAINWB1, Epoch 1000/1000
TRAINWB1, Maximum epoch reached.
» % Выявленные центры кластеров
» plotsom(net.iw{1,1},net.layers{1}.distances);
» p = [0.5; 0.3];
» a = sim(net,p) % Опрос сети
a =
  (11,1)        1
```

Предъявленный на этапе тестирования вектор отнесен сетью к классу 11.

Выявленные центры кластеров представлены на рис. 5.36.

Другие примеры доступны через главное меню MATLAB (команда Help ▸ Examples and Demos, раздел Toolboxes/Neural Networks).

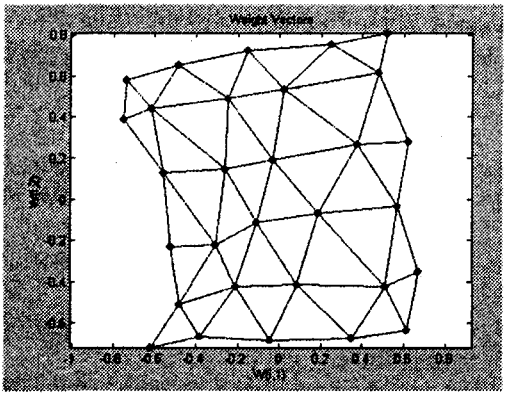


**Рис. 5.36.** Выявленные центры кластеров

# Использование Simulink при построении нейронных сетей

Пакет Neural Network Toolbox содержит ряд блоков, которые могут быть либо непосредственно использованы для построения нейрон-

ных сетей в среде Simulink, либо применяться вместе с рассмотренной выше функцией gensim.

Для вызова этого набора блоков в командной строке необходимо набрать команду neural, после выполнения который появляется окно вида рис. 5.37.

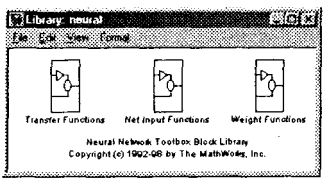


**Рис. 5.37.** Основные нейросетевые блоки Simulink

Каждый из представленных на рис. 5.37 блоков, в свою очередь, является набором (библиотекой) некоторых блоков. Рассмотрим их.

## Блоки функций активации (Transfer Functions)

Двойной щелчок на блоке Transfer Functions приводит к появлению библиотеки функций активации (рис. 5.38).

Каждый из блоков данной библиотеки преобразует подаваемый на него вектор в соответствующий вектор той же размерности (см. табл. 5.1).

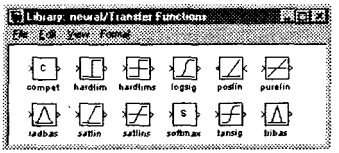


**Рис. 5.38.** Библиотека функций активации

## Блоки преобразования входов сети

Произведя аналогичную рассмотренной операцию, но с блоком Net Input Functions, придем к библиотеке блоков, показанной на рис. 5.39.

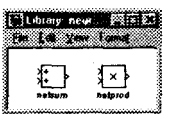


**Рис. 5.39.** Библиотека блоков преобразований сигналов

Блоки данной библиотеки реализуют рассмотренные выше функции преобразования входов сети.

## Блоки весовых коэффициентов

Точно так же (но щелкая на блоке Weight Functions) придем к библиотеке блоков (рис. 5.40), реализующих некоторые функции весов и расстояний.

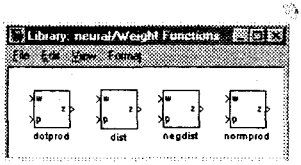


**Рис. 5.40.** Библиотека блоков весовых коэффициентов

Отметим, что в процессе работы со всеми приведенными блоками при задании конкретных числовых значений ввиду особенностей Simulink векторы необходимо представлять как столбцы, а не как строки (как это было до сих пор).

## Формирование нейросетевых моделей

Основной функцией для формирования нейросетевых моделей в Simulink является функция gensim, записываемая в форме

```
gensim(net,st)
```

где net — имя созданной НС, st — интервал дискретизации (если НС не имеет задержек, ассоциированных с ее входами или слоями, значение данного аргумента устанавливается равным −1).

В качестве примера использования средств Simulink рассмотрим следующую задачу.

Пусть входной и целевой векторы имеют вид

```
p = [1 2 3 4 5];
t =  [1 3 5 7 9];
```

Создадим линейную НС и протестируем ее:

```
» p = [1 2 3 4 5];
» t = [1 3 5 7 9];
» net = newlind(p,t);
» y = sim(net,p)
y =
    1.0000    3.0000    5.0000    7.0000    9.0000
```

Затем запустим Simulink командой

```
» gensim(net,-1)
```

Это приведет к открытию блок-диаграммы, показанной на рис. 5.41.

Для проведения тестирования модели щелкнем дважды на левом значке (с надписью Input 1 — Вход 1), что приведет к открытию диалогового окна, показанного на рис. 5.42.

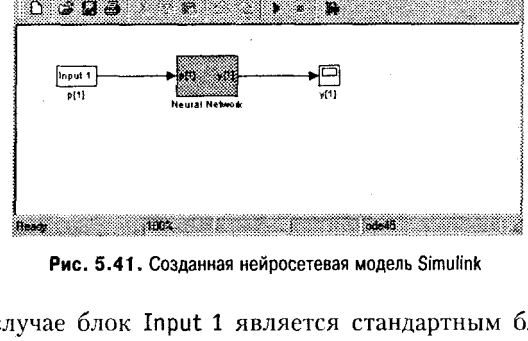


**Рис. 5.41.** Созданная нейросетевая модель Simulink

В данном случае блок Input 1 является стандартным блоком задания константы (Constant). Изменим значение по умолчанию на 2 и нажмем кнопку OK. Затем нажмем кнопку Start в панели инструментов окна модели. Расчет нового значения сетью производится практически мгновенно. Для его вывода необходимо дважды щелкнуть на правом значке (на блоке y(1)). Результат вычислений показан на рис. 5.43 — он равен 3, как и требуется.

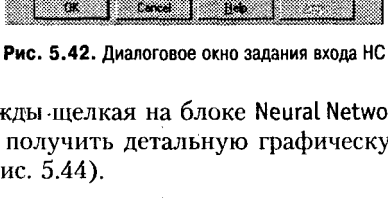


**Рис. 5.42.** Диалоговое окно задания входа НС

Отметим, что дважды щелкая на блоке Neural Network, а затем на блоке Layer 1, можно получить детальную графическую информацию о структуре сети (рис. 5.44).

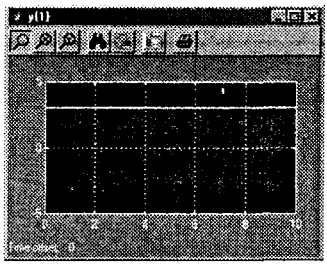

**Рис. 5.43.** Окно с выходом НС

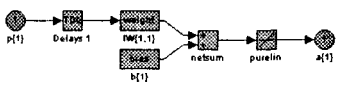


**Рис. 5.44.** Структура созданной НС



С созданной сетью можно проводить различные эксперименты, возможные в среде Simulink; вообще, с помощью команды `gensim` осуществляется интеграция созданных нейросетей в блок-диаграммы этого пакета с использованием имеющихся при этом инструментов моделирования различных систем (например, возможно встраивание нейросетевого регулятора в систему управления и моделирование последней и т. п.).

# Глава 6
# Пакет нечеткой логики Fuzzy Logic Toolbox

## Назначение и возможности пакета Fuzzy Logic Toolbox

Пакет нечеткой логики — это пакет прикладных программ, относящихся к теории *размытых* или *нечетких* множеств и позволяющих конструировать так называемые нечеткие экспертные и/или управляющие системы. Основные возможности пакета:

- Построение систем нечеткого вывода (экспертных систем, регуляторов, аппроксиматоров зависимостей).
- Построение адаптивных нечетких систем (гибридных нейронных сетей).
- Интерактивное динамическое моделирование в Simulink.

Пакет позволяет работу:

- в режиме графического интерфейса;
- в режиме командной строки;
- с использованием блоков и примеров пакета Simulink.

В системе MATLAB 5.3.1 используется версия 2.0.1 данного пакета.

Рекомендуемая литература:

1. Змитрович А. И. Интеллектуальные информационные системы/ НТООО «ТетраСистемс». — Минск, 1997. — 368 с.
2. Прикладные нечеткие системы/Под ред. Т. Тэрано, К. Асаи, М. Сугэно. — М.: Мир, 1993. — 368 с.

## Нечеткая информация и выводы

Пожалуй, наиболее поразительным свойством человеческого интеллекта является способность принимать правильные решения в обстановке неполной и нечеткой информации. Построение моделей приближенных рассуждений человека и использование их в компь-

ютерных системах будущих поколений представляет сегодня одну из важнейших проблем науки.

Значительный шаг в этом направлении сделал профессор Калифорнийского университета (Беркли) Лотфи А. Заде (Lotfi A. Zadeh). Его работа «Fuzzy Sets», появившаяся в 1965 г. в журнале «Information and Control», № 8, заложила основы моделирования интеллектуальной деятельности человека и явилась начальным толчком к развитию новой математической теории.

Заде расширил классическое канторовское понятие множества, допустив, что характеристическая функция (функция принадлежности элемента множеству) может принимать любые значения в интервале [0, 1], а не только значения 0 либо 1. Такие множества были названы им *нечеткими* (fuzzy). Л. Заде определил также ряд операций над нечеткими множествами и предложил обобщение известных методов логического вывода.

Введя затем понятие лингвистической переменной и допустив, что в качестве ее значений (термов) выступают нечеткие множества, Л. Заде создал аппарат для описания процессов интеллектуальной деятельности, включая нечеткость и неопределенность выражений.



Дальнейшие работы профессора Л. Заде и его последователей заложили прочный фундамент новой теории и создали предпосылки для внедрения методов нечеткого управления в инженерную практику.

Спектр их приложений широк: от управления процессом отправления и остановки поездов метрополитена, управления грузовыми лифтами и доменной печью до моделирования работы стиральных машин, пылесосов и СВЧ-печей. При этом нечеткие системы позволяют повысить качество продукции при уменьшении ресурсо- и энергозатрат и обеспечивают более высокую устойчивость к воздействию мешающих факторов по сравнению с традиционными системами автоматического управления.

Другими словами, новые подходы позволяют расширить сферу приложения систем автоматизации за пределы применимости классической теории. В этом плане любопытна точка зрения Л. Заде: «Я считаю, что излишнее стремление к точности стало оказывать действие, сводящее на нет теорию управления и теорию систем, так как оно приводит к тому, что исследования в этой области сосредоточиваются на тех и только тех проблемах, которые поддаются точному решению. В результате многие классы важных проблем, в которых данные, цели и ограничения являются слишком сложными или плохо определенными для того, чтобы допустить точный мате-

матический анализ, оставались и остаются в стороне по той причине, что они не поддаются математической трактовке. Для того чтобы сказать что-либо существенное для проблем подобного рода, мы должны отказаться от наших требований точности и допустить результаты, которые являются несколько размытыми или неопределенными».

Смещение центра исследований нечетких систем в сторону практических приложений привело к постановке целого ряда проблем, таких как новые архитектуры компьютеров для нечетких вычислений, элементная база нечетких компьютеров и контроллеров, инструментальные средства разработки, инженерные методы расчета и разработки нечетких систем управления и многое другое.

Математическая теория нечетких множеств позволяет описывать нечеткие понятия и знания, оперировать этими знаниями и делать нечеткие выводы. Нечеткое управление оказывается особенно полезным, когда технологические процессы являются слишком сложными для анализа с помощью общепринятых количественных методов или когда доступные источники информации интерпретируются качественно, неточно или неопределенно. Нечеткая логика, на которой основано нечеткое управление, ближе по духу к человеческому мышлению и естественным языкам, чем традиционные логические системы. Нечеткая логика в основном обеспечивает эффективные средства отображения неопределенностей и неточностей реального мира. Наличие математических средств отражения нечеткости исходной информации позволяет построить модель, адекватную реальности.

## Нечеткие множества

Пусть $E$ — универсальное множество, $x$ — элемент $E$, а $P$ — некоторое свойство. Обычное (*четкое*) подмножество $A$ универсального множества $E$, элементы которого удовлетворяют свойству $P$, определяется как множество упорядоченных пар

$$A = \{\mu_A(x)/x\},$$

где $\mu_A(x)$ — характеристическая функция, принимающая значение 1, если $x$ удовлетворяет свойству $P$, и 0 — в противном случае.

Нечеткое подмножество отличается от обычного тем, что для элементов $x$ из $E$ нет однозначного ответа «да/нет» относительно свойства $P$. В связи с этим *нечеткое* подмножество $A$ универсального множества $E$ определяется как множество упорядоченных пар с характеристической функцией принадлежности $\mu_A(x)$ (или просто

*функцией принадлежности*, в англоязычной литературе обозначаемой аббревиатурой mf), принимающей значения в некотором вполне упорядоченном множестве $M$ (например, $M = [0, 1]$).

Функция принадлежности указывает степень (или уровень) принадлежности элемента $x$ подмножеству $A$. Множество $M$ называют *множеством принадлежностей*. Если $M = \{0, 1\}$, то нечеткое подмножество $A$ может рассматриваться как обычное или четкое множество.

Примеры записи нечеткого множества.

Пусть $E = \{x_1, x_2, x_3, x_4, x_5\}$, $M = [0, 1]$; $A$ — нечеткое множество, для которого

$\mu_A(x_1) = 0{,}3;$

$\mu_A(x_2) = 0;$

$\mu_A(x_3) = 1;$

$\mu_A(x_4) = 0{,}5;$

$\mu_A(x_5) = 0{,}9.$

Тогда $A$ можно представить в виде

$$A = \{0{,}3/x_1;\ 0/x_2;\ 1/x_3;\ 0{,}5/x_4;\ 0{,}9/x_5\}$$

или

$$A = 0{,}3/x_1 + 0/x_2 + 1/x_3 + 0{,}5/x_4 + 0{,}9/x_5,$$

или

| $A =$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|
| | 0,3 | 0 | 1 | 0,5 | 0,9 |

**ЗАМЕЧАНИЕ**

Здесь знак «плюс» не является обозначением операции сложения, а имеет смысл объединения.

Примеры нечетких множеств.

1. Пусть $E = \{0, 1, 2, ..., 10\}$, $M = [0, 1]$. Нечеткое множество «несколько» можно определить следующим образом: «несколько» = 0,5/3+ +0,8/4+1/5+1/6+0,8/7+0,5/8.
2. Пусть $E = \{0, 1, 2, 3, ..., n, ...\}$. Нечеткое множество «малый» можно определить так:

$$\mu_{\text{«малый»}} = \frac{1}{n\left(1+\left(\frac{n}{10}\right)^2\right)}.$$

3. Пусть $E = \{1,2,3,...,100\}$ и соответствует понятию «возраст», тогда нечеткое множество «молодой» может быть определено с помощью выражения

$$\mu_{\text{«молодой»}} = \begin{cases} 1, & x \in [1,25], \\ \dfrac{1}{1+\left(\dfrac{x-25}{5}\right)^2}, & x > 25. \end{cases}$$

## Функции принадлежности нечеткой логики

В приведенных выше примерах использованы прямые методы определения функций принадлежности, когда эксперт просто задает значение $\mu_A(x)$ для каждого $x \in E$. Как правило прямые методы задания функции принадлежности используются для измеримых понятий, таких как скорость, время, расстояние, давление, температура и т. д., или когда выделяются полярные значения. При прямых методах используются также *групповые* прямые методы, когда, например, группе экспертов предъявляют конкретное лицо и каждый должен дать один из двух ответов, например: «этот человек лысый» или «этот человек не лысый», — тогда количество утвердительных ответов, деленное на общее число экспертов, дает значение $\mu_{\text{«лысый»}}$ для данного лица.

Кроме указанного в нечеткой логике для задания функций принадлежности используются также типовые формы приведенных ниже функций (рис. 6.1):

- треугольная (trimf);
- трапецеидальная (trapmf);
- гауссова (gaussmf);
- двойная гауссова (gauss2mf);
- обобщенная колоколообразная (gbellmf);
- сигмоидальная (sigmf);
- двойная сигмоидальная (dsigmf);

- произведение двух сигмоидальных функций (psigmf);
- Z-функция;
- S-функция;
- Pi-функция.

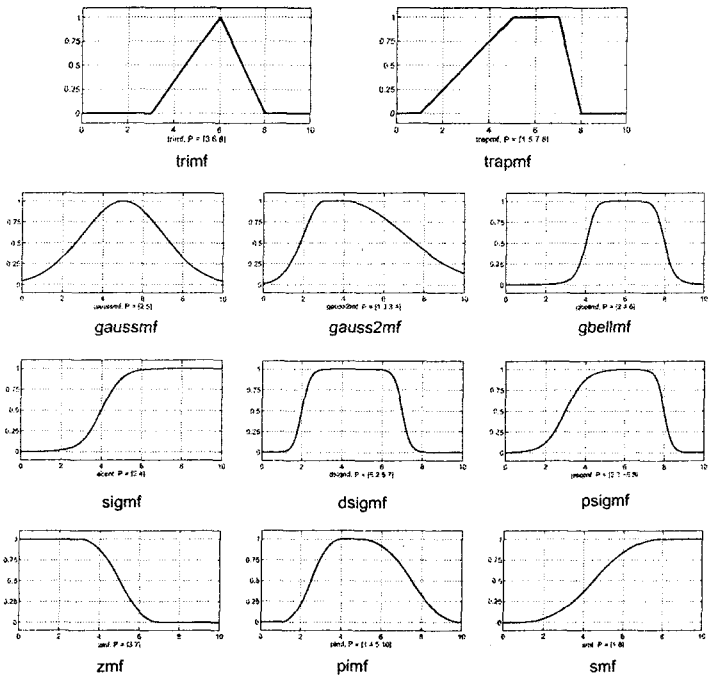


**Рис. 6.1.** Типовые функции принадлежности нечетких множеств

Конкретный вид данных функций определяется значениями параметров, входящих в их аналитические представления, например:

$$\text{trimf}(x, a, b, c) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right),$$

$$\text{trapmf}(x, a, b, c, d) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{c-d}\right), 0\right),$$

$$\text{gaussmf}(x, \sigma, c) = e^{-\left(\frac{x-c}{\sigma}\right)^2},$$

$$\text{bellmf}(x, a, b, c) = \frac{1}{1 + \left|\frac{x - c}{a}\right|^{2b}},$$

$$\text{sigmf}(x, a, c) = \frac{1}{1 + \exp(-a(x - c))}$$

и т. д.

# Операции над нечеткими множествами

## Логические операции

Рассмотрим основные логические операции, которые возможны с нечеткими множествами.

### Включение

Пусть $A$ и $B$ — нечеткие множества на универсальном множестве $E$. Говорят, что $A$ содержится в $B$, если $\forall x \in E \ \mu_A(x) \le \mu_B(x)$.

*Обозначение*: $A \subset B$.

Иногда используют термин «доминирование», то есть в случае, когда $A \subset B$, говорят, что $B$ доминирует над $A$.

### Равенство

$A$ и $B$ равны, если $\forall x \in E \ \mu_A(x) = \mu_B(x)$.

*Обозначение*: $A = B$.

### Дополнение

Пусть $M = [0, 1]$, $A$ и $B$ — нечеткие множества, заданные на $E$. $A$ и $B$ дополняют друг друга, если

$$\forall x \in E \ \mu_A(x) = 1 - \mu_B(x).$$

*Обозначение*: $B = \overline{A}$ или $A = \overline{B}$.

Очевидно, что $\left(\overline{\overline{A}}\right) = A$ (дополнение определено для $M = [0, 1]$, но очевидно, что его можно определить для любого упорядоченного $M$).

### Пересечение

$A \cap B$ — наибольшее нечеткое подмножество, содержащееся одновременно в $A$ и $B$:

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x).$$

### Объединение

$A \cup B$ — наименьшее нечеткое подмножество, включающее как $A$, так и $B$, с функцией принадлежности

$$\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x).$$

### Разность

$A - B = A \cap \overline{B}$ с функцией принадлежности

$$\mu_{A-B}(x) = \mu_{A\cap\overline{B}(x)} = \min(\mu_A(x), 1 - \mu_B(x)).$$

Примеры. Пусть

$A = 0{,}4/x_1 + 0{,}2/x_2 + 0/x_3 + 1/x_4;$

$B = 0{,}7/x_1 + 0{,}9/x_2 + 0{,}1/x_3 + 1/x_4;$

$C = 0{,}1/x_1 + 1/x_2 + 0{,}2/x_3 + 0{,}9/x_4.$

Здесь:

1. $A \subset B$, то есть $A$ содержится в $B$ или $B$ доминирует над $A$, $C$ несравнимо ни с $A$, ни с $B$, то есть пары $\{A, C\}$ и $\{B, C\}$ — пары недоминирующих нечетких множеств.
2. $A \neq B \neq C$.
3. $\overline{A} = 0{,}6/x_1 + 0{,}8/x_2 + 1/x_3 + 0/x_4;$
   $\overline{B} = 0{,}3/x_1 + 0{,}1/x_2 + 0{,}9/x_3 + 0/x_4.$
4. $A \cap B = 0{,}4/x_1 + 0{,}2/x_2 + 0/x_3 + 1/x_4.$
5. $A \cup B = 0{,}7/x_1 + 0{,}9/x_2 + 0{,}1/x_3 + 1/x_4.$
6. $A - B = A \cap \overline{B} = 0{,}3/x_1 + 0{,}1/x_2 + 0/x_3 + 0/x_4;$
   $B - A = \overline{A} \cap B = 0{,}6/x_1 + 0{,}8/x_2 + 0{,}1/x_3 + 0/x_4.$

Для нечетких множеств можно строить визуальное представление. Рассмотрим прямоугольную систему координат, на оси ординат которой откладываются значения $\mu_A(x)$, а на оси абсцисс в произвольном порядке расположены элементы $E$. Если $E$ по своей природе упорядочено, то этот порядок желательно сохранить в расположении элементов на оси абсцисс. Такое представление делает наглядными простые логические операции над нечеткими множествами (рис. 6.2).

В верхней части рисунка заштрихованная область соответствует нечеткому множеству $A$ и, если говорить точно, изображает область значений $A$ и всех нечетких множеств, содержащихся в $A$. В нижней части рисунка показаны $\overline{A}$, $A \cap \overline{A}$, $A \cup \overline{A}$.

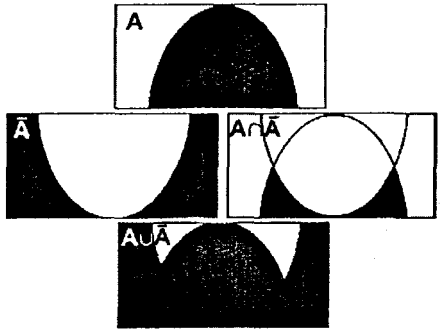


**Рис. 6.2.** Графическая интерпретация логических операций

**ЗАМЕЧАНИЕ**

Введенные выше операции над нечеткими множествами основаны на использовании операций max и min. В теории нечетких множеств разрабатываются вопросы построения обобщенных, параметризованных операторов пересечения, объединения и дополнения, позволяющих учесть разнообразные смысловые оттенки соответствующих им связок «и», «или», «не».

Один из подходов к операторам пересечения и объединения заключается в их определении в классе треугольных норм и конорм.

*Треугольной нормой* (Т-нормой) называется двуместная действительная функция $T$: $[0, 1] \times [0, 1] \to [0, 1]$, удовлетворяющая следующим условиям:

1) $T(0, 0) = 0$; $T(\mu_A, 1) = \mu_A$; $T(1, \mu_A) = \mu_A$ — ограниченность;
2) $T(\mu_A, \mu_B) \le T(\mu_C, \mu_D)$, если $\mu_A \le \mu_C$, $\mu_B \le \mu_D$ — монотонность;
3) $T(\mu_A, \mu_B) = T(\mu_B, \mu_A)$ — коммутативность;
4) $T(\mu_A, T(\mu_B, \mu_C)) = T(T(\mu_A, \mu_B), \mu_C)$ — ассоциативность;

Простым случаем треугольных норм являются:

- $\min(\mu_A, \mu_B)$,
- произведение $\mu_A \cdot \mu_B$,
- $\max(0, \mu_A + \mu_B - 1)$.

*Треугольной конормой* (Т-конормой или S-нормой) называется двуместная действительная функция $S$: $[0, 1] \times [0, 1] \to [0, 1]$, со свойствами:

1) $S(1,1) = 1$; $S(\mu_A, 0) = \mu_A$; $S(0, \mu_A) = \mu_A$ — ограниченность;

2) $S(\mu_A, \mu_B) \geq S(\mu_C, \mu_D)$, если $\mu_A \geq \mu_C$, $\mu_B \geq \mu_D$ — монотонность;

3) $S(\mu_A, \mu_B) = S(\mu_B, \mu_A)$ — коммутативность;

4) $S(\mu_A, S(\mu_B, \mu_C)) = S(S(\mu_A, \mu_B), \mu_C)$ — ассоциативность.

Примеры Т-конорм:

- $\max(\mu_A, \mu_B)$,
- $\mu_A + \mu_B - \mu_A \cdot \mu_B$,
- $\min(1, \mu_A + \mu_B)$.

### Алгебраические операции

Теперь представим алгебраические операции с нечеткими множествами.

*Алгебраическое произведение* $A$ и $B$ обозначается $A \cdot B$ и определяется так:

$$\forall x \in E \quad \mu_{A \cdot B}(x) = \mu_A(x)\mu_B(x).$$

*Алгебраическая сумма* этих множеств обозначается $A \hat{+} B$ и определяется так:

$$\forall x \in E\, \mu_{A \hat{+} B}(X) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x).$$

На основе операции алгебраического произведения определяется операция *возведения в степень* $\alpha$ нечеткого множества $A$, где $\alpha$ — положительное число. Нечеткое множество $A^\alpha$ определяется функцией принадлежности $\mu_{A^\alpha}(x) = \mu_A^\alpha(x)$. Частными случаями возведения в степень являются:

- $\mathrm{CON}(A) = A^2$ — операция *концентрирования* (уплотнения),
- $\mathrm{DIL}(A) = A^{0,5}$ — операция *растяжения*,

которые используются при работе с лингвистическими неопределенностями.

*Декартово (прямое) произведение нечетких множеств.* Пусть $A_1$, $A_2$, ..., $A_n$ — нечеткие подмножества универсальных множеств $E_1$, $E_2$, ..., $E_n$ соответственно. Декартово или прямое произведение $A = A_1 \times \times A_2 \times \cdots \times A_n$ является нечетким подмножеством множества $E = E_1 \times E_2 \times \cdots \times E_n$ с функцией принадлежности

$$\mu_A(x_1, x_2, ..., x_n) = \min\{\mu_{A_1}(x_1), \mu_{A_2}(x_2), ..., \mu_{A_i}(x_n)\}.$$

## Нечеткие отношения

Пусть $E = E_1 \times E_2 \times \cdots \times E_n$ — прямое произведение универсальных множеств и $M$ — некоторое множество принадлежностей (например,

$M = [0, 1]$). Нечеткое $n$-арное отношение определяется как нечеткое подмножество $R$ на $E$, принимающее свои значения в $M$. В случае $n = 2$ и $M = [0, 1]$ нечетким отношением $R$ между множествами $X = E_1$ и $Y = E_2$ будет называться функция $R$: $(X, Y) \to [0, 1]$, которая ставит в соответствие каждой паре элементов $(x, y) \in X \times Y$ величину $\mu_R(x, y) \in [0, 1]$. *Обозначение*: нечеткое отношение на $X \times Y$ запишется в следующем виде: $x \in X$, $y \in Y$: $xRy$. В случае, когда $X = Y$, то есть $X$ и $Y$ совпадают, нечеткое отношение $R$: $X \times X \to [0, 1]$ называется нечетким отношением на множестве $X$.

Примеры.

1. Пусть $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2, y_3, y_4\}$, $M = [0, 1]$. Нечеткое отношение $R = XRY$ может быть задано, к примеру, табл. 6.1.

**Таблица 6.1.** Задание нечеткого отношения

| $R$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ |
|---|---|---|---|---|
| $x_1$ | 0 | 0 | 0,1 | 0,3 |
| $x_2$ | 0 | 0,8 | 1 | 0,7 |
| $x_3$ | 1 | 0,5 | 0,6 | 1 |

2. Пусть $X = Y = (-\infty, \infty)$, то есть множество всех действительных чисел. Отношение $x \gg y$ ($x$ много больше $y$) можно задать следующей функцией принадлежности:

$$\mu_R = \begin{cases} 0, & \text{если } x \le y, \\ \dfrac{1}{1 + \dfrac{1}{(x-y)^2}}, & \text{если } x < y. \end{cases}$$

Отношение $R$, для которого $\mu_R(x, y) = e^{-k(x-y)^2}$, при достаточно больших $k$ можно интерпретировать так: «$x$ и $y$ — близкие друг к другу числа».

# Операции над нечеткими отношениями

## Объединение двух отношений

Объединение двух отношений $R_1$ и $R_2$ обозначается $R_1 \cup R_2$ и определяется выражением

$$\mu_{R_1 \cup R_2}(x, y) = \mu_{R_1}(x, y) \vee \mu_{R_2}(x, y) = \max(\mu_{R_1}(x, y), \mu_{R_2}(x, y)).$$

## Пересечение двух отношений

Пересечение двух отношений $R_1$ и $R_2$ обозначается $R_1 \cap R_2$ и определяется выражением

$$\mu_{R_1 \cap R_2}(x, y) = \mu_{R_1}(x, y) \wedge \mu_{R_2}(x, y) = \min(\mu_{R_1}(x, y), \mu_{R_2}(x, y)).$$

## Алгебраическое произведение двух отношений

Алгебраическое произведение двух отношений $R_1$ и $R_2$ обозначается $R_1 \cdot R_2$ и определяется выражением

$$\mu_{R_1 \cdot R_2}(x, y) = \mu_{R_1}(x, y) \cdot \mu_{R_2}(x, y).$$

## Алгебраическая сумма двух отношений

Алгебраическая сумма двух отношений $R_1$ и $R_2$ обозначается $R_1 \hat{+} R_2$ и определяется выражением

$$\mu_{R_1 + R_2}(x, y) = \mu_{R_1}(x, y) + \mu_{R_2}(x, y) + \mu_{R_1}(x, y) \cdot \mu_{R_2}(x, y).$$

## Дополнение отношения

Дополнение отношения $R$ обозначается $\overline{R}$ и определяется функцией принадлежности

$$\mu_{\overline{R}}(x, y) = 1 - \mu_R(x, y).$$

## Обычное отношение, ближайшее к нечеткому

Пусть $R$ — нечеткое отношение с функцией принадлежности $\mu_R(x, y)$. *Обычное отношение, ближайшее к нечеткому, обозначается* $\underline{R}$ и определяется выражением

$$\mu_{\underline{R}}(x, y) = \begin{cases} 0, & \text{если } \mu_R(x, y) < 0{,}5, \\ 1 & \text{если } \mu_R(x, y) > 0{,}5, \\ 0 \text{ или } 1, & \text{если } \mu_R(x, y) = 0{,}5. \end{cases}$$

По договоренности принимают $\mu_{\underline{R}}(x, y) = 0$ при $\mu_R(x, y) = 0.5$.

## Композиция (свертка) двух нечетких отношений

Пусть $R_1$ — нечеткое отношение $R_1$: $(X \times Y) \to [0, 1]$ между $X$ и $Y$, и $R_2$ — нечеткое отношение $R_2$: $(Y \times Z) \to [0, 1]$ между $Y$ и $Z$. Нечеткое отношение между $X$ и $Z$, обозначаемое $R_2 \bullet R_1$, определенное через $R_1$ и $R_2$ выражением

$$\mu_{R_1 \bullet R_2}(x, z) = \bigvee_y [\mu_{R_1}(x, y) \wedge \mu_{R_2}(y, z)] = \max(\min(\mu_{R_1}(x, y), \mu_{R_2}(y, z))),$$

где символом $\bigvee_y$ обозначена операция выбора наибольшего по $y$ значения, называется (max-min)-композицией ((max-min)-сверткой) отношений $R_1$ и $R_2$.

**Пример.** Пусть отношения $R_1$ и $R_2$ определяются следующим образом:

| $R_1$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|
| $x_1$ | 0,1 | 0,7 | 0,4 |
| $x_2$ | 1 | 0,5 | 0 |

| $R_2$ | $z_1$ | $z_2$ | $z_3$ | $z_4$ |
|---|---|---|---|---|
| $y_1$ | 0,9 | 0 | 1 | 0,2 |
| $y_2$ | 0,3 | 0,6 | 0 | 0,9 |
| $y_3$ | 0,1 | 1 | 0 | 0,5 |

Тогда

| $R_1 \bullet R_2$ | $z_1$ | $z_2$ | $z_3$ | $z_4$ |
|---|---|---|---|---|
| $x_1$ | 0,3 | 0,6 | 0,1 | 0,7 |
| $x_2$ | 0,9 | 0,5 | 1 | 0,5 |

При этом

$\mu_{R_1 \bullet R_2}(x_1, z_1) = [\mu_{R_1}(x_1, y_1) \wedge \mu_{R_2}(y_1, z_1)] \vee [\mu_{R_1}(x_1, y_2) \wedge \mu_{R_2}(y_2, z_1)] \vee$
$\vee [\mu_{R_1}(x_1, y_3) \wedge \mu_{R_2}(y_3, z_1)] =$
$= (0,1 \wedge 0,9) \vee (0,7 \wedge 0,3) \vee (0,4 \wedge 0,1) = 0,1 \vee 0,3 \vee 0,1 = 0,3;$

$\mu_{R_1 \bullet R_2}(x_1, z_2) = (0,1 \wedge 0) \vee (0,7 \wedge 0,6) \vee (0,4 \wedge 1) = 0 \vee 0,6 \vee 0,4 = 0,6;$

$\mu_{R_1 \bullet R_2}(x_1, z_3) = 0,1;$

...

$\mu_{R_1 \bullet R_2}(x_2, z_5) = 0,5.$

**ЗАМЕЧАНИЕ**

В данном примере вначале использован «аналитический» способ композиции отношений $R_1$ и $R_2$, то есть $i$-я строка $R_1$ «умножается» на $j$-й столбец $R_2$ с использованием операции $\wedge$, полученный результат «свертывается» с использованием операции $\vee$ в $\mu(x_i, z_j)$.

## (max-*)-композиция

В выражении $\mu_{R_1 \bullet R_2}(x, z) = \bigvee_y [\mu_{R_1}(x, y) \wedge \mu_{R_2}(y, z)]$ для (max-min)-композиции отношений $R_1$ и $R_2$ операцию $\wedge$ (то есть min) можно

заменить любой другой, для которой выполняются те же ограничения, что и для Λ: ассоциативность и монотонность (в смысле неубывания) по каждому аргументу. Тогда

$$\mu_{R_1 \bullet R_2}(x, z) = \bigvee_y [\mu_{R_1}(x, y) * \mu_{R_2}(y, z)].$$

В частности, операция Λ может быть заменена алгебраическим умножением, тогда говорят о (max-prod)-композиции.

## Нечеткие выводы

Используемый в различного рода экспертных и управляющих системах механизм нечетких выводов в своей основе имеет базу знаний, формируемую специалистами предметной области в виде совокупности нечетких предикатных правил вида

$П_1$: если $x$ есть $A_1$, то $y$ есть $B_1$,

$П_2$: если $x$ есть $A_2$, то $y$ есть $B_2$,

...

$П_n$: если $x$ есть $A_n$, то $y$ есть $B_n$,

где $x$ — входная переменная (имя для известных значений данных), $y$ — переменная вывода (имя для значения данных, которое будет вычислено); $A$ и $B$ — функции принадлежности, определенные, соответственно, на $x$ и $y$.

Пример подобного правила:

Если $x$ — низко, то $y$ — высоко.

Приведем более детальное пояснение. Знание эксперта $A \to B$ отражает нечеткое причинное отношение предпосылки и заключения, поэтому его можно назвать нечетким отношением и обозначить через $R$:

$$R = A \to B,$$

где «→» называют *нечеткой импликацией.*

Отношение $R$ можно рассматривать как нечеткое подмножество прямого произведения $X \times Y$ полного множества предпосылок $X$ и заключений $Y$. Таким образом, процесс получения (нечеткого) результата вывода $B'$ с использованием данного наблюдения $A'$ и знания $A \to B$ можно представить в виде формулы

$$B' = A' \bullet R = A' \bullet (A \to B),$$

где «•» — введенная выше операция свертки.

Как операцию композиции, так и операцию импликации в алгебре нечетких множеств можно реализовывать по-разному (при этом, естественно, будет разниться и итоговый получаемый результат), но в любом случае общий логический вывод осуществляется за следующие четыре этапа.

1. *Нечеткость* (введение нечеткости, фаззификация, fuzzification). Функции принадлежности, определенные на входных переменных, применяются к их фактическим значениям для определения степени истинности каждой предпосылки каждого правила.
2. *Логический вывод.* Вычисленное значение истинности для предпосылок каждого правила применяется к заключениям каждого правила. Это приводит к одному нечеткому подмножеству, которое будет назначено каждой переменной вывода для каждого правила. В качестве правил логического вывода обычно используются только операции min (минимум) или prod (умножение). В логическом выводе min функция принадлежности вывода «отсекается» по высоте, соответствующей вычисленной степени истинности предпосылки правила (нечеткая логика «И»). В логическом выводе prod функция принадлежности вывода масштабируется при помощи вычисленной степени истинности предпосылки правила.
3. *Композиция.* Нечеткие подмножества, назначенные для каждой переменной вывода (во всех правилах), объединяются вместе, чтобы сформировать одно нечеткое подмножество для каждой переменной вывода. При подобном объединении обычно используются операции max (максимум) или sum (сумма). При композиции max комбинированный вывод нечеткого подмножества конструируется как поточечный максимум по всем нечетким подмножествам (нечеткая логика «ИЛИ»). При композиции sum комбинированный вывод нечеткого подмножества конструируется как поточечная сумма по всем нечетким подмножествам, назначенным переменной вывода правилами логического вывода.
4. В заключение (дополнительно) — *приведение к четкости* (дефаззификация, defuzzification), которое используется, когда полезно преобразовать нечеткий набор выводов в четкое число.

Рассмотрим следующие наиболее употребительные модификации алгоритма нечеткого вывода, полагая, для простоты, что базу знаний организуют два нечетких правила вида

$П_1$: если $x$ есть $A_1$ и $y$ есть $B_1$, тогда $z$ есть $C_1$,

$П_2$: если $x$ есть $A_2$ и $y$ есть $B_2$, тогда $z$ есть $C_2$,

где $x$ и $y$ — имена входных переменных, $z$ — имя переменной вывода, $A_1$, $A_2$, $B_1$, $B_2$, $C_1$, $C_2$ — некоторые заданные функции принадлежности, при этом четкое значение $z_0$ необходимо определить на основе приведенной информации и четких значений $x_0$ и $y_0$.

## Алгоритм Мамдани (Mamdani)

Данный алгоритм математически может быть описан следующим образом.

1. Нечеткость: находятся степени истинности для предпосылок каждого правила: $A_1(x_0)$, $A_2(x_0)$, $B_1(y_0)$, $B_2(y_0)$.
2. Нечеткий вывод: находятся уровни «отсечения» для предпосылок каждого из правил (с использованием операции min):

   $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$,

   $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$,

   где через «$\wedge$», как и раньше, обозначена операция логического минимума (min), затем находятся усеченные функции принадлежности

   $C'_1(z) = (\alpha_1 \wedge C_1(z))$,

   $C'_2(z) = (\alpha_2 \wedge C_2(z))$.
3. Композиция: с использованием операции max (обозначаемой как «$\vee$») производится объединение найденных усеченных функций, что приводит к получению итогового нечеткого подмножества для переменной выхода с функцией принадлежности

   $\mu_\Sigma(z) = C(z) = C'_1(z) \vee C'_2(z) = (\alpha_1 \wedge C_1(z)) \vee (\alpha_2 \wedge C_2(z))$.
4. Наконец, приведение к четкости (для нахождения $z_0$) проводится, например, центроидным методом (как центр тяжести для кривой $\mu_\Sigma(z)$):

$$z_0 = \frac{\int\limits_\Omega z\mu_\Sigma(z)dz}{\int\limits_\Omega \mu_\Sigma(z)dz}.$$

Алгоритм иллюстрируется рис. 6.3.

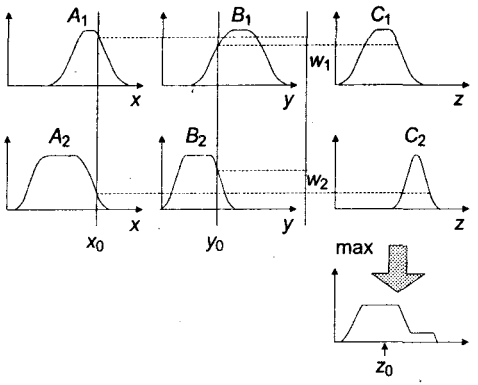


**Рис. 6.3.** Иллюстрация к алгоритму Мамдани

## Алгоритм Сугэно (Sugeno)

Сугэно (Sugeno) и Такаги (Takagi) использовали набор правил в следующей форме (как и раньше, приводим пример двух правил):

$П_1$: если $x$ есть $A_1$ и $y$ есть $B_1$, тогда $z_1 = a_1x + b_1y$,

$П_2$: если $x$ есть $A_2$ и $y$ есть $B_2$, тогда $z_2 = a_2x + b_2y$.

Представление алгоритма.

1. Первый этап — как в алгоритме Мамдани.
2. На втором этапе находятся $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$, $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$ и индивидуальные выходы правил:

   $z^*_1 = a_1x_0 + b_1y_0$,

   $z^*_2 = a_2x_0 + b_2y_0$.
3. На третьем этапе определяется четкое значение переменной вывода:

$$z_0 = \frac{\alpha_1 z_1^* + \alpha_2 z_2^*}{\alpha_1 + \alpha_2}.$$

Алгоритм иллюстрируется рис. 6.4.

Приведенное представление относится к алгоритму Сугэно 1-го порядка. Если правила записаны в форме

$П_1$: если $x$ есть $A_1$ и $y$ есть $B_1$, то $z_1 = c_1$,

$П_2$: если $x$ есть $A_2$ и $y$ есть $B_2$, то $z_2 = c_2$,

то говорят, что задан алгоритм Сугэно 0-го порядка (рис. 6.5).

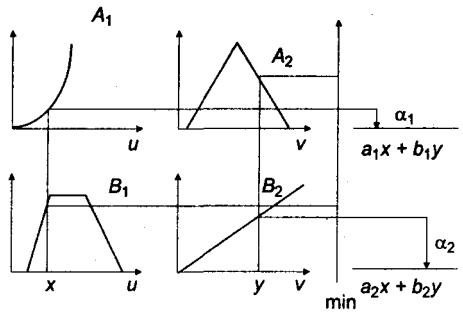


**Рис. 6.4.** Иллюстрация к алгоритму Сугэно

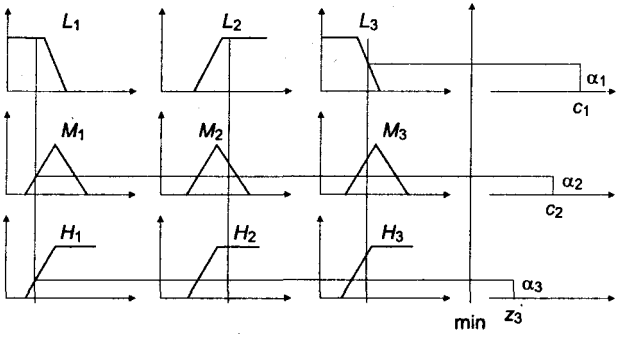


**Рис. 6.5.** Алгоритм Сугэно 0-го порядка

## Методы приведения к четкости

Выше уже был рассмотрен один из данных методов — *центроидный* (centroid of area). Приведем соответствующие формулы еще раз.

Для непрерывного варианта:

$$z_0 = \frac{\int\limits_{\Omega} z\,C(z)\mathrm{d}z}{\int\limits_{\Omega} C(z)\mathrm{d}z};$$

для дискретного варианта:

$$z_0 = \frac{\sum\limits_{i=1}^{n} \alpha_i z_i}{\sum\limits_{i=1}^{n} \alpha_i}.$$

Другие возможные методы приведения к четкости иллюстрирует рис. 6.6: наименьший максимум (smallest of max, som), наибольший максимум (largest of max, lom), средний максимум (mean of max, mom), бисекторный (bisector of area).

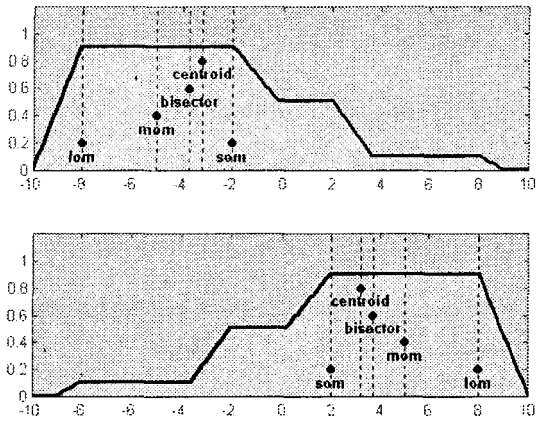


**Рис. 6.6.** Методы приведения к четкости



# Эффективность систем принятия решений

*Эффективность систем нечеткой логики базируется на ряде следующих результатов:*

- в 1992 г. Ванг (Wang) доказал теорему: для каждой вещественной непрерывной функции $G(x)$, заданной на компакте $U$ и для произвольного $\varepsilon > 0$, существует нечеткая экспертная система, формирующая выходную функцию $F(x)$ такую, что

  $$\sup_{x \in U} \|F(x) - G(x)\| \le \varepsilon,$$

  где $\|\bullet\|$ — символ принятого расстояния между функциями;
- в 1995 г. Кастро (Castro) показал, что логический контроллер Мамдани при определенных условиях также является универсальным аппроксиматором;
- согласно знаменитой теореме FAT (Fuzzy Approximation Theorem), доказанной Б. Коско (B. Kosko) в 1993 г., любая математическая система может быть аппроксимирована системой, основанной на нечеткой логике.

Вообще говоря, системы с нечеткой логикой целесообразно применять в следующих случаях:

1) для сложных процессов, когда нет простой математической модели;
2) если экспертные знания об объекте или о процессе можно сформулировать только в лингвистической, то есть в словесной форме.

## Гибридные сети

Каждая разновидность систем искусственного интеллекта имеет свои особенности, например, по возможностям обучения, обобщения и выработки выводов, что делает ее наиболее пригодной для решения одного класса задач и менее пригодной — для другого.

Например, нейронные сети хороши для задач распознавания образов, но весьма неудобны для выяснения вопроса, как они такое распознавание осуществляют. Они могут автоматически приобретать знания, но процесс их обучения зачастую происходит достаточно медленно, а анализ обученной сети весьма сложен (обученная сеть для пользователя обычно представляется как черный ящик). При этом какую-либо априорную информацию (знания эксперта) для ускорения процесса обучения в нейронную сеть ввести невозможно

Системы с нечеткой логикой, напротив, хороши для объяснения получаемых с их помощью выводов, но они не могут автоматически приобретать знания для использования их в механизмах выводов. Необходимость разбиения универсальных множеств на отдельные области, как правило, ограничивает количество входных переменных в таких системах небольшим значением.

Вообще говоря, теоретически, системы с нечеткой логикой и искусственные нейронные сети эквивалентны друг другу, однако на практике у них имеются свои собственные достоинства и недостатки. Данное соображение легло в основу аппарата гибридных сетей, где выводы делаются на основе аппарата нечеткой логики, но соответствующие функции принадлежности подстраиваются с использованием алгоритмов обучения нейронных сетей, например алгоритма обратного распространения ошибки. Такие системы не только используют априорную информацию, но могут приобретать новые знания и для пользователя являются логически прозрачными.

**Определение.** *Гибридная нейронная сеть* — это нейронная сеть с четкими сигналами, весами и активизационной функцией, но с объеди-

нением сигналов и весов сети с использованием Т-нормы, Т-конормы или некоторых других непрерывных операций.

Входы, выходы и веса гибридной нейронной сети — вещественные числа, принадлежащие отрезку [0, 1].

Примером подобной сети может служить система, имеющая следующую базу знаний:

$П_1$: если $x_1$ есть $L_1$ и $x_2$ есть $L_2$ и $x_3$ есть $L_3$, тогда $z$ *есть* $H$,

$П_2$: если $x_1$ есть $H_1$ и $x_2$ есть $H_2$ и $x_3$ есть $L_3$, тогда $z$ есть $M$,

$П_3$: если $x_1$ есть $H_1$ и $x_2$ есть $H_2$ и $x_3$ есть $H_3$, тогда $z$ есть $S$,

где $x_1$, $x_2$, $x_3$ — входные переменные, $z$ — выход системы, $L_1$, $L_2$, $L_3$, $H_1$, $H_2$, $H_3$, $H$, $M$, $S$ — некоторые нечеткие множества с функциями принадлежности сигмоидного типа:

$$L_j(t) = \frac{1}{1 + \exp(b_j(t - c_j))}, \quad H_j(t) = \frac{1}{1 + \exp(-b_j(t - c_j))}, \quad j = 1, 2, 3,$$

$$H(t) = \frac{1}{1 + \exp(-b_4(t - c_4 + c_5))}, \quad M(t) = \frac{1}{1 + \exp(-b_4(t - c_4))},$$

$$S(t) = \frac{1}{1 + \exp(b_4(t - c_4))}.$$

Для определения выходной переменной используется следующий алгоритм вывода:

1) подсчитываются значения истинности предпосылок для каждого правила:

$$\alpha_1 = L_1(a_1) \wedge L_2(a_2) \wedge L_3(a_3),$$

$$\alpha_2 = H_1(a_1) \wedge H_2(a_2) \wedge L_3(a_3),$$

$$\alpha_3 = H_1(a_1) \wedge H_2(a_2) \wedge H_3(a_3),$$

где в данном случае $a_1$, $a_2$, $a_3$ — текущие значения входов системы.

2) для каждого правила определяются частные выходы:

$$z_1 = B^{-1}(\alpha_1) = c_4 + c_5 + \frac{1}{b_4} \ln \frac{1 - \alpha_1}{\alpha_1},$$

$$z_2 = B^{-1}(\alpha_2) = c_4 + \frac{1}{b_4} \ln \frac{1 - \alpha_2}{\alpha_2},$$

$$z_3 = B^{-1}(\alpha_3) = c_4 + \frac{1}{b_4} \ln \frac{1 - \alpha_3}{\alpha_3}.$$

3) находится общий выход системы:

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3}{\alpha_1 + \alpha_2 + \alpha_3}.$$

Изложенный процесс иллюстрируется рис. 6.7, где оси абсцисс трех левых графиков соответствуют переменной $x_1$, трех следующих — переменной $x_2$, далее — $x_3$, а трех правых — переменной вывода $z$.

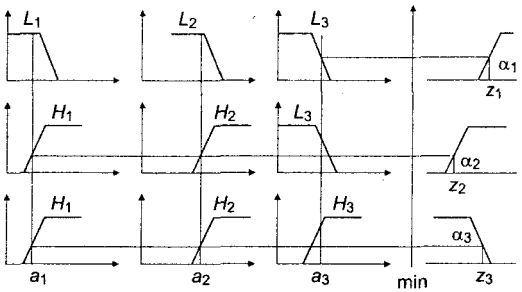


**Рис. 6.7.** Иллюстрация алгоритма вывода

Гибридная нейронная сеть, отражающая приведенный механизм вывода, представлена на рис. 6.8. Заметим, что сети с подобной архитектурой в англоязычной литературе получили название ANFIS (Adaptive Neuro-Fuzzy Inference System, то есть адаптивная нечеткая нейронная система вывода).

Данная сеть может быть описана следующим образом:

- **Слой 1** (Layer 1). Выходы узлов этого слоя представляют собой значения функций принадлежности при конкретных (заданных) значениях входов.
- **Слой 2** (Layer 2). Выходами нейронов этого слоя являются степени истинности предпосылок каждого правила базы знаний системы, вычисляемые по формулам

  $\alpha_1 = L_1(a_1) \wedge L_2(a_2) \wedge L_3(a_3)$,

  $\alpha_2 = H_1(a_1) \wedge H_2(a_2) \wedge L_3(a_3)$,

  $\alpha_3 = H_1(a_1) \wedge H_2(a_2) \wedge H_3(a_3)$.

  Все нейроны этого слоя обозначены буквой T, что означает, что они могут реализовывать произвольную T-норму для моделирования операции «И».
- **Слой 3** (Layer 3). Нейроны этого слоя (обозначены буквой N) вычисляют величины

$$\beta_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2 + \alpha_3}, \quad \beta_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2 + \alpha_3}, \quad \beta_3 = \frac{\alpha_3}{\alpha_1 + \alpha_2 + \alpha_3}.$$

- **Слой 4** (Layer 4). Нейроны данного слоя выполняют операции $\beta_1 z_1 = \beta_1 H^{-1}(a_1)$, $\beta_2 z_2 = \beta_2 M^{-1}(a_2)$, $\beta_3 z_3 = \beta_3 S^{-1}(a_3)$.
- **Слой 5** (Layer 5). Единственный нейрон этого слоя вычисляет выход сети:

$$z_0 = \beta_1 z_1 + \beta_2 z_2 + \beta_3 z_3.$$

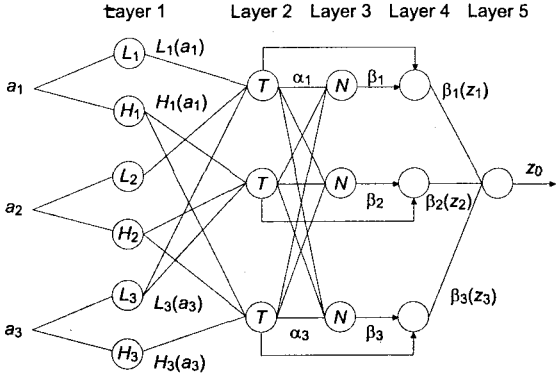


**Рис. 6.8.** Структура гибридной нейронной сети (архитектура ANFIS)

Корректировка параметров системы здесь производится либо в соответствии с наиболее распространенным для нейронных сетей алгоритмом обратного распространения ошибки (back propagation), либо комбинированным методом, специально разработанным для гибридных сетей.

# Графический интерфейс Fuzzy Logic Toolbox

## Состав графического интерфейса

В состав программных средств Fuzzy Logic Toolbox входят следующие основные программы, позволяющие работать в режиме графического интерфейса:

- редактор нечеткой системы вывода Fuzzy Inference System Editor (FIS Editor или FIS-редактор) вместе со вспомогательными программами — редактором функций принадлежности (Membership Function Editor), редактором правил (Rule Editor), просмотрщи-

ком правил (Rule Viewer) и просмотрщиком поверхности отклика (Surface Viewer);

- редактор гибридных систем (ANFIS Editor, ANFIS-редактор);
- программа нахождения центров кластеров (программа Clustering — кластеризация).

Набор данных программ предоставляет пользователю максимальные удобства для создания, редактирования и использования различных систем нечеткого вывода.

## Построение нечеткой аппроксимирующей системы

Командой (функцией) Fuzzy из режима командной строки запускается основная интерфейсная программа пакета Fuzzy Logic — редактор нечеткой системы вывода. Вид открывающегося при этом окна приведен на рис. 6.9.

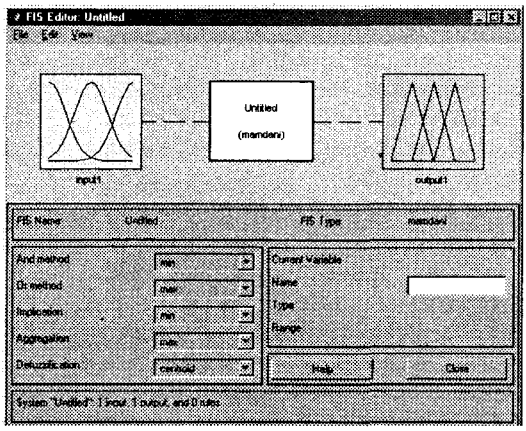


**Рис. 6.9.** Вид окна FIS Editor

Строка меню редактора содержит следующие позиции:

- File — работа с файлами моделей (их создание, сохранение, считывание и печать).
- Edit — операции редактирования (добавление и исключение входных и выходных переменных).
- View — переход к дополнительному инструментарию.

Как говорят англичане, чтобы узнать вкус пудинга, надо его съесть, поэтому представляется целесообразным изучить различные опции

и возможности данного редактора и связанных с ним других программ на каком-либо конкретном примере.

Попробуем сконструировать нечеткую систему, отображающую зависимость между переменными $x$ и $y$, заданную с помощью табл. 6.2 (*легко видеть, что представленные в таблице данные отражают зависимость* $y = x^2$).

**Таблица 6.2.** Значения x и y

| | | | | | |
|---|---|---|---|---|---|
| $x$ | −1 | −0,6 | 0 | 0,4 | 1 |
| $y$ | 1 | 0,36 | 0 | 0,16 | 1 |

Требуемые действия отобразим следующими пунктами.

1. В меню File выбираем команду New Sugeno FIS (Новая система типа Sugeno), при этом в блоке, отображаемом белым квадратом в верхней части окна редактора, появится надпись Untitled2 (sugeno).
2. Щелкнем на блоке, озаглавленном input1 (Вход1). Затем в правой части редактора в поле Name (Имя) вместо input1 введем обозначение нашего аргумента, то есть x. Обратим внимание, что если теперь сделать где-нибудь (вне блоков редактора) однократный щелчок, то имя отмеченного блока изменится на x; то же достигается нажатием клавиши Enter после ввода.
3. Дважды щелкнем на этом блоке. Перед нами откроется окно редактора функций принадлежности — Membership Function Editor (рис. 6.10). Откроем меню Edit данного редактора и выберем в нем команду Add MFs (Add Membership Functions — Добавить функции принадлежности). При этом появится диалоговое окно (рис. 6.11), позволяющее задать тип (MF type) и количество (Number of MFs) функций принадлежности (в данном случае все относится к входному сигналу, то есть к переменной $x$). Выберем гауссовы функции принадлежности (gaussmf), а их количество зададим равным пяти — по числу значений аргумента в табл. 6.2. Подтвердим ввод информации нажатием кнопки OK, после чего произойдет возврат *к окну редактора функций принадлежности.*
4. В поле Range (Диапазон) установим диапазон изменения $x$ от −1 до 1, то есть диапазон, соответствующий табл. 6.2. Щелкнем затем левой кнопкой мыши где-нибудь в поле редактора (или нажмем клавишу Enter). Обратим внимание, что после этого произойдет соответствующее изменение диапазона в поле Display Range (Диапазон отображения).

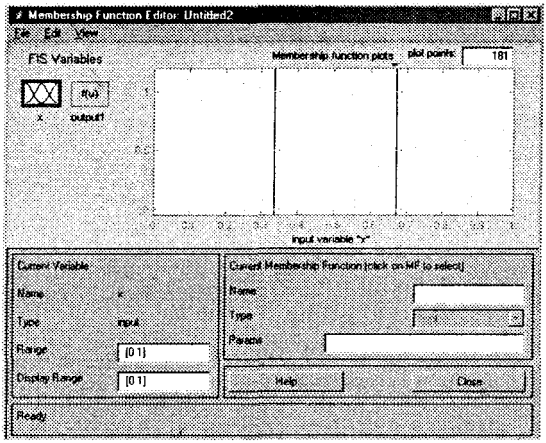


**Рис. 6.10.** Окно редактора функций принадлежности

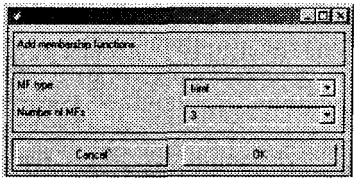


**Рис. 6.11.** Диалоговое окно задания типа и количества функций принадлежности

5. Обратимся к графикам заданных нами функций принадлежности, изображенным в верхней части окна редактора функций принадлежности. Заметим, что для успешного решения поставленной задачи необходимо, чтобы ординаты максимумов этих функций совпадали с заданными значениями аргумента $x$. Для левой, центральной и правой функций такое условие выполнено, но две других необходимо «подвинуть» вдоль оси абсцисс. Это делается весьма просто: подводим указатель к нужной кривой и щелкаем левой кнопкой мыши. Кривая выбирается, окрашиваясь в красный цвет, после чего с помощью мыши ее можно подвинуть в нужную сторону (более точную установку можно провести, изменяя числовые значения в поле Params (Параметры), — в данном случае каждой функции принадлежности соответствуют два параметра, при этом первый определяет размах кривой, а второй — положение ее центра). Помимо этого в поле Name можно изменить имя для выбранной кривой (завершая ввод каждого имени

нажатием клавиши Enter). Проделаем требуемые перемещения кривых и зададим всем пяти кривым новые имена, например, так:

- самой левой — bn,
- следующей — n,
- центральной — z,
- следующей за ней справа — p,
- самой правой — bp.

Нажмем кнопку Close и выйдем из редактора функций принадлежности, возвратившись при этом в окно редактора нечеткой системы (FIS Editor).

6. Сделаем однократный щелчок на голубом квадрате (блоке), озаглавленном output1 (Выход1). В поле Name заменим имя output1 на y (как в п. 2).
7. Дважды щелкнем на выделенном блоке и перейдем к редактору функций принадлежности. В меню Edit выберем команду Add MFs. Появляющееся затем диалоговое окно вида рис. 6.11 позволяет теперь задать в качестве функций принадлежности только линейные (linear) или постоянные (constant) — в зависимости от того, какой алгоритм Сугэно (1-го или 0-го порядка) мы выбираем. В рассматриваемой задаче необходимо выбрать постоянные функции принадлежности с общим числом 4 (по числу различных значений $y$ в табл. 6.2). Подтвердим введенные данные нажатием кнопки OK, после чего произойдет возврат в окно редактора функций принадлежности.
8. Обратим внимание, что диапазон изменения (Range), устанавливаемый по умолчанию — [0, 1], — менять в данном случае не нужно. Изменим лишь имена функций принадлежности (их графики при использовании алгоритма Сугэно для выходных переменных не приводятся), например, задав их как соответствующие числовые значения $y$, то есть 0, 0.16, 0.36, 1; одновременно эти же числовые значения введем в поле Params (рис. 6.12). Затем закроем окно нажатием кнопки Close и вернемся в окно FIS-редактора.
9. Дважды щелкнем на среднем (белом) блоке, при этом раскроется окно еще одной программы — редактора правил (Rule Editor). Введем соответствующие правила. При вводе каждого правила необходимо обозначить соответствие между каждой функцией принадлежности аргумента $x$ и числовым значением $y$. Кривая, обозначенная нами bn, соответствует $x = -1$, то есть $y = 1$. Выберем поэтому в левом поле (с заголовком x is) вариант bn, а в пра-

вом 1 и нажмем кнопку Add rule (Добавить правило). Введенное правило появится в окне правил и будет представлять собой такую запись: 1. If (x is bn) then (y is 1) (1). Аналогично поступим для всех других значений $x$, в результате чего сформируется набор из пяти правил (рис. 6.13). Закроем окно редактора правил и возвратимся в окно FIS-редактора. Построение системы закончено, и можно начать эксперименты по ее исследованию. Заметим, что для большинства опций были сохранены значения по умолчанию.

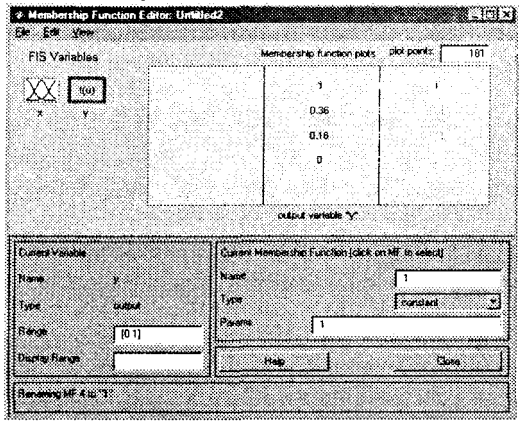


**Рис. 6.12.** Параметры функций принадлежности переменной y

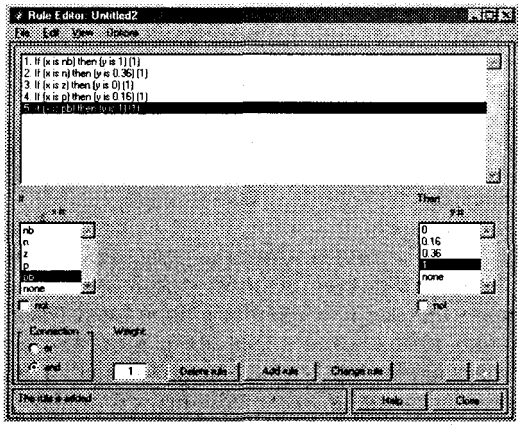


**Рис. 6.13.** Окно редактора правил

10. Предварительно сохраним на диске (используя команды меню File ▸ Save to disk as) созданную систему под каким-либо именем, например Proba.

    Раскроем меню View. С помощью его команд Edit membership functions и Edit rules можно совершить переход к двум рассмотренным ранее программам — редакторам функций принадлежности и правил (то же можно сделать нажатием клавиш Ctrl+2 и Ctrl+3), но сейчас нас будут интересовать две других команды — View rules (Просмотр правил) и View surface (Просмотр поверхности). Выберем команду View rules, при этом откроется окно (рис. 6.14) еще одной программы — просмотра правил (Rule Viewer).

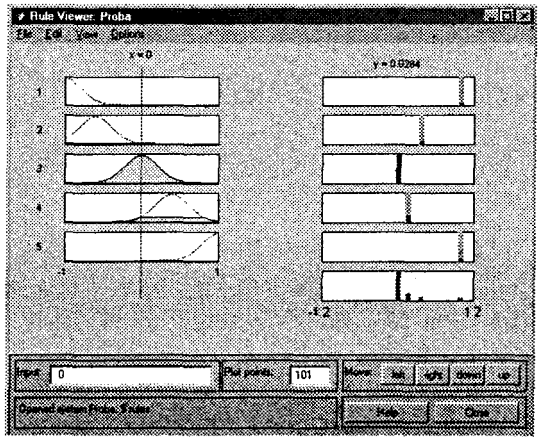


**Рис. 6.14.** Окно просмотра правил

11. В правой части окна в графической форме представлены функции принадлежности аргумента $x$, в левой — функции принадлежности переменной выхода $y$ с пояснением механизма принятия решения. Красная вертикальная черта, пересекающая графики в правой части окна, которую можно перемещать с помощью мыши, позволяет изменять значения переменной входа (это же можно делать, задавая числовые значения в поле Input (Вход)), при этом соответственно изменяются значения $y$ в правой верхней части окна. Зададим, например, $x = 0{,}5$ в поле Input и нажмем затем клавишу Enter. Значение $y$ сразу изменится и станет равным 0,202. Таким образом, с помощью построенной модели и окна просмотра правил можно решать задачу интерполяции, то есть задачу,

решение которой и требовалось найти. Изменение аргумента путем перемещения красной вертикальной линии очень наглядно демонстрирует, как система определяет значения выхода.

12. Закроем окно просмотра правил и выбором команды меню View ▸ View surface перейдем к окну просмотра поверхности отклика (выхода), в нашем случае — к просмотру кривой $y(x)$ (рис. 6.15). Видно, что смоделированное системой по таблице данных (см. табл. 6.2) отображение не очень-то напоминает функцию $x^2$ . Ну что ж, ничего удивительного в этом нет: число экспериментальных точек невелико, да и параметры функций принадлежности (для $x$) выбраны, скорее всего, неоптимальным образом. Ниже мы рассмотрим возможность улучшения качества подобной модели.

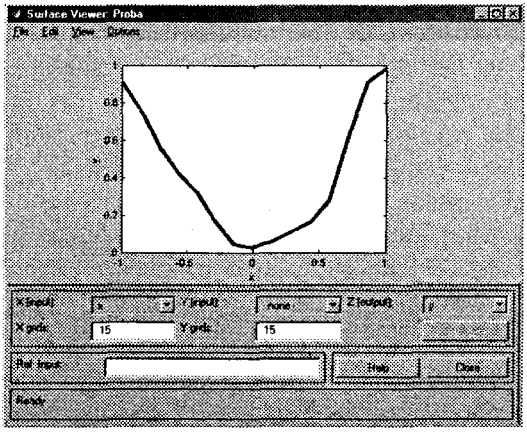


**Рис. 6.15.** Окно просмотра поверхности отклика

В заключение рассмотрения примера отметим, что с помощью вышеуказанных программ-редакторов на любом этапе проектирования нечеткой модели в нее можно внести необходимые коррективы, вплоть до задания какой-либо особенной пользовательской функции принадлежности. Из опций, устанавливаемых в FIS-редакторе по умолчанию при использовании алгоритма Сугэно, можно отметить следующие:

- логический вывод организуется с помощью операции умножения (prod);
- композиция организуется с помощью операции логической суммы (вероятностного ИЛИ, probor);

- приведение к четкости организуется дискретным вариантом центроидного метода (взвешенным средним, wtaver).

Используя соответствующие поля в левой нижней части окна FIS-редактора, данные опции можно при желании изменить.

## Построение экспертной системы: сколько дать на чай?

Рассмотрим теперь методику построения нечеткой экспертной системы, которая должна помочь пользователю с ответом на вопрос: сколько дать на чай официанту за обслуживание в ресторане? (Предположим, речь идет о местах, где такие чаевые принято давать, например, в ресторанах Парижа или Рио-де-Жанейро).

Основываясь на каких-то устоявшихся обычаях и интуитивных представлениях, примем, что задача о чаевых может быть описана следующими предложениями.

1. Если обслуживание плохое или еда подгоревшая, то чаевые — малые.
2. Если обслуживание хорошее, то чаевые — средние.
3. Если обслуживание отличное или еда превосходная, то чаевые — щедрые.

Качество обслуживания и еды будем оценивать по 10-балльной системе (0 — наихудшая оценка, 10 — наилучшая).

Будем предполагать далее, что малые чаевые составляют около 5 % от стоимости обеда, средние — около 15 % и щедрые — примерно 25 %.

Заметим, что представленной информации, в принципе, достаточно для проектирования нечеткой экспертной системы. Такая система будет иметь 2 входа (которые условно можно назвать «сервис» и «еда»), один выход («чаевые»), три правила типа «если... то» (в соответствии с тремя приведенными предложениями) и по три значения (соответственно, 0 баллов, 5 баллов, 10 баллов и 5 %, 15 %, 25 %) для центров функций принадлежности входов и выхода. Построим данную систему, используя алгоритм вывода Мамдани и, как в предыдущем примере, описывая требуемые действия по пунктам.

1. Командой fuzzy запускаем FIS-редактор. По умолчанию предлагается алгоритм вывода типа Мамдани (о чем говорит надпись в центральном белом блоке), и здесь никаких изменений не требу-

ется, но в системе должно быть два входа, поэтому через пункт меню Edit ▸ Add input добавляем в систему этот второй вход (в окне редактора появляется второй желтый блок с именем input2). Делая далее однократный щелчок на блоке input1, меняем его имя на «сервис», завершая ввод нового имени нажатием клавиши Enter. Аналогичным образом устанавливаем имя «еда» блоку input2 и «чаевые» — выходному блоку (справа вверху) output1. Присвоим сразу же и имя всей системе, например «tip» (по-английски это и есть чаевые), выполнив это через пункт меню File ▸ Save to workspace as (Сохранить в рабочем пространстве как...). Вид окна редактора после указанных действий приведен на рис. 6.16.

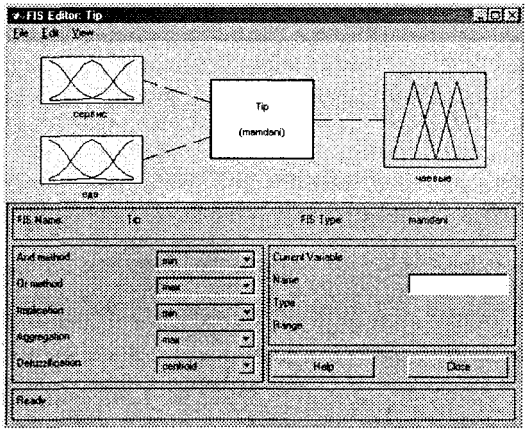


**Рис. 6.16.** Вид окна FIS-редактора после задания структуры системы

2. Зададим теперь функции принадлежности переменных. Напомним еще раз, что программу-редактор функций принадлежности можно открыть тремя способами:
   - через пункт меню View ▸ Edit membership functions,
   - двойным щелчком на значке, отображающем соответствующую переменную,
   - нажатием клавиш Ctrl+2.

   Любым из приведенных способов перейдем к данной программе.

   Задание и редактирование функций принадлежности начнем с переменной «сервис». Сначала в полях Range и Display Range установим диапазон изменения и отображения этой переменной — от

0 до 10 (баллов), подтверждая ввод нажатием клавиши Enter. Затем через пункт меню Edit/Add MFs перейдем к диалоговому окну вида рис. 6.12 и зададим в нем три функции принадлежности гауссова типа (gaussmf). Нажмем кнопку OK и возвратимся в окно редактора функций принадлежности. Не изменяя размах и положение заданных функций, заменим только их имена на «плохой», «хороший» и «отличный» (как в п. 5 предыдущего примера).

Щелчком на значке «еда» войдем в окно редактирования функций принадлежности для этой переменной. Зададим сначала диапазон ее изменения от 0 до 10, а затем, поступая как ранее, зададим две функции принадлежности трапецеидальной формы с параметрами, соответственно, [0 0 1 3] и [7 9 10 10] и именами «подгоревшая» и «превосходная».

Для выходной переменной «чаевые» укажем сначала диапазон изменения (от 0 до 30), потом зададим три функции принадлежности треугольной формы с именами «малые», «средние» и «щедрые» так, как это представлено на рис. 6.17. Заметим, что можно, разумеется, задать и какие-либо другие функции или выбрать другие их параметры.

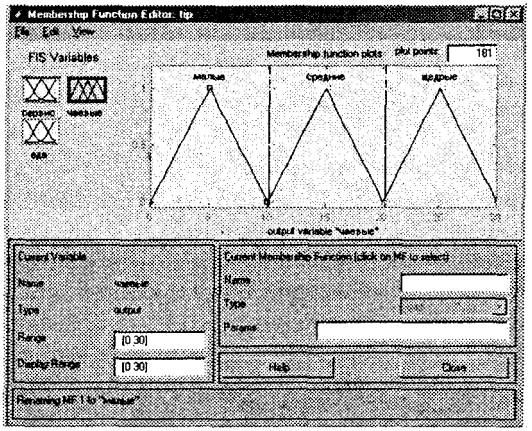


**Рис. 6.17.** Функции принадлежности переменной «чаевые»

Перейдем к конструированию правил. Для этого выберем пункт меню View ▸ Edit rules. Далее ввод правил производится так же, как в п. 9 предыдущего примера, и в соответствии с предложениями, описывающими задачу. Заметим, что в первом и третьем

правилах в качестве «связки» в предпосылках правила необходимо использовать не «И» (and), а «ИЛИ» (or); при вводе второго правила, где отсутствует переменная «еда», для нее выбирается опция none. Итоговый набор правил отображен на рис. 6.18.

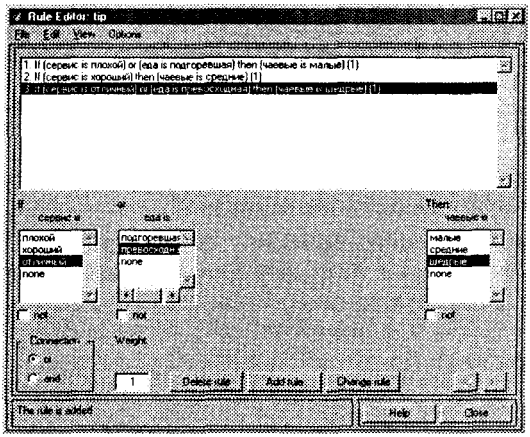


**Рис. 6.18.** Итоговый набор правил в задаче о чаевых



Такая подробная (verbose) запись представляется достаточно понятной; единица в скобках после каждого правила указывает его «вес» (Weight), то есть значимость правила. Данный вес можно менять, используя соответствующее поле в левой нижней части окна редактора правил. Правила представимы и в других формах — символической (symbolic) и индексной (indexed), — при этом переход от одной формы к другой происходит с помощью меню Options ▸ Format редактора правил. Вот как выглядят рассмотренные правила в символической форме:

- 1. (сервис==плохой)|(еда==подгоревшая)=>(чаевые=малые) (1)
- 2. (сервис==хороший)=>(чаевые=средние) (1)
- 3. (сервис==отличный)|(еда==превосходная)=>(чаевые=щедрые) (1)

По-видимому, здесь тоже понятно все.

Наконец, самый сжатый формат представления правил — индексный — является тем форматом, который в действительности используется программой. В этом формате приведенные правила выглядят так:

- 1 1, 1 (1): 2
- 2 0, 2 (1): 2
- 3 2, 3 (1): 2

Здесь первая колонка относится к первой входной переменной (соответственно, первое, второе или третье возможное значение), вторая — ко второй, третья (после запятой) — к выходной переменной, цифра в скобках показывает вес правила и последняя цифра (после двоеточия) указывает тип «связки» (1 для «И», 2 для «ИЛИ»).

На этом, собственно, конструирование экспертной системы закончено. Сохраним ее на диске под выбранным именем (tip).

3. Теперь самое время проверить систему в действии. Откроем (через пункт меню View ▸ View rules) окно просмотра правил и установим значения переменных: сервис = 0 (то есть никуда не годный), еда = 10 (то есть превосходная). Увидим ответ: чаевые = 15 (то есть средние). Ну что ж, с системой не поспоришь, надо платить (рис. 6.19).

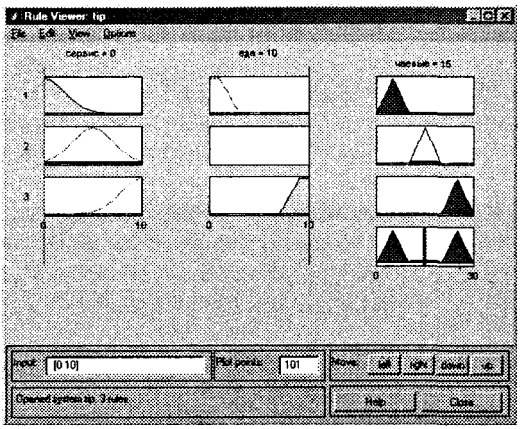


**Рис. 6.19.** Окно просмотра правил в задаче о чаевых

Можно проверить и другие варианты. В частности (может быть, не без удивления), выяснится, что нашей системой обслуживание ценится больше, чем качество еды: при наборе «сервис = 10, еда = 3» система советует определить размер чаевых в 23.9 %, в то время как набору «сервис = 3, еда = 10» размер чаевых по рекомендации систе-

мы — 16.6 % (от стоимости обеда). Впрочем, ничего удивительного здесь нет: это мы сами (не особенно подозревая об этом) заложили в систему соответствующие знания в виде совокупности приведенных правил.

Подтверждением отмеченной зависимости выходной переменной от входных может служить вид поверхности отклика, который представляется при выборе пункта меню View/View surface (рис. 6.20); обратите внимание, что с помощью мышки график можно поворачивать во все стороны.

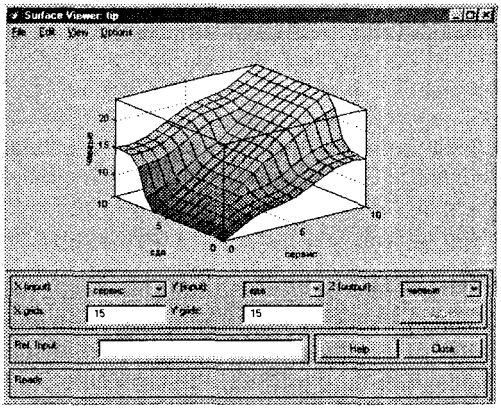


**Рис. 6.20.** Графический вид зависимости выходной переменной от входных

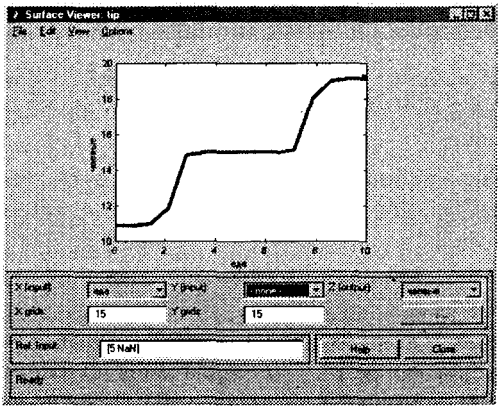


**Рис. 6.21.** Одномерная зависимость размера чаевых от качества еды

В открывшемся окне, меняя имена переменных в полях ввода (X (input) и Y (input)), можно задать и просмотр одномерных зависимостей, например «чаевых» от «еды» (рис. 6.21).

## Экспорт и импорт результатов

Когда вы сохраняете созданную вами нечеткую систему, используя пункты меню File ▸ Save to disk или File ▸ Save to disk as, на диске создается текстовый (ASCII) файл достаточно простого формата с расширением .fis. Его можно просматривать, при необходимости редактировать вне системы MATLAB, а также использовать повторно при последующих сеансах работы с системой. Однако сохранение с использованием пунктов File ▸ Save to workspace или File ▸ Save to workspace as на самом деле только «легализует» созданную вами систему (под каким-либо именем) в среде MATLAB в течение текущего сеанса работы и не допускает ее повторного использования в других сеансах.

## Создание пользовательских функций принадлежности

Если по каким-либо причинам вас не устраивает ни одна из встроенных функций принадлежности, вы можете создать и использовать собственную подходящую функцию. Такая функция должна быть создана как m-файл, возвращать значения в диапазоне от 0 до 1 и иметь число аргументов не более 16. Приведем этапы создания данной функции под некоторым именем custmf.

1. Создается соответствующий m-файл с именем custmf.m.
2. Выбирается пункт меню Edit ▸ Add custom MF (Добавить пользовательскую функцию принадлежности) в меню редактора функций принадлежности.
3. В поле M-File function name появившегося диалогового окна Add customized membership function вводится имя созданного m-файла (custmf).
4. В поле Parameter list данного окна вводятся необходимые числовые параметры.
5. Наконец, в поле MF name (Имя функции принадлежности) вводится какое-либо (уникальное) имя задаваемой функции (например, custmf).
6. Указанный ввод подтверждается нажатием кнопки OK (рис. 6.22).

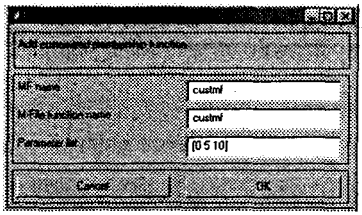


**Рис. 6.22.** Окно задания функции принадлежности пользователя

Ниже приведен пример m-файла некоторой пользовательской функции принадлежности дискретного типа, имеющей имя testmf1 и зависящей от 8 числовых параметров (каждый — из диапазона [0, 10]).

```
function out = testmf1(x, params)
for i=1:length(x)
   if x(i)<params(1)
      y(i)=params(1);
   elseif x(i)<params(2)
      y(i)=params(2);
   elseif x(i)<params(3)
      y(i)=params(3);
   elseif x(i)<params(4)
      y(i)=params(4);
   elseif x(i)<params(5)
      y(i)=params(5);
   elseif x(i)<params(6)
      y(i)=params(6);
   elseif x(i)<params(7)
      y(i)=params(7);
   elseif x(i)<params(8)
      y(i)=params(8);
   else
      y(i)=0;
   end
end
out=.1*y';
```

# Графический интерфейс гибридных систем

Графический интерфейс гибридных (нечетких) нейронных систем вызывается функцией anfisedit (из режима командной строки). Исполнение функции приводит к появлению окна редактора гибрид-

ных систем (ANFIS Editor, ANFIS-редактор), вид которого приведен на рис. 6.23.

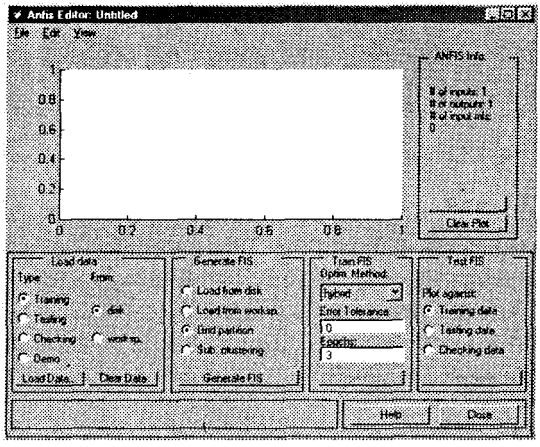


**Рис 6.23.** Окно редактора гибридных систем

С помощью данного редактора осуществляются создание или загрузка структуры гибридной системы, просмотр структуры, настройка ее параметров, проверка качества функционирования такой системы. Создание структуры, настройка параметров и проверка осуществляются по выборкам (наборам данных) — обучающей (Training data), проверочной (Checking data) и тестирующей (Testing data), которые предварительно должны быть представлены в виде текстовых файлов (с расширением .dat и разделителями-табуляциями), первые столбцы которых соответствуют входным переменным, а последний (правый) — единственной выходной переменной; количество строк в таких файлах равно количеству образцов (примеров). Так, обучающая выборка, сформированная по табл. 6.2, представляется в виде

```
-1 1
-0.6  0.36
0.00.00
0.40.16
1  1
```

Строгих рекомендаций по объемам указанных выборок не существует; по-видимому, лучше всего исходить из принципа «чем больше, тем лучше». Обучающая и проверочная выборки непосредственно

задействуются в процессе настройки параметров гибридной сети (проверочная — для выяснения ситуации, не происходит ли так называемого *переобучения* сети, при котором ошибка для обучающей последовательности стремится к нулю, а для проверочной — возрастает; впрочем, наличие проверочной выборки не является строго необходимым, оно лишь крайне желательно). Тестовая (или тестирующая) выборка применяется для проверки качества функционирования настроенной (обученной) сети.

Поясним пункты меню и опции редактора.

Меню File и View в общем идентичны аналогичным меню FIS-редактора за тем исключением, что здесь работа может происходить только с алгоритмом нечеткого вывода Sugeno. Меню Edit содержит единственную команду — Undo (Отменить выполненное действие).

Набор опций Load data (Загрузка данных) в нижней левой части окна редактора включает в себя:

- тип (Type) загружаемых данных (для обучения — Training, для тестирования — Testing, для проверки — Checking, демонстрационные — Demo);
- место, откуда должны загружаться данные: с диска (disk) или из рабочей области MATLAB (workspace).

К данным опциям относятся две кнопки, нажатие на которые приводит к требуемым действиям — Load Data (Загрузить данные) и Clear Data (Стереть данные).

Следующая группа опций (в середине нижней части окна ANFIS-редактора) объединена под именем Generate FIS (Создание нечеткой системы вывода). Данная группа включает в себя следующие опции:

- загрузку структуры системы с диска (Load from disk);
- загрузку структуры системы из рабочей области MATLAB (Load from worksp.);
- разбиение (деление) областей определения входных переменных (аргументов) на подобласти — независимо для каждого аргумента (Grid partition);
- разбиение всей области определения аргументов (входных переменных) на подобласти — в комплексе для всех аргументов (Sub. clustering),

Кроме того, имеется также кнопка Generate FIS, нажатие которой приводит к процессу создания гибридной системы с точностью до ряда параметров.

Следующая группа опций — Train FIS (Обучение нечеткой системы вывода) — позволяет определить метод «обучения» (Optim. Method) системы (то есть метод настройки ее параметров) — гибридный (hybrid) или обратного распространения ошибки (backpropa), установить уровень текущей суммарной (по всем образцам) ошибки обучения (Error Tolerance), при достижении которого процесс обучения заканчивается, и количество циклов обучения (Epochs), то есть количество «прогонов» всех образцов (или примеров) обучающей выборки; процесс обучения, таким образом, заканчивается либо при достижении отмеченного уровня ошибки обучения, либо после проведения заданного количества циклов.

Кнопка Train Now (Начать обучение) запускает процесс обучения, то есть процесс настройки параметров гибридной сети.

В правом верхнем углу окна ANFIS-редактора выдается информация (ANFIS Info.) о проектируемой системе: количество входов, выходов, функций принадлежности входов; нажатие кнопки Structure (Структура) позволяет увидеть структуру сети в виде, аналогичном представленному на рис. 6.8. Кнопка Clear (Очистить) позволяет стереть все результаты.



Опции Test FIS в правом нижнем углу окна позволяют провести проверку и тестирование созданной и обученной системы с выводом результатов в виде графиков (соответствующие графики для обучающей выборки — Training data, тестирующей выборки — Testing data и проверочной выборки — Checking data). Кнопка Test Now позволяет запустить указанные процессы.

Работу с редактором рассмотрим на примере восстановления зависимости $y = x^2$ по данным табл. 6.2. Предположим, что эти данные сохранены в файле Proba.dat. Создание и проверку системы, как и раньше, проведем по этапам:

1. В окне ANFIS-редактора выберем тип загружаемых данных Training и нажмем кнопку Load data. В последующем стандартном окне диалога укажем местоположение и имя файла. Его открытие приводит к появлению в графической части окна редактора набора точек, соответствующих введенным данным (рис. 6.24).
2. В группе опций Generate FIS по умолчанию активизирован вариант Grid partition. Не будем его изменять и нажмем кнопку Generate FIS, после чего появится диалоговое окно (рис. 6.25) для задания числа и типов функций принадлежности. Сохраним все установки по умолчанию, согласившись с ними нажатием кнопки OK. Произойдет возврат в основное окно ANFIS-редактора. Теперь струк-

тура гибридной сети создана, и ее графический вид можно просмотреть с помощью кнопки Structure (рис. 6.25).

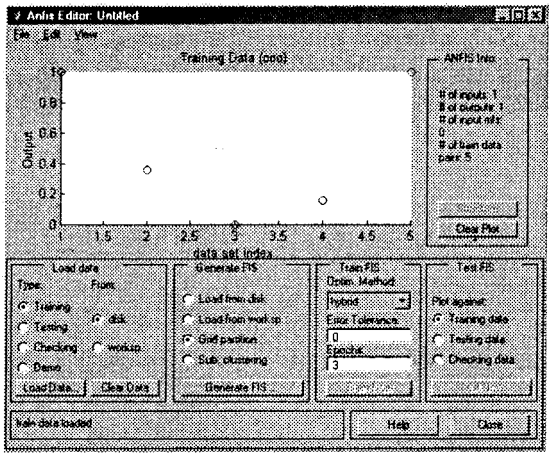


**Рис. 6.24.** Окно ANFIS-редактора после загрузки обучающей выборки

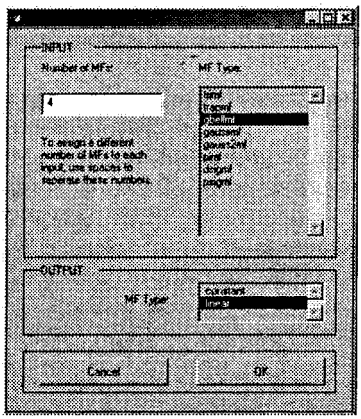


**Рис. 6.25.** Окно задания функций принадлежности

3. Перейдем к опциям Train FIS. Не будем менять задаваемые по умолчанию метод настройки параметров (hybrid — гибридный) и уровень ошибки (0), но количество циклов обучения изменим на 40, после чего нажмем кнопку запуска процесса обучения (Train

Now). Получившийся результат в виде графика ошибки сети в зависимости от числа проведенных циклов обучения (из которого следует, что фактически обучение закончилось после пятого цикла) представлен на рис. 6.27.

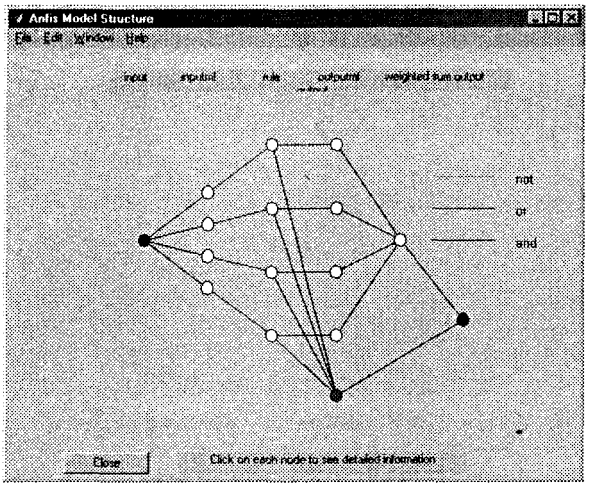


**Рис. 6.26.** Структура созданной гибридной сети

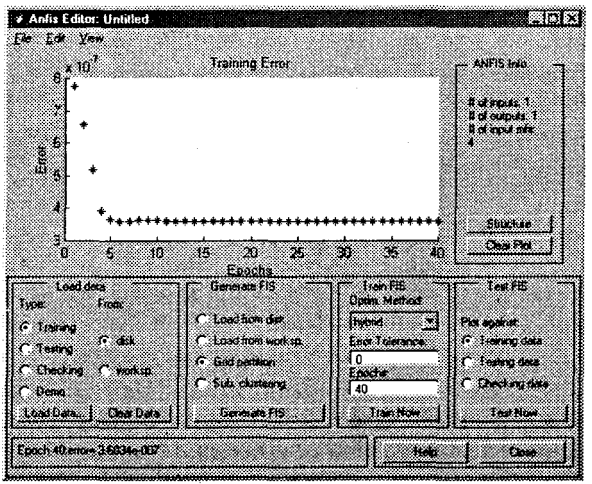


**Рис. 6.27.** Результат обучения сети

4. Теперь нажатием кнопки Test Now можно начать процесс тестирования обученной сети, но поскольку использовалась только одна (обучающая) выборка, ничего особенно интересного ожидать не приходится. Действительно, выход обученной системы практически совпадает с точками обучающей выборки (рис. 6.28).

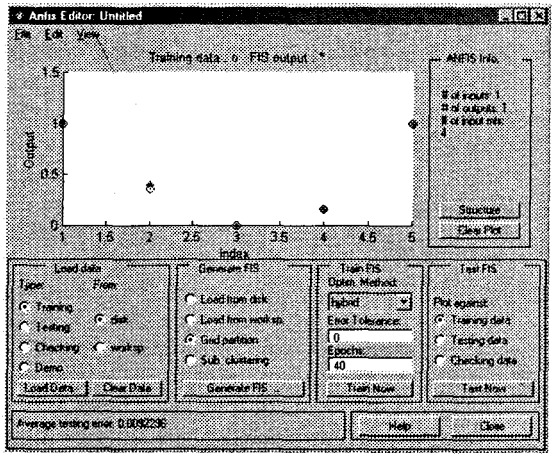


**Рис. 6.28.** Результат тестирования обученной системы

5. Сохраним разработанную систему на диске в файл с именем Proba1 (с расширением .fis) и для исследования разработанной системы средствами FIS-редактора из командной строки MATLAB выполним команду fuzzy, а затем через пункты меню File | Open FIS from disk откроем созданный файл. С созданной системой можно теперь выполнять все приемы редактирования (изменение имен переменных и т. п.) и исследования, которые были рассмотрены выше. Здесь нетрудно, кстати, убедиться, что качество аппроксимации существенно не улучшилось — слишком мало данных.

Что можно сказать про эффективность использования гибридных систем (и ANFIS-редактора)?

В данном случае используется только один алгоритм нечеткого вывода — Sugeno (нулевого или первого порядков), может быть только одна выходная переменная, всем правилам приписывается один и тот же единичный вес. Вообще говоря, возникают значительные проблемы при большом (более 5–6) количестве входных переменных. Это — ограничения и недостатки подхода.

Его несомненные достоинства: практически полная автоматизация процесса создания нечеткой (гибридной) системы, возможность просмотра сформированных правил и придания им содержательной (лингвистической) интерпретации, что позволяет, кстати говоря, рассматривать аппарат гибридных сетей как средство извлечения знаний из баз данных и существенно отличает данные сети от классических нейронных.

Рекомендуемая область применения: построение аппроксиматоров зависимостей по экспериментальным данным, построение систем классификации (в случае бинарной или дискретной выходной переменной), изучение механизма явлений.

## Графический интерфейс программы кластеризации

В пакет Fuzzy Logic Toolbox входит еще одна программа, позволяющая работу в режиме графического интерфейса — программа Clustering (Кластеризация) выявления центров кластеров, то есть точек в многомерном пространстве данных, около которых группируются (скапливаются) экспериментальные данные. Выявление подобных центров, надо сказать, является значимым этапом при предварительной обработке данных, поскольку позволяет сопоставить с этими центрами функции принадлежности переменных при последующем проектировании системы нечеткого вывода.

Запуск программы Clustering осуществляется командой (функцией) findcluster. В появляющемся окне программы имеется (вверху) главное меню, содержащее достаточно стандартный набор пунктов (File, Edit, Window, Help) и набор управляющих кнопок и опций (справа). К этом кнопкам относятся:

- кнопка загрузки файла данных — Load Data;
- кнопка выбора алгоритма кластеризации — Method;
- четыре расположенных ниже кнопок опций алгоритма (их названия меняются в зависимости от выбранного алгоритма);
- кнопка начала итеративного процесса нахождения центров кластеров (кластеризации) — Start;
- кнопка сохранения результатов кластеризации — Save Center;
- кнопка очистки (стирания) графиков — Clear Plot;
- кнопка справочной информации — Info;
- кнопка завершения работы с программой — Close.

В программе используются два алгоритма выявления центров кластеров: Fuzzy c-means (который можно перевести как «Алгоритм нечетких центров») и Subtractive clustering («Вычитающая кластеризация»). Если не вдаваться в их детальное теоретическое изложение, а ограничиться выявлением различий на уровне пользователя, то можно отметить, что алгоритм Fuzzy c-means, являясь, пожалуй, более точным (если понятие точности вообще здесь применимо), для своей работы требует задания таких опций, как число кластеров (кнопка Cluster num.) и число итераций (кнопка Max Iteration#). Ну, если число итераций еще можно задать как-то наугад, то ошибка в задании числа кластеров может впоследствии привести к неприятным последствиям. Алгоритм Subtractive clustering менее точен, но и менее требователен к априорной информации; при работе с ним можно сохранить опции, заданные в программе по умолчанию. На рис. 6.29 приведен пример использования программы для файла данных clusterdemo.dat из директории Matlab/toolbox/fuzzy/fuzdemos/ при использовании алгоритма Subtractive clustering. Заметим, что выводится только двумерное поле рассеяния, но изменяя переменные в соответствующих полях (X-axis и Y-axis), можно «просмотреть» все многомерное пространство переменных.



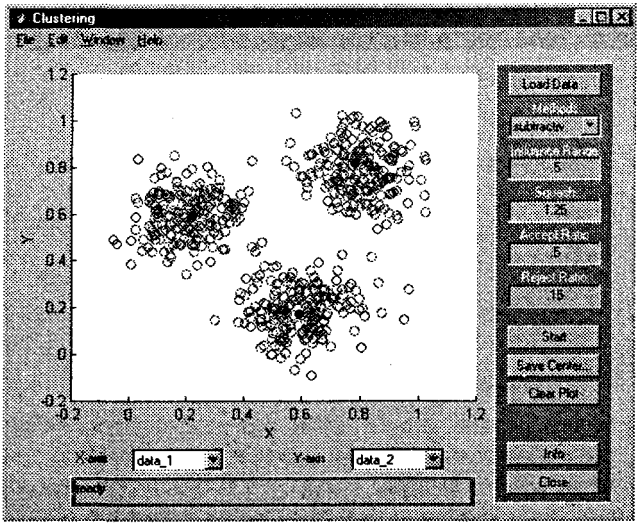


**Рис. 6.29.** Результат работы программы Clustering (центры кластеров окрашены в черный цвет)

# Работа с Fuzzy Logic Toolbox в режиме командной строки

## Возможности работы в режиме командной строки

Пакет Fuzzy Logic располагает большим набором функций, исполняемых из командной строки MATLAB и позволяющих, в принципе, не использовать при работе с системами нечеткого вывода рассмотренные программы графического интерфейса. Все функции делятся на группы:

1) вызова программ графического интерфейса;
2) задания функций принадлежности;
3) создания, редактирования, просмотра, открытия и сохранения систем нечеткого вывода;
4) дополнительные;
5) сервисные;
6) вызова диалоговых окон интерфейса;
7) блоков Simulink;
8) демонстрации возможностей пакета.



## Функции вызова программ графического интерфейса

К этой группе относятся следующие функции:

- `fuzzy` — вызов FIS-редактора;
- `mfedit` — вызов редактора функций принадлежности;
- `ruleedit` — вызов редактора правил;
- `ruleview` — вызов программы просмотра правил;
- `surfview` — вызов программы просмотра поверхности отклика;
- `anfisedit` — вызов ANFIS-редактора (только для систем, использующих алгоритм Sugeno и имеющих одну выходную переменную);
- `findcluster` — вызов программы кластеризации.

Использование первых шести функций с аргументом (например, `fuzzy(a)`, где `a` — имя переменной рабочего пространства, присвоенное системе нечеткого вывода) открывает соответствующую программу с одновременной загрузкой в нее рассматриваемой системы.

Функция `findcluster(имя_файла)` открывает программу кластеризации с одновременной загрузкой указанного файла данных.

## Задание функций принадлежности

В данную группу включены 11 функций (по числу функций принадлежности, используемых в пакете Fuzzy Logic).

### Функция dsigmf

Запись:

```
y = dsigmf(x,[a1 c1 a2 c2])
```

*Описание.* Задается функция принадлежности, определяемая как разность двух сигмоидальных функций. Сигмоидальная функция, как известно, описывается выражением

$$f(x,a,c) = \frac{1}{1+\exp(-a(x-c))}$$

и зависит от двух числовых параметров $a$ и $c$. Описываемая функция, как отмечено, является разностью двух сигмоидальных функций

$$f_1(x, a_1, c_1) - f_2(x, a_2, c_2)$$

и зависит от четырех параметров $a_1$, $c_1$, $a_2$, $c_2$ (от вектора параметров `[a1 c1 a2 c2]`).

Пример (рис. 6.30):

```
» x=0:0.1:10;
» y=dsigmf(x,[5 2 5 7]);
» plot(x,y)
» xlabel('dsigmf, P=[5 2 5 7]')
```

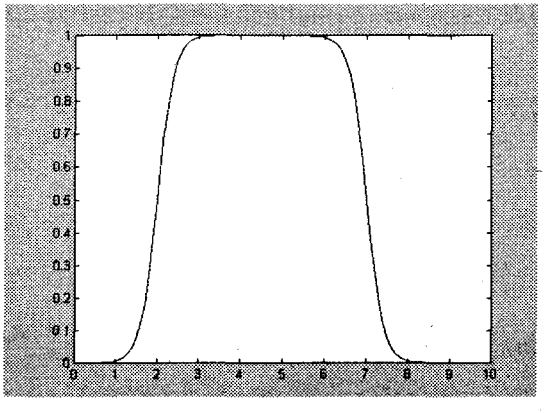


**Рис. 6.30.** Вид функции dsigmf(x,[5 2 5 7])

### Функция gauss2mf

Запись:

```
y = gauss2mf(x,[sig1 c1 sig2 c2])
```

*Описание.* Задается функция принадлежности, являющаяся разностью двух гауссовых функций, определяемая соотношением

$$f(x,\sigma_1,c_1,\sigma_2,c_2) = e^{-\frac{(x-c_1)^2}{\sigma_1^2}} - e^{-\frac{(x-c_2)^2}{\sigma_2^2}}$$

и зависящая от четырех параметров ($\sigma_1$, $c_1$, $\sigma_2$, $c_2$) или вектора параметров `[sig1 c1 sig2 c2]`.

Пример (рис. 6.31):

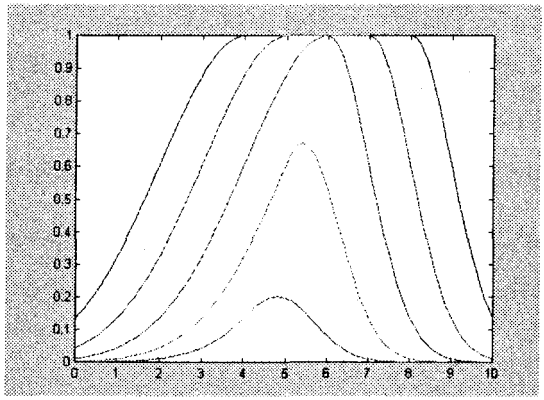


**Рис. 6.31.** Семейство кривых, определяемых функцией gauss2mf

```
» x = (0:0.1:10)';
» y1 = gauss2mf(x, [2 4 1 8]);
» y2 = gauss2mf(x, [2 5 1 7]);
» y3 = gauss2mf(x, [2 6 1 6]);
» y4 = gauss2mf(x, [2 7 1 5]);
» y5 = gauss2mf(x, [2 8 1 4]);
» plot(x, [y1 y2 y3 y4 y5]);
» set(gcf, 'name', 'gauss2mf', 'numbertitle', 'off');
```

### Функция gaussmf

Запись:

```
y = gaussmf(x,[sig c])
```

*Описание.* Задается функция принадлежности гауссова типа, зависящая от двух параметров (`[sig c]`).

Пример (рис. 6.32):

```
» x=0:0.1:10;
» y=gaussmf(x,[2 5]);
» plot(x,y)
» xlabel('gaussmf, P=[2 5]')
```

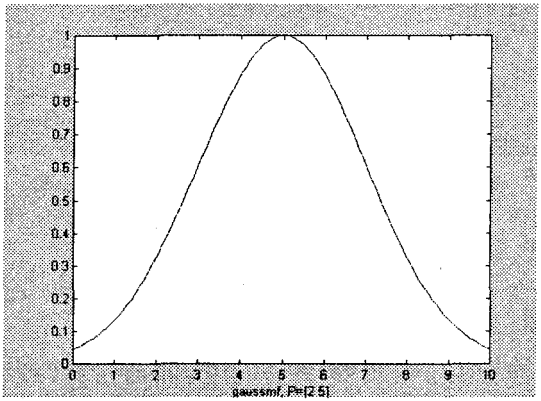


**Рис. 6.32.** Функция принадлежности типа гауссовой кривой



### Функция gbellmf

Запись:

```
y = gbellmf(x,params)
```

*Описание.* Задается функция принадлежности так называемого обобщенного колоколообразного типа с аналитическим описанием, зависящим от трех числовых параметров:

$$f(x,a,b,c) = \frac{1}{1+\left|\frac{x-c}{a}\right|^{2b}}.$$

Пример (рис. 6.33):

```
» x=0:0.1:10;
» y=gbellmf(x,[2 4 6]);
» plot(x,y)
» xlabel('gbellmf, P=[2 4 6]')
```

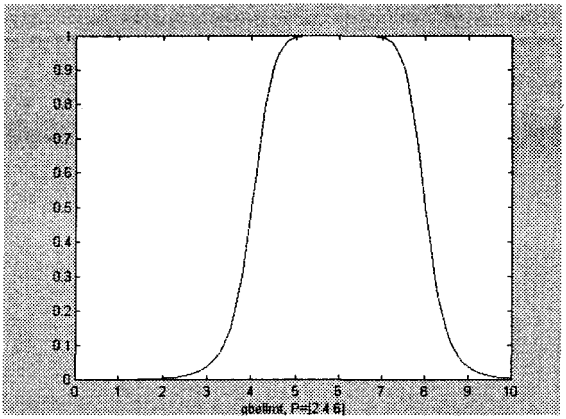


**Рис. 6.33.** График обобщенной колоколообразной функции

### Функция pimf

Запись:

```
y = pimf(x,[a b c d])
```

*Описание.* Задается так называемая π-образная функция принадлежности, получившая свое название из-за своеобразной формы. Функция вычисляется с использованием сплайн-аппроксимации по четырем точкам, задаваемым вектором параметров. Параметры `a` и `d` определяют основание кривой, а параметры `b` и `c` — положение плоской вершины.

Пример (рис. 6.34):

```
» x=0:0.1:10;
» y=pimf(x,[1 4 5 10]);
» plot(x,y)
» xlabel('pimf, P=[1 4 5 10]')
```

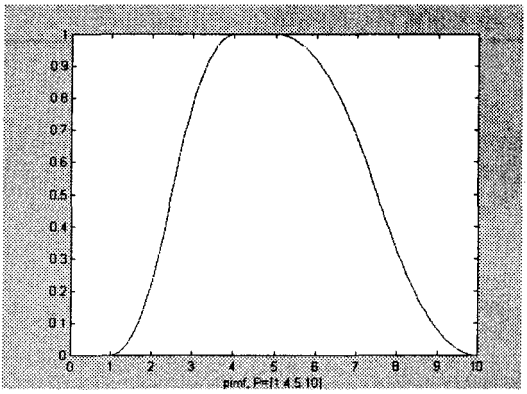

**Рис. 6.34.** График π-образной функции

### Функция psigmf

Запись:

```
y = psigmf(x,[a1 c1 a2 c2])
```

*Описание.* Задается функция принадлежности, определяемая как произведение двух сигмоидальных функций

$$f_1(x, a_1, c_1)\, f_2(x, a_2, c_2)$$

и зависящая от четырех параметров $a_1$, $c_1$, $a_2$, $c_2$ (от вектора параметров `[a1 c1 a2 c2]`).

Пример (рис. 6.35):

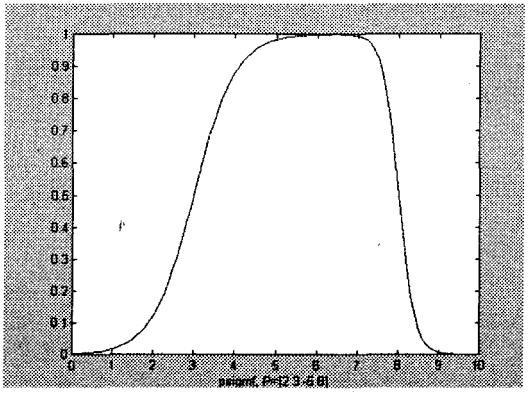

**Рис. 6.35.** Пример использования функции psigmf

```
» x=0:0.1:10;
» y=psigmf(x,[2 3 -5 8]);
» plot(x,y)
» xlabel('psigmf, P=[2 3 -5 8]')
```

### Функция smf

Запись:

```
y = smf(x,[a b])
```

*Описание.* Задается зависящая от двух параметров так называемая S-образная функция принадлежности. Параметры a и b определяют диапазон значений аргумента, где функция возрастает.

Пример (рис. 6.36):

```
» x=0:0.1:10;
» y=smf(x,[1 8]);
» plot(x,y)
» xlabel('smf, P=[1 8]')
```

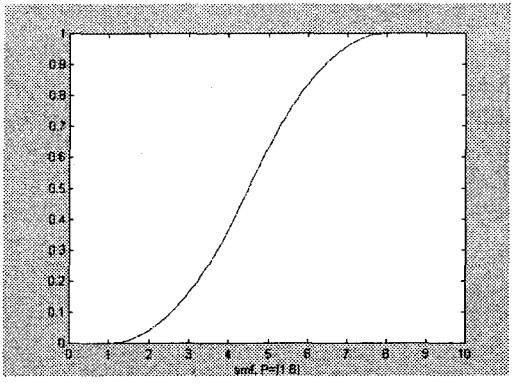


**Рис. 6.36.** График S-образной функции

### Функция sigmf

Запись:

```
y = sigmf(x,[a c])
```

*Описание.* Задается так называемая сигмоидальная функция принадлежности, определяемая выражением, приведенным выше и зависящим от двух параметров.

Пример (рис. 6.37):

```
» x=0:0.1:10;
» y=sigmf(x,[2 4]);
» plot(x,y)
» xlabel('sigmf, P=[2 4]')
```

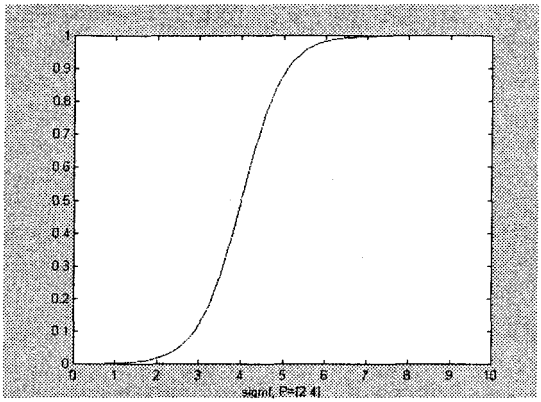


**Рис. 6.37.** График сигмоидальной функции



### Функция trapmf

Запись:

```
y = trapmf(x,[a b c d])
```

*Описание.* Задается так называемая трапецеидальная функция принадлежности, зависящая от четырех параметров и определяемая формулой

$$f(x,a,b,c,d)=\begin{cases}0, & x \le a,\\ \dfrac{x-a}{b-a}, & a \le x \le b,\\ 1, & b \le x \le c,\\ \dfrac{d-x}{d-c}, & c \le x \le d,\\ 0, & d \le x\end{cases}$$

при этом параметры $a$ и $d$ определяют основание кривой, а $b$ и $c$ — положение вершины.

Пример (рис. 6.38):

```
» x=0:0.1:10;
» y=trapmf(x,[1 5 7 8]);
» plot(x,y)
» xlabel('trapmf, P=[1 5 7 8]')
```

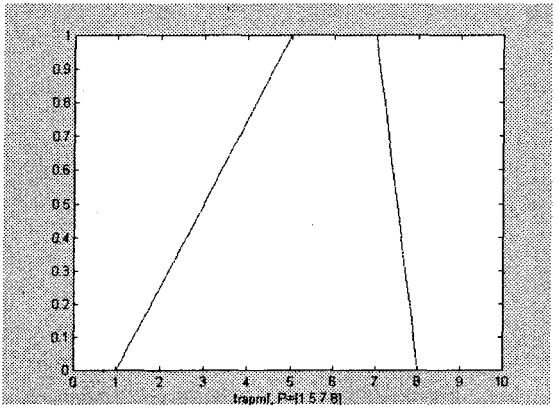


**Рис. 6.38.** Трапецеидальная функция принадлежности

### Функция trimf

Запись:

```
y = trimf(x,params)
y = trimf(x,[a b c])
```

*Описание.* Задается зависящая от трех параметров функция принадлежности треугольной формы, при этом параметры $a$ и $c$ определяют основание треугольника, а параметр $b$ — координату его вершины.

Пример (рис. 6.39):

```
» x=0:0.1:10;
» y=trimf(x,[3 6 8]);
» plot(x,y)
» xlabel('trimf, P=[3 6 8]')
```

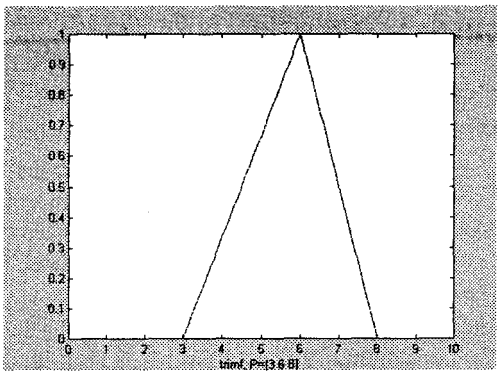

**Рис. 6.39.** График функции принадлежности треугольной формы

### Функция zmf

Запись:

```
y = zmf(x,[a b])
```

*Описание.* Задается зависящая от двух параметров функция принадлежности так называемой Z-образной формы. Параметры определяют диапазон убывания функции.



Пример (рис. 6.40):

```
» x=0:0.1:10;
» y=zmf(x,[3 7]);
» plot(x,y)
» xlabel('zmf, P=[3 7]')
```

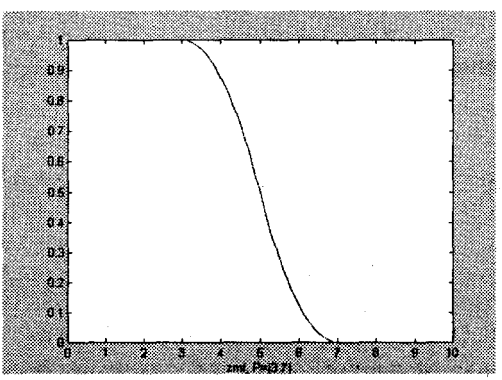

**Рис. 6.40.** Функция принадлежности Z-образной формы

# Функции систем нечеткого вывода

## Функции сохранения, открытия и использования созданной системы

Чтение, использование и сохранение на диск созданной системы нечеткого вывода в режиме командной строки осуществляется следующими функциями:

```
readfis('имя_файла'),
evalfis(вектор_параметров, имя),
writefis(имя) или writefis( имя, 'имя_файла')
```

Здесь `имя_файла` — наименование файла с записанной системой (без указания расширения), `имя` — идентификатор, который дан системе в рабочей среде MATLAB, `вектор_параметров` — набор значений входов, для которых требуется рассчитать выход (возможна и матрица параметров, тогда результат расчетов — вектор в случае одной выходной переменной или матрица при нескольких таких переменных).

Примеры (применительно к ранее созданной и сохраненной на диске в виде файла с именем tip экспертной системе для задачи о чаевых):

```
» readfis('tip');
» evalfis([1 2],a)
ans =
    7.3949
» writefis(a)
ans =
tip
```

## Функции использования графического окна

Следующие три функции позволяют использовать элементы графических изображений вне программ с графическим интерфейсом.

Команда (функция) `plotfis(a)` вызовет появление графического окна MATLAB с мнемоническим представлением разработанной системы (рис. 6.41).

Команда (функция) `plotmf` выполняет аналогичную операцию, но по отношению к графикам функций принадлежности. На рис. 6.42 приведен результат выполнения функции `plotmf(a,'input',1)`.

Наконец, команда (функция) `gensurf(a)` также делает то же самое, но применительно к поверхности отклика (рис. 6.43).

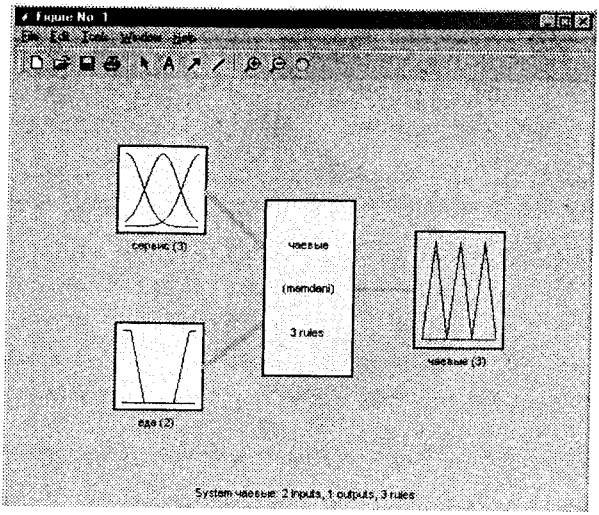


**Рис. 6.41.** Мнемоническое представление системы нечеткого вывода

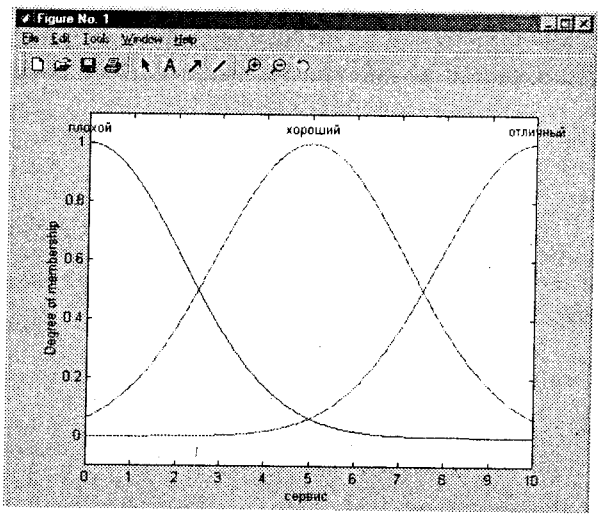


**Рис. 6.42.** Результат выполнения функции plotmf(a,'input',1)

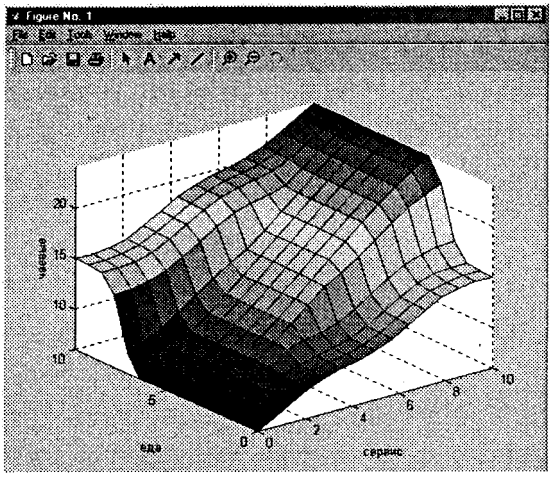


**Рис. 6.43.** Результат выполнения функции gensurf(a)

6

## Функции создания, просмотра структуры и редактирования систем нечеткого вывода

Вообще, при желании можно совсем обойтись без программ графического интерфейса и, используя функции `newfis` (новая система), `addvar` (добавить переменную), `addmf` (добавить функцию принадлежности) и `addrule` (добавить правило), сконструировать систему нечеткого вывода целиком в режиме командной строки MATLAB. Процесс этот, надо сказать, значительно более трудоемкий, чем с применением указанных программ, поэтому, не вдаваясь особенно в детали, приведем лишь пример построения такой системы в задаче о чаевых.

```
a=newfis('tip');
a=addmf(a,'input',1,'сервис',[0 10]);
a=addmf(a,'input',1,'плохой','gaussmf',[1.5 0]);
a=addmf(a,'input',1,'хороший','gaussmf',[1.5 5]);
a=addmf(a,'input',1,'отличный','gaussmf',[1.5 10]);
a=addvar(a,'input','еда',[0 10]);
a=addmf(a,'input',2,'подгоревшая','trapmf',[-2 0 1 3]);
a=addmf(a,'input',2,'превосходная','trapmf',[7 9 10 12]);
a=addvar(a,'output','tip',[0 30]);
a=addmf(a,'output',1,'малые','trimf',[0 5 10]);
a=addmf(a,'output',1,'средние','trimf',[10 15 20]);
```

```
a=addmf(a,'output',1,'щедрые','trimf',[20 25 30]);
ruleList=[ ...
1 1 1 1 2
2 0 2 1 1
3 2 3 1 2 ];
a=addrule(a,ruleList);
```

Далее при необходимости просмотра элементов структуры (системы с идентификатором a) в режиме командной строки следует ввести команду вида a.параметр, например

```
» a.type
```

Получим ответ:

```
ans =
mamdani
```

Получение информации по всем элементам структуры обеспечивается функцией getfis(a). Вот результат ее выполнения:

```
Name       = tip
      Type       = mamdani
      NumInputs = 2
      InLabels  =
            сервис
            еда
      NumOutputs = 1
      OutLabels =
            чаевые
      NumRules = 3
      AndMethod = min
      OrMethod = max
      ImpMethod = min
      AggMethod = max
      DefuzzMethod = centroid
ans =
tip
```

Функция setfis в определенном смысле противоположна функции getfis и позволяет изменять параметры. Пример выполнения функции:

```
» a=setfis(a,'name','чаевые')
a =
name: 'чаевые'
type: 'mamdani'
andMethod: 'min'
```

```
orMethod: 'max'
defuzzMethod: 'centroid'
impMethod: 'min'
aggMethod: 'max'
input: [1x2 struct]
output: [1x1 struct]
rule: [1x3 struct]
```

Полный просмотр структуры системы нечеткого вывода осуществляется функцией **showfis(a)**.

Просмотр правил, включенных в систему нечеткого вывода, осуществляется с помощью функции **showrule**.

Запись:

```
showrule(имя)
showrule(имя,номера_правил)
showrule(fis,номера_правил,формат)
showrule(fis,номера_правил,формат,язык)
```

*Описание.* Параметр **имя** — это идентификатор рассматриваемой системы в среде MATLAB, **номера_правил** — список правил, которые нужно показать, **формат** — строковая переменная, имеющая значения **'verbose'**, **'symbolic'** или **'indexed'**, **язык** — строковая переменная со значениями **'english'**, **'francais'** или **'deutsch'** (установки по умолчанию — **'verbose'** и **'english'**).

Примеры:

```
» a=readfis('tip');
» showrule(a,1)
ans =
1. If (сервис is плохой) or (еда is подгоревшая) then (чаевые is
малые) (1)
» showrule(a,[3 1],'symbolic')
ans =
3. (сервис==отличный) | (еда==превосходная) => (чаевые=щедрые) (1)
1. (сервис==плохой) | (еда==подгоревшая) => (чаевые=малые) (1)
```

Функция **rmmf** используется для удаления функции принадлежности из состава системы.

Запись:

```
имя = rmmf(имя,'varType',varIndex,'mf',mfIndex)
```

*Описание.* Параметры функции: **имя** — идентификатор системы в среде MATLAB, **varType** — строковая переменная со значениями **'input'**

или 'output', varIndex — порядковый номер переменной (по списку переменных входа или выхода), mf — строковая переменная, задающая функцию принадлежности, mfIndex — порядковый данной функции.

Функция rmvar удаляет переменную из состава системы.

Запись:

```
[новое_имя,errorStr] = rmvar(имя,'varType',varIndex)
новое_имя = rmvar(имя,'varType',varIndex)
```

*Описание.* Здесь переменные varType и varIndex имеют тот же смысл, что и в предыдущей функции, переменная errorStr позволяет записать любое сообщение об ошибке, новое_имя — новое имя системы с исключенной переменной.

Функция приведения к четкости (дефаззификации) defuzz позволяет по заданной функции принадлежности определить соответствующее четкое значение.

Запись:

```
out = defuzz(x,mf,type)
```

*Описание.* Здесь x — обозначение числового аргумента, mf — тип функции принадлежности, type — задаваемый метод приведения к четкости — строковая переменная (в кавычках), имеющая следующие возможные значения:

- centroid;
- bisector;
- mom;
- som;
- lom.

Смысл этих значений был рассмотрен ранее.

Пример:

```
» x = -10:0.1:10;
» mf = trapmf(x,[-10 -8 -4 7]);
» xx = defuzz(x,mf,'centroid')
xx =
   -3.2857
```

Функция evalmf вычисляет значения заданной функции принадлежности.

Запись:

```
y = evalmf(x,mfParams,mfType)
```

*Описание.* Здесь x — обозначение числового аргумента, **mfParams** — вектор необходимых числовых параметров, **mfType** — тип функции принадлежности.

Пример (рис. 6.44):

```
» x=0:0.1:10;
» mfparams = [2 4 6];
» mftype = 'gbellmf';
» y=evalmf(x,mfparams,mftype);
» plot(x,y)
» xlabel('gbellmf, P=[2 4 6]')
```

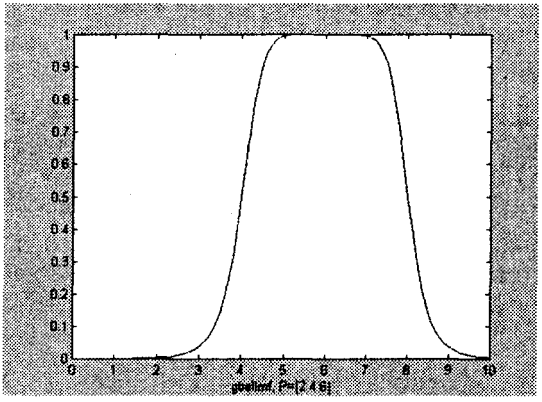


**Рис. 6.44.** Графическое представление результата выполнения команды y = evalmf(x,mfParams,mfType)

Функция **mf2mf** позволяет заменять одну функцию принадлежности близкой ей по форме другой с некоторым «эквивалентным» набором числовых параметров.

Запись:

```
outParams = mf2mf(inParams,inType,outType)
```

*Описание.* Здесь **inParams** — набор параметров исходной функции, **inType** — тип исходной функции принадлежности, **outType** — тип эквивалентной функции принадлежности, **outParams** — набор преобразованных параметров.

Пример:

```
» x=0:0.1:5;
» mfp1 = [1 2 3];
```

```
» mfp2 = mf2mf(mfp1,'gbellmf','trimf')
mfp2 =
     1     3     5
» plot(x,gbellmf(x,mfp1),x,trimf(x,mfp2))
```

Функцию parsrule можно отнести к числу сервисных.

Запись:

```
новое_имя = parsrule(имя,текст_правил)
новое_имя = parsrule(имя, текст_правил,формат)
новое_имя = parsrule(имя, текст_правил,формат,язык)
```

*Описание.* В данной функции идентификаторы имя и новое_имя относятся к исходной и модернизированной системам с нечетким выводом, строковые переменные формат и язык имеют тот же смысл, что и в рассмотренной ранее функции showrule, текст_правил — заданный в текстовой форме набор правил системы (на языке, определенном переменной; по умолчанию — английский, что допускает использование, в частности, русских имен для переменных, но требует английских связующих слов if, and, or, then, is). Функция выполняет грамматический разбор исходного текста и выдает набор нечетких правил в указанном формате (по умолчанию — подробный, verbose).

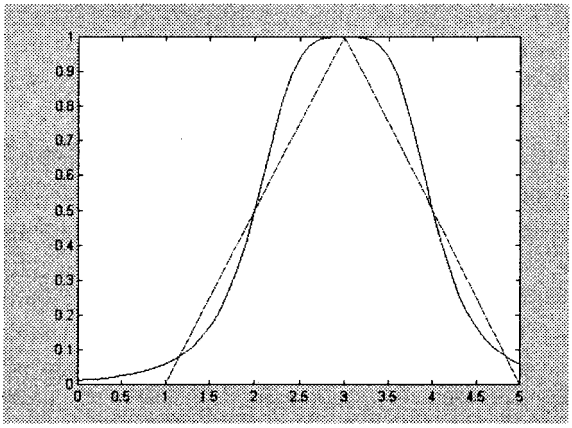


**Рис. 6.45.** Графики исходной (колоколообразной) и эквивалентной (треугольной) функций принадлежности

# Дополнительные функции

## Функция создания и/или обучения гибридных сетей с архитектурой ANFIS

Функция `anfis` записывается в виде

```
[имя,ошибка_о,шаг] = anfis(trnData)
[имя,ошибка_о,шаг] = anfis(trnData,имя)
[имя1,ошибка_о,шаг] = anfis(trnData,имя,trnOpt,dispOpt)
[имя1,ошибка_о,шаг,имя2,ошибка_п] =
anfis(trnData,trnOpt,dispOpt,chkData)
[имя1,ошибка_о,шаг,имя2,ошибка_п] =
anfis(trnData,trnOpt,dispOpt,chkData,optMethod)
```

Функция предназначена для создания и/или обучения гибридных сетей с архитектурой ANFIS. Значения аргументов функции:

- `trnData` — матрица данных для обучения сети (обучающая выборка); последний столбец соответствует единственной выходной переменной, остальные столбцы — входным переменным;
- `имя` — идентификатор создаваемой гибридной сети; если структура системы нечеткого вывода с таким идентификатором уже создана, то она будет использована для настройки числовых параметров, в противном случае структура будет создана при выполнении функции с опциями, по умолчанию соответствующими выполнению функции `genfis1` (см. ниже);
- `trnOpt` — вектор опций обучения, элементы которого имеют следующий смысл:
  - `trnOpt(1)` — количество циклов обучения (по умолчанию 10);
  - `trnOpt(2)` — целевой уровень ошибки обучения (по умолчанию 0);
  - `trnOpt(3)` — начальный шаг алгоритма обучения (по умолчанию 0,01);
  - `trnOpt(4)` — коэффициент уменьшения шага (по умолчанию 0,9);
  - `trnOpt(5)` — коэффициент увеличения шага (по умолчанию 1,1);
- `dispOpt` — вектор опций вида выводимой информации (по умолчанию все элементы — единичные, что означает вывод всех видов возможной информации в процессе выполнения функции) со следующими элементами:
  - `dispOpt(1)` — ANFIS-информация, такая как число входов, функции принадлежности и т. п.;
  - `dispOpt(2)` — ошибка;

  - `dispOpt(3)` — шаг обновления (корректировки) по каждому параметру;
  - `dispOpt(4)` — конечные результаты;
- `chkData` — матрица проверочных данных (проверочная выборка), имеющая такой же набор столбцов, что и матрица данных для обучения сети, но, вообще говоря, другое число строк;
- `optMethod` — параметр, определяющий вид алгоритма обучения гибридной системы (1 — гибридный алгоритм (используется по умолчанию), 0 — алгоритм обратного распространения ошибки).

Процесс обучения сети заканчивается, когда выполнено заданное число циклов обучения или ошибка обучения уменьшилась до заданного уровня. При отсутствии некоторых аргументов их значения принимаются по умолчанию.

Выходные параметры функции:

- `ошибка_о` — массив (вектор) значений ошибок для обучающей выборки;
- `ошибка_п` — массив (вектор) значений ошибок для проверочной выборки;
- `шаг` — массив значений шага в алгоритме обучения;
- `имя1` — идентификатор (имя) сети, образованной исходя из минимальной ошибки обучения;
- `имя2` — идентификатор (имя) сети, образованной исходя из минимальной ошибки для проверочной выборки.



Рассмотрим пару примеров.

**Пример 1** (рис. 6.46):

```
» x = (0:0.1:10)';
» y = sin(2*x)./exp(x/5);
» trnData = [x y];
» a = anfis(trnData);
ANFIS info:
   Number of nodes: 12
   Number of linear parameters: 4
   Number of nonlinear parameters: 6
   Total number of parameters: 10
   Number of training data pairs: 101
   Number of checking data pairs: 0
   Number of fuzzy rules: 2
Start training ANFIS ...
```

```
   1    0.313942
   2    0.31388
   3    0.313817
   4    0.313753
   5    0.313689
Step size increases to 0.011000 after epoch 5.
   6    0.313624
   7    0.313552
   8    0.313479
   9    0.313406
Step size increases to 0.012100 after epoch 9.
  10    0.313332
Designated epoch number reached --> ANFIS training completed at epoch 10.
» plot(x,y,x,evalfis(x,a),'-.');
» legend('Обучающая выборка','Выход ANFIS');
```

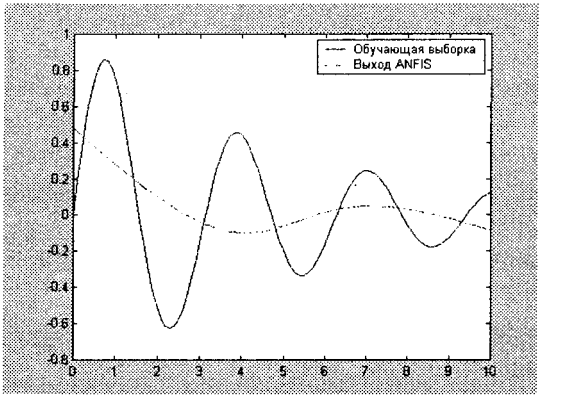


**Рис. 6.46.** Обучающая выборка и выход системы a

**Пример 2** (рис. 6.47):

```
» x = (0:0.1:10)';
» y = sin(2*x)./exp(x/5);
» trnData = [x y];
» numMFs = 5;
» mfType = 'gbellmf';
» epoch_n = 20;
» in_fismat = genfis1(trnData,numMFs,mfType);
» out_fismat = anfis(trnData,in_fismat,20);
ANFIS info:
```

```
    Number of nodes: 24
    Number of linear parameters: 10
    Number of nonlinear parameters: 15
    Total number of parameters: 25
    Number of training data pairs: 101
    Number of checking data pairs: 0
    Number of fuzzy rules: 5
Start training ANFIS ...
    1    0.0694086
    2    0.0680259
    3    0.066663
    4    0.0653198
    5    0.0639961
Step size increases to 0.011000 after epoch 5.
    6    0.0626917
    7    0.0612787
    8    0.0598881
    9    0.0585193
Step size increases to 0.012100 after epoch 9.
   10    0.0571712
   11    0.0557113
   12    0.0542741
   13    0.052858
Step size increases to 0.013310 after epoch 13.
   14    0.0514612
   15    0.0499449
   16    0.0484475
   17    0.0469667
Step size increases to 0.014641 after epoch 17.
   18    0.0455003
   19    0.0439022
   20    0.0423184
Designated epoch number reached --> ANFIS training completed at epoch 20.
» plot(x,y,x,evalfis(x,out_fismat),'-.');
»  legend('Training Data','ANFIS Output');
```

Во втором примере при проектировании гибридной системы заранее были заданы некоторые значения (количество и тип функций принадлежности, количество циклов обучения), в первом примере для той же обучающей выборки все установки при проектировании системы выбраны по умолчанию. Различия в качестве функционирования двух систем наглядно иллюстрируются рис. 6.46 и 6.47.

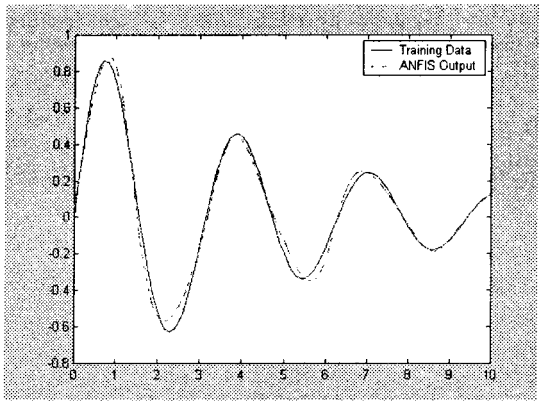


**Рис. 6.47.** Обучающая выборка и выход системы fismat

## Функция кластеризации

Функция `fcm` осуществляет кластеризацию с использованием алгоритма Fuzzy c-means (см. выше). Она записывается в виде

```
[center,U,obj_fcn] = fcm(data,cluster_n)
[center,U,obj_fcn] = fcm(data,cluster_n,options)
```

*Описание.* Аргументы функции:

- `data` — матрица данных, каждый столбец которой соответствует одной из переменных, а каждая строка — одному из опытов;
- `cluster_n` — число задаваемых кластеров (должно быть больше 1);
- `options` — вектор опций функции со следующими элементами:
  - `options(1)` — показатель для матрицы `U` (по умолчанию 2,0);
  - `options(2)` — максимальное число итераций при определении центров кластеров (по умолчанию 100);
  - `options(3)` — минимальная сумма улучшения (по умолчанию 1e-5);
  - `options(4)` — вывод информации в процессе итераций (по умолчанию 1).

Выходные параметры функции:

- `center` — матрица, элементы которой (по столбцам) являются координатами найденных центров кластеров, число строк равно числу центров;

- U — матрица значений принадлежности данных к выявленным кластерам;
- obj_fcn — значения целевой функции в процессе итераций.

Пример:

```
» load clusterdemo.dat;
» data=clusterdemo;
» [center,U,obj_fcn] = fcm(data, 3);
Iteration count = 1, obj. fcn = 55.941585
Iteration count = 2, obj. fcn = 42.971277
Iteration count = 3, obj. fcn = 42.764109
Iteration count = 4, obj. fcn = 41.688335
Iteration count = 5, obj. fcn = 37.070571
Iteration count = 6, obj. fcn = 29.262573
Iteration count = 7, obj. fcn = 22.996848
Iteration count = 8, obj. fcn = 16.162985
Iteration count = 9, obj. fcn = 15.443897
Iteration count = 10, obj. fcn = 15.430759
Iteration count = 11, obj. fcn = 15.430533
Iteration count = 12, obj. fcn = 15.430528
» center
center =
    0.2051    0.5960    0.8090
    0.7939    0.7892    0.1968
    0.5963    0.1923    0.5057
```



## Функция генерации FIS-структуры

Функция genfis1 генерирует FIS-структуру по экспериментальным данным без проведения их кластеризации:

```
имя = genfis1(data)
имя = genfis1(data,numMFs,inmftype, outmftype)
```

*Описание.* Функцией генерируется структура системы нечеткого вывода типа Sugeno, являющаяся исходной для последующего формирования и обучения гибридной системы с помощью функции anfis.

Аргументы функции:

- data — матрица данных (обучающая выборка), последний столбец которой соответствует выходной переменной, а остальные — входным переменным и число строк в которой равно числу наборов экспериментальных данных (образцов);
- numMFs — вектор, элементы которого определяют число функций принадлежности, задаваемых для каждого входа; если для всех

входов нужно указать одно и то же число таких функций, данный аргумент задается как скаляр (один элемент);

- `inmftype` — строковый массив, элементы которого — типы функций принадлежности, задаваемые для входных переменных;
- `outmftype` — строковая переменная, определяющая тип функций принадлежности выходной переменной (возможны только значения `'linear'` или `'constant'`).

Число функций принадлежности выходной переменной равно числу нечетких правил, генерируемых `genfis1`, и зависит от элементов вектора `numMFs`. По умолчанию `numMFs=2`, `inmftype='gbellmf'`. Значения по умолчанию устанавливаются, если функция применяется без трех последних аргументов.

Применение функции ограничено размерностью задачи: так, при $N$ входных переменных с минимальным разбиением области определения каждой переменной на две подобласти будет генерироваться $2^N$ правил, что при числе входов больше 5–6 делает анализ этих правил практически невозможным. Более экономной в этом плане является функция `genfis2` (см. ниже).

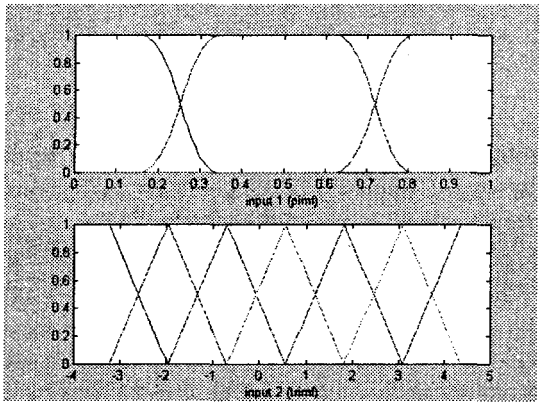


**Рис. 6.48.** Функции принадлежности, определенные genfis1

Пример (рис. 6.48):

```
» data = [rand(10,1) 10*rand(10,1)-5 rand(10,1)];
» numMFs = [3 7];
» mfType = str2mat('pimf','trimf');
» fismat = genfis1(data,numMFs,mfType);
```

```
» [x,mf] = plotmf(fismat,'input',1);
» subplot(2,1,1), plot(x,mf);
» xlabel('input 1 (pimf)');
» [x,mf] = plotmf(fismat,'input',2);
» subplot(2,1,2), plot(x,mf);
» xlabel('input 2 (trimf)');
```

См. также пример 2 для функции anfis.

## Функция генерации структуры нечеткого вывода

Функция genfis2 генерирует структуру системы нечеткого вывода (типа Sugeno) с использованием алгоритма Subtractive clustering кластеризации данных:

```
имя = genfis2(Xin,Xout,radii)
имя = genfis2(Xin,Xout,radii,xBounds)
имя = genfis2(Xin,Xout,radii,xBounds,options)
```

Аргументами рассматриваемой функции являются:

- Xin — матрица (массив) входных данных обучающей выборки, столбцы которой ассоциированы с входными переменными, а каждая строка — с отдельным опытом;
- Xout — матрица выходных переменных обучающей выборки, столбцы которой представляют значения данных переменных, а число строк равно числу строк матрицы Xin;
- radii (радиусы) — вектор, определяющий «области влияния» центров кластеров по каждой входной переменной (в предположении, что область определения данных переменных — многомерный параллелепипед) с неотрицательными элементами, меньшими единицы; пусть, например, число входов — 3, тогда radii=[0.5 0.4 0.3] означает, что по первой переменной «область влияния» составляет 0,5 от диапазона изменения этой переменной, а по второй и третьей — соответственно, 0,4 и 0,3 (эти диапазоны определяются по элементам столбцов матрицы Xin). Если рассматриваемый аргумент задан как скаляр, то по всем переменным устанавливается один и тот же размер «области влияния» кластеров. Параметр имеет важное значение для проведения разбиения области определения входов на подобласти с последующим установлением числа нечетких правил;
- XBounds — матрица с 2 строками и $N$ столбцами ($N$ — число столбцов в матрице Xin, то есть число входных переменных), устанавливающая границы областей определения входных переменных.

Данные этой матрицы используются для преобразования (масштабирования) входов к диапазону [-1, 1]. Например, Xbounds = = [-10 0 -1; 10 50 1] задает диапазон [-10, 10] для первой переменной, [0, 50] — для второй и [-1, 1] — для третьей. Если рассматриваемый аргумент опущен, границы областей определяются как максимальные и минимальные элементы по столбцам матрицы Xin;

- options — вектор опций, устанавливающий параметры алгоритма кластеризации (подробнее об этом см. в описании функции subclust).

Функция genfis2 так же, как genfis1, может применяться в комплекте с функцией anfis (для настройки параметров системы нечеткого вывода), но, в отличие от последней, genfis2 возвращает уже полностью определенную, готовую к использованию систему, не требующую, вообще говоря, дополнительной оптимизации.

Функция genfis2 производит более экономное, чем genfis1, разбиение пространства входов на подобласти и может поэтому являться эффективным инструментом для выявления правил из набора *экспериментальных данных*.

Примеры:

```
a = genfis2(Xin,Xout,0.5)
a = genfis2(Xin,Xout,[0.5 0.25 0.3])
```

### Функция возврата центров кластеров

Функция subclust возвращает центры кластеров.

Запись:

```
[C,S] = subclust(X,radii,xBounds,options)
```

Функцией определяются центры кластеров набора экспериментальных данных, отражаемых матрицей X (столбцы матрицы соответствуют переменным, строки — экспериментальным «точкам» или опытам) на основе реализации «вычитающего» алгоритма кластеризации (Subtractive clustering). В его основе лежит предположение, что каждая экспериментальная точка может быть центром кластера, при этом вначале для каждой точки вычисляется мера правдоподобия данного предположения («потенциал точки»), основанная на плотности точек в заданной окрестности рассматриваемой. Дальнейшие вычисления происходят итеративно:

- точка с наибольшим потенциалом объявляется центром первого кластера;

- из отмеченной окрестности этой точки удаляются все остальные точки;
- из оставшихся точек объявляется центр следующего кластера.

Процесс продолжается, пока не будут рассмотрены (исключены или объявлены центрами кластеров) все точки.

Размеры окрестностей задаются элементами вектора `radii`, который (как и аргумент `xBounds`) аналогичен рассмотренному при описании функции `Genfis2`.

Аргумент `options` является вектором со следующими элементами:

- `options(1) = quashFactor` (фактор подавления): используется для расширения «сферы влияния» выделенного центра кластера для «подавления» (уменьшения) потенциала точек, относящихся к данному кластеру; по умолчанию равен 1,25;
- `options(2) = acceptRatio` (коэффициент принятия): устанавливает, больше какого значения по отношению к потенциалу принятого центра кластера должен быть потенциал точки, чтобы эту точку можно было объявить центром нового кластера; по умолчанию 0,5;
- `options(3) = rejectRatio` (коэффициент отклонения): устанавливает, меньше какого значения по отношению к потенциалу принятого центра кластера должен быть потенциал точки, чтобы эту точку нельзя было объявить центром нового кластера; по умолчанию 0,15;
- `options(4) = verbose` (подробности): если этот элемент ненулевой, то выдается подробная информация о процессе нахождения центров кластеров; по умолчанию — 0.

Функция возвращает центры кластеров в виде матрицы `C`; каждая строка этой матрицы содержит координаты одного из найденных центров. Вектор `S` содержит элементы, определяющие диапазоны влияния центров кластеров по каждой переменной (для всех центров эти диапазоны одинаковы).

Пример:

```
» X=load('clusterdemo.dat');
» [C,S] = subclust(X,0.5)
C =
    0.2220    0.5937    0.8113
    0.7797    0.8191    0.1801
    0.5914    0.1721    0.4872
S =
    0.1904    0.1988    0.2251
```

## Сервисные функции

К этой группе относятся 10 функций:

- `convertfis` — обеспечивает преобразование FIS-матрицы пакета Fuzzy Logic версии 1.0 в FIS-структуру пакета Fuzzy Logic версии 2.0:
  `имя_новое=convertfis(имя_старое)`
- `discfis` — преобразует непрерывные функции принадлежности системы нечеткого вывода в дискретные (задаваемые наборами точек);
- `evalmmf` — вычисляет значения одновременно нескольких функций принадлежности. Является многомерным аналогом функции `evalmf`;
- `fstrvcat` — объединяет заданные векторы и матрицы в одну матрицу:
  `Y= fstrvcat(y1,y2,y3,..)`
  Функция возвращает матрицу `Y`, представляющую собой объединение матриц-аргументов `y1`, `y2`, `y3`,..., при этом для выравнивания размеров итоговой матрицы она дополняется нулями. Элементы аргументов могут быть строкового типа, при этом их символы преобразуются в числовую форму.
- `fuzarith` — выполняет алгебраические операции над нечеткими множествами:
  `C = fuzarith(X, A, B, operator)`
  Здесь `X` — вектор, выполняющий роль универсального множества, `A` и `B` — векторы, определяющие нечеткие подмножества-операнды, `C` — вектор, определяющий результат операции (нечеткое подмножество), `operator` — строковая переменная, задающая вид операции и имеющая одно из возможных значений `'sum'` (сумма), `'sub'` (*вычитание*), `'prod'` (*умножение*), `'div'` (*деление*). Пример (рис. 6.49):

```
» % Задание количества точек универсального множества
» point_n = 101;
» x = linspace(min_x, max_x, point_n)';
» % Трапецеидальная функция принадлежности подмножества A
» A = trapmf(x, [-10 -2 1 3]);
» % Гауссова функция принадлежности подмножества B
» B = gaussmf(x, [2 5]);
» C1 = fuzarith(x, A, B, 'sum');
```

```
» subplot(2,1,1);
» plot(x, A, 'b--', x, B, 'm:', x, C1, 'c');
» title('Нечеткая сумма A+B');
» C2 = fuzarith(x, A, B, 'prod');
» subplot(2,1,2);
» plot(x, A, 'b--', x, B, 'm:', x, C2, 'c');
» title('Нечеткое произведение A*B');
```

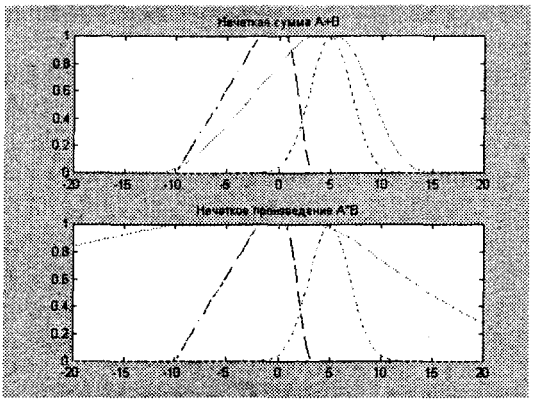


**Рис. 6.49.** Результаты выполнения функции fuzarith

- `findrow` — возвращает номера аргументов функции `fstrvcat` по их именам.
- `genparam` — возвращает параметры заданных функций принадлежности, определяя их по набору входных экспериментальных данных. Данные результаты могут быть использованы как начальные приближения при последующем применении команды `anfis`.
- `mam2sug` — преобразует систему нечеткого вывода с алгоритмом Mamdani в систему, использующую алгоритм Sugeno:

  `имя_sug=mam2sug(имя_mam)`

  Функция возвращает систему нечеткого вывода (с именем `имя_sug`), использующую алгоритм вывода Sugeno, по заданной системе `имя_ mam`, использующей алгоритм Mamdani. Возвращаемая система имеет постоянные функции принадлежности выходных переменных (их может быть несколько), определяемые как центроиды функций принадлежности следствий нечетких правил алгоритма Mamdani. Предпосылки правил при преобразовании не изменяются. Пример (рис. 6.50):

```
» a=readfis('tip');
» b=mam2sug(a);
» subplot(2,1,1); gensurf(a);
» title('Выход системы Mamdani (чаевые)');
» subplot(2,1,2); gensurf(b);
» title('выход системы Sugeno (чаевые)');
```

**ЗАМЕЧАНИЕ**

Проверка, проведенная авторами, показала, что рассмотренная функция выполняется некорректно и приводит к неработающей системе вывода (попытки использования системы Sugeno вызывают сообщение об ошибке). Данный недостаток можно устранить, изменив в предпоследней строке файла mam2sug.m (в директории Matlab/toolbox/fuzzy/fuzzy/) defuzzmethod на defuzzMethod.

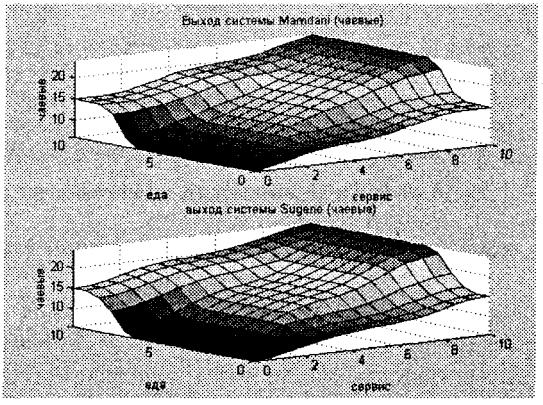


**Рис. 6.50.** Иллюстрация к выполнению функции mam2sug

- probor — выполняет операцию вероятностного ИЛИ над столбцами матрицы-аргумента:

  ```
  Y = probor(X)
  ```

  Если матрица-аргумент X имеет два столбца, A и B, то функция возвращает матрицу Y=A+B-A.*B; если аргумент содержит только один столбец, то Y=X.

- sugmax — позволяет определить максимально возможные диапазоны изменения выходных переменных в заданной системе нечеткого вывода, использующей алгоритм Sugeno.

# Функции вызова диалоговых окон интерфейса

Данную группу образуют 12 функций:

- `cmfdlg` — вызывает диалоговое окно для задания пользовательской функции принадлежности;
- `cmthdlg` — вызывает диалоговое окно для задания пользовательского алгоритма нечеткого вывода;
- `fisgui` — позволяет генерировать элементы графического интерфейса пользователя;
- `gfmfdlg` — вызывает диалоговое окно для задания числа и типа функций принадлежности при проектировании системы типа Sugeno (с последующим применением функции `anfis`);
- `mfdlg` — вызывает диалоговое окно для задания дополнительной функции принадлежности;
- `mfdrag` — обеспечивает «перетаскивание» функции принадлежности с использованием мыши;
- `popundo` — возвращает из стека результаты отмены предыдущего действия;
- `pushundo` — помещает в стек данные по отменяемому действию;
- `savedlg` — вызывает диалоговое окно, выдающее запрос на сохранение результатов;
- `statmsg` — помещает сообщения в поле состояния окна;
- `updtfis` — обновляет элементы графического интерфейса пользователя;
- `wsdlg` — вызывает диалоговое окно для выполнения операции сохранения в рабочем пространстве.



Приведем пример выполнения функции `gfmfdlg` (рис. 6.51):

```
» a=readfis('tip');
» gfmfdlg('#init',a)
ans =
    '3 2'    'gaussmf'    'linear'
```

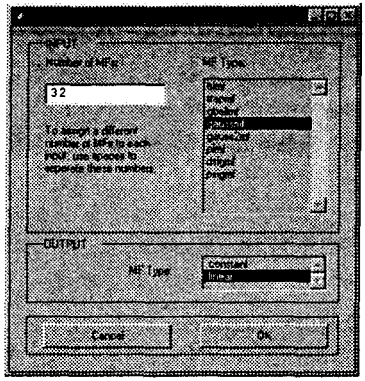


**Рис. 6.51.** Результат выполнения команды gfmfdlg

# Работа Fuzzy Logic с Simulink

Системы нечеткого вывода, созданные тем или иным образом с помощью пакета Fuzzy Logic Toolbox, допускают интеграцию с инструментами пакета Simulink, что позволяет выполнять моделирование систем в рамках последнего.

## Пример: контроль уровня воды в баке

На рис. 6.52 изображен объект управления в виде бака с водой, к которому подходят две трубы: через одну трубу, снабженную краном, вода втекает в бак, через другую — вытекает.

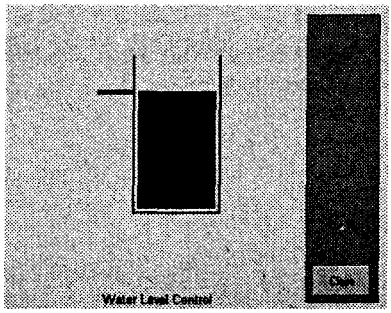


**Рис. 6.52.** *Схематическое представление объекта управления (бака с водой)*

Подачу воды в бак можно регулировать, больше или меньше открывая кран. Расход воды является неконтролируемым и зависит от

диаметра выходной трубы (он фиксирован) и от текущего уровня воды в баке. Если понимать под выходной (регулируемой) переменной уровень воды, а под регулирующим элементом — кран, то можно отметить, что подобный объект регулирования с точки зрения его математического описания является динамическим и существенно нелинейным.

Определим цель управления как установление уровня воды в баке на требуемом (изменяющемся) уровне и попробуем решить соответствующую задачу управления средствами нечеткой логики.

Очевидно, в регулятор, обеспечивающий достижение цели управления, должна поступать информация о несоответствии (разности) требуемого и фактического уровней воды, при этом данный регулятор должен вырабатывать управляющий сигнал, подаваемый на регулирующий элемент (кран).

В первом приближении функционирование регулятора можно описать набором из трех следующих правил:

1. If (level is okay) then (valve is no_change) (1);
2. If (level is low) then (valve is open_fast) (1);
3. If (level is high) then (valve is close_fast) (1);
4. If (level is okay) and (rate is positive) then (valve is close_slow) (1);
5. If (level is okay and (rate is negative) then (valve is open_slow) (1),



что в переводе означает:

1. Если (уровень соответствует заданному), то (кран без изменения) (1);
2. Если (уровень низкий), то (кран быстро_открыть) (1);
3. Если (уровень высокий), то (кран быстро_закрыть) (1);
4. Если (уровень соответствует заданному) и (его прирост — положительный), то (кран надо медленно_закрывать) (1);
5. Если (уровень соответствует заданному) и (его прирост — отрицательный), то (кран надо медленно_открывать) (1).

Модель системы управления уровнем воды в баке с нечетким регулятором, основанным на приведенных правилах, является одной из демонстрационных моделей пакетов Fuzzy Logic и Simulink. Ее можно вызвать из командной строки командой `sltank`, что приведет к открытию окна Simulink с блок-диаграммой указанной модели (рис. 6.53).

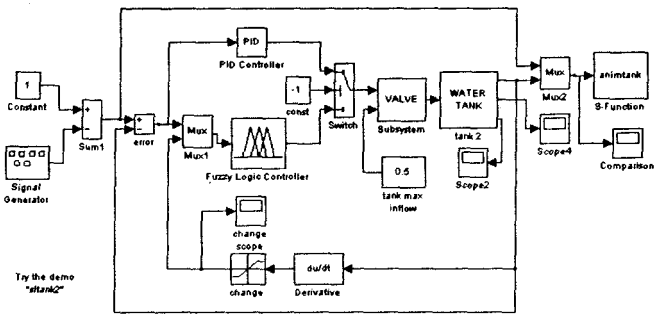


**Рис. 6.53.** Блок-диаграмма модели системы управления уровнем воды в баке с нечетким регулятором

Одновременно блок Fuzzy Logic Controller будет сопоставлен с системой нечеткого вывода, записанной в файле tank.fis.

Для изучения процесса функционирования системы необходимо нажать кнопку Start панели инструментов блок-диаграммы модели и дважды щелкнуть на блоке Comparison (Сравнение). В появившемся окне этого блока будут показаны изменяющиеся во времени сигналы заданного (последовательность импульсов прямоугольной формы) и фактического уровней воды (рис. 6.54). Как видно, переходный процесс в системе имеет апериодическую форму и заканчивается достаточно быстро, то есть качество регулирования следует признать хорошим.

Можно попытаться изменить характер функционирования системы, внося (например, с помощью рассмотренных программ с графическим интерфейсом типа FIS-редактора и т. п.) определенные изменения в систему, задаваемую файлом tank.fis (изменяя правила, функции принадлежности, метод приведения к четкости и т. д.). Разумеется, все это лучше делать, остановив процесс моделирования.

Выполнив (в режиме командной строки) команду `sltankrule`, перейдем к блок-диаграмме той же системы управления, но с нечетким регулятором с просмотрщиком правил (Fuzzy Logic Controller with Ruleviewer). Структура такого регулятора представляется в отдельном окне (рис. 6.55), если дважды щелкнуть на указанном блоке.

Запуск процесса моделирования приводит к открытию окна просмотрщика правил (рис. 6.56), наглядно иллюстрирующего процесс выработки сигнала управления (выходного сигнала регулятора).

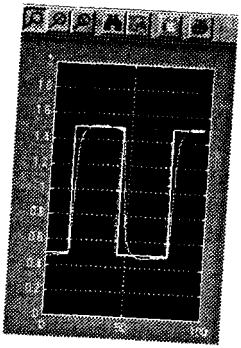

**Рис. 6.54.** Результаты моделирования системы управления с нечетким регулятором

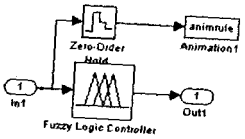


**Рис. 6.55.** Структура блока Fuzzy Logic Controller with Ruleviewer

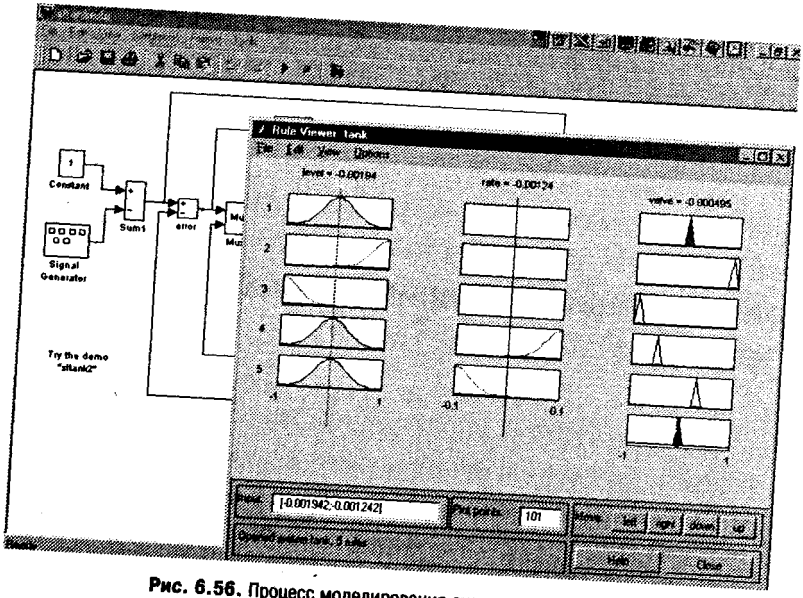


**Рис. 6.56.** Процесс моделирования системы управления с демонстрацией применения нечетких правил

### Построение нечеткой модели с использованием блоков Simulink

Для построения какой-то собственной моделирующей системы с использованием средств нечеткой логики и блоков Simulink рекомендуется просто скопировать блок Fuzzy Logic Controller из рассмотренной системы sltank (или какого-либо другого демонстрационного примера MATLAB) и поместить его в блок-диаграмму разрабатываемой системы.

Из командной строки командой `fuzblock` можно также открыть библиотеку нечетких блоков пакета Simulink (в версии MATLAB 5.3 содержащую два блока: нечеткий регулятор, Fuzzy Logic Controller, и нечеткий регулятор с просмотрщиком правил, Fuzzy Logic Controller with Ruleviewer, а в версии 5.2 еще и несколько демонстрационных блоков), которые могут быть использованы точно так же, при необходимости — с дополнительным редактированием.

Укажем, что функционирование указанных блоков осуществляется с использованием системной S-функции sffis.mex. Запись этой функции такова:

```
output = sffis(t,x,u,flag,fismat)
```

Здесь `output` — выход нечеткого регулятора, `t`, `x` и `flag` — стандартные аргументы системной функции Simulink, `fismat` — идентификатор (имя) нечеткой системы вывода, `u` — входной сигнал (вектор входных сигналов) регулятора.

По смыслу данная функция аналогична рассмотренной ранее функции `evalfis`, но она оптимизирована для работы в среде Simulink.

## Демонстрационные примеры работы с пакетом Fuzzy Logic Toolbox

Для ознакомления с пакетом Fuzzy Logic Toolbox можно использовать следующие функции (команды) в режиме командной строки:

- `defuzzdm` — обзор методов приведения к четкости (дефаззификации);
- `fcmdemo` — демонстрация алгоритма кластеризации Fuzzy c-means (двумерная графика);
- `fuzdemos` — демонстрация графического интерфейса пакета Fuzzy Logic;

- `gasdemo` — демонстрация использования аппарата гибридных сетей для решения задачи о выборе автомобиля, наиболее экономичного по расходу топлива;
- `juggler` — демонстрация системы жонглирования мячом с использованием нечеткого регулятора;
- `invkine` — демонстрация нечеткого управления движением робота-манипулятора;
- `irisfcm` — демонстрация алгоритма кластеризации Fuzzy c-means;
- `noisedm` — демонстрация решения задачи фильтрации на основе методов нечеткой логики;
- `slbb` — демонстрация задачи «шар на качелях»;
- `slcp` — демонстрация нечеткой системы управления перевернутым маятником,
- `sltank` — демонстрация системы управления уровнем воды в баке с нечетким регулятором;
- `sltankrule` — то же, что в предыдущем случае, но с дополнительным просмотром нечетких правил;
- `sltbu` — демонстрация функционирования нечеткой системы управления грузовиком.



С каждым из этих примеров связан m-файл, fis-файл или/и блок-диаграмма Simulink, запуск которых производится с помощью одной из указанных команд. Текст пояснений в примерах — на английском языке. Данные примеры доступны и через главное меню MATLAB (Help ▸ Examples and Demos, раздел Toolboxes/ Fuzzy Logic).

Дополнительную информацию можно получить, используя команду

```
help fuzzy
```

# Глава 7
# Пакет оптимизации Optimization Toolbox

## Назначение и возможности пакета

Пакет оптимизации (Optimization Toolbox) — это библиотека функций, расширяющих возможности системы MATLAB по численным вычислениям и предназначенная для решения задач оптимизации и систем нелинейных уравнений. Поддерживает основные методы оптимизации функций ряда переменных:

- Безусловная оптимизация нелинейных функций.
- Метод наименьших квадратов.
- Решение нелинейных уравнений.
- Линейное программирование.
- Квадратичное программирование.
- Условная минимизация нелинейных функций.
- Методы минимакса.
- Многокритериальная оптимизация.

Рассматриваемый пакет дает возможности решать задачи минимизации функций, нахождения решений уравнений, задачи аппроксимации («подгонки» кривых под экспериментальные данные).

Различные типы таких задач вместе с применяемыми для их решения функциями пакета Optimization Toolbox (дальнейшее изложение будет относиться к версии 2.0, используемой в составе MATLAB 5.3.1) приведены в табл. 7.1.

**Таблица 7.1.** Типы задач, решаемых средствами пакета Optimization Toolbox

| Тип задачи | Математическая запись | Функция MATLAB |
|---|---|---|
| **Задачи минимизации** | | |
| Скалярная (одномерная) минимизация | $\min_a f(a),\ a_1 < a < a_2$ | `fminbnd` |

| Тип задачи | Математическая запись | Функция MATLAB |
|---|---|---|
| Безусловная минимизация (без ограничений) | $\min_{\mathbf{x}} f(\mathbf{x})$ | `fminunc`, `fminsearch` |
| Линейное программирование | $\min_{\mathbf{x}} f^T\mathbf{x}$ при условиях $\mathbf{A}\mathbf{x} \le \mathbf{b}$, $\mathbf{Aeq}\,\mathbf{x}=\mathbf{beq}$, $\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U$ | `linprog` |
| Квадратичное программирование | $\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{f}^T\mathbf{x}$ при условиях $\mathbf{A}\mathbf{x} \le \mathbf{b}$, $\mathbf{Aeq}\,\mathbf{x}=\mathbf{beq}$, $\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U$ | `quadprog` |
| Минимизация при наличии ограничений | $\min_{\mathbf{x}} f(\mathbf{x})$ при условиях $\mathbf{c}(\mathbf{x}) \le 0$, $\mathbf{ceq}(\mathbf{x})=0$, $\mathbf{A}\mathbf{x} \le \mathbf{b}$, $\mathbf{Aeq}\,\mathbf{x}=\mathbf{beq}$, $\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U$ | `fmincon` |
| Достижение цели | $\min_{\mathbf{x},\gamma} \gamma$ при условиях $F(\mathbf{x}) - \mathbf{w}\gamma \le \mathbf{goal}$, $\mathbf{c}(\mathbf{x}) \le 0$, $\mathbf{ceq}(\mathbf{x})=0$, $\mathbf{A}\mathbf{x} \le \mathbf{b}$, $\mathbf{Aeq}\,\mathbf{x}=\mathbf{beq}$, $\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U$ | `fgoalattain` |
| Минимакс | $\min_{\mathbf{x}} \max_{\{F_i\}} \{F_i(\mathbf{x})\}$ при условиях $\mathbf{c}(\mathbf{x}) \le 0$, $\mathbf{ceq}(\mathbf{x})=0$, $\mathbf{A}\mathbf{x} \le \mathbf{b}$, $\mathbf{Aeq}\,\mathbf{x}=\mathbf{beq}$, $\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U$ | `fminimax` |
| Полубесконечная минимизация | $\min_{\mathbf{x}} f(\mathbf{x})$ при условиях $\mathbf{K}(\mathbf{x}, \mathbf{w}) \le 0$ для всех $\mathbf{w}$, $\mathbf{c}(\mathbf{x}) \le 0$, $\mathbf{ceq}(\mathbf{x})=0$, $\mathbf{A}\mathbf{x} \le \mathbf{b}$, $\mathbf{Aeq}\,\mathbf{x}=\mathbf{beq}$, $\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U$ | `fseminf` |

| Тип задачи | Математическая запись | Функция MATLAB |
|---|---|---|
| **Нахождение решений уравнений** | | |
| Линейные уравнения | $\mathbf{C}\,\mathbf{x}=\mathbf{d}$, $n$ уравнений, $n$ переменных | \ (оператор левого деления, `slash`) |
| Нелинейное уравнение одной переменной | $f(a) = 0$ | `fzero` |
| Нелинейные уравнения многих переменных | $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, $n$ уравнений, $n$ неизвестных | `fsolve` |
| **Задачи аппроксимации («подгонки» кривых)** | | |
| Линейный метод наименьших квадратов (МНК) | $\min_x \|\mathbf{C}\cdot\mathbf{x}-\mathbf{d}\|_2^2$, $m$ уравнений, $n$ переменных | \ (оператор левого деления, `backslash`) |
| Неотрицательный линейный МНК | $\min_x \|\mathbf{C}\cdot\mathbf{x}-\mathbf{d}\|_2^2$ при условии $\mathbf{x}\ge\mathbf{0}$ | `lsqnonneg` |
| Линейный МНК при наличии ограничений | $\min_x \|\mathbf{C}\cdot\mathbf{x}-\mathbf{d}\|_2^2$ при условиях $\mathbf{A}\,\mathbf{x}\le\mathbf{b}$, $\mathbf{Aeq}\,\mathbf{x}=\mathbf{beq}$, $\mathbf{x}_L\le\mathbf{x}\le\mathbf{x}_U$ | `lsqlin` |
| Нелинейный МНК | $\min_x \frac{1}{2}\|\mathbf{F}(\mathbf{x})\|_2^2 = \frac{1}{2}\sum_i f_i^2(\mathbf{x})$ при условии $\mathbf{x}_L\le\mathbf{x}\le\mathbf{x}_U$ | `lsqnonlin` |
| Нелинейная «подгонка» кривой | $\min_x \frac{1}{2}\|F(\mathbf{x},\mathbf{xdata})-\mathbf{ydata}\|_2^2$ при условии $\mathbf{x}_L\le\mathbf{x}\le\mathbf{x}_U$ | `lsqcurvefit` |

Принятые обозначения:

- $a$ — скалярный аргумент; $\mathbf{x}$, $\gamma$ — в общем случае векторные аргументы;
- $f(\mathbf{a})$, $f(\mathbf{x})$ — скалярные функции; $\mathbf{F}(\mathbf{x})$, $\mathbf{c}(\mathbf{x})$, $\mathbf{ceq}(\mathbf{x})$, $\mathbf{K}(\mathbf{x},\mathbf{w})$ — векторные функции;
- **A**, **Aeq**, **C**, **H** — матрицы;
- **b**, **beq**, **d**, **f**, **w**, **goal**, **xdata**, **ydata** — векторы;
- $\mathbf{x}_L$, $\mathbf{x}_U$ — соответственно, нижняя и верхняя границы области изменения аргумента.

Методы оптимизации достаточно полно описаны в следующих монографиях:

1. Аоки М. Введение в методы оптимизации. — М.: Наука, 1977. — 344 с.
2. Табак Д., Куо Б. Оптимальное управление и математическое программирование. — М.: Наука, 1975. — 280 с.
3. Банди Б. Методы оптимизации. Вводный курс. — М.: Радио и связь, 1988. — 128 с.
4. Банди Б. Основы линейного программирования. — М.: Радио и связь, 1989. — 176 с.
5. Каханер Д., Моулер К., Нэш С. Численные методы и математическое обеспечение. — М.: Мир, 1998. — 575 с.
6. Теория выбора и принятия решений / И. М. Макаров, Т. М. Виноградская, А. А. Рубчинский, В. Б. Соколов. — М.: Наука, 1982. — 328 с.
7. Васильков Ю. В., Василькова Н. Н. Компьютерные технологии вычислений в математическом моделировании. — М.: Финансы и статистика, 1999. — 256 с.

## Применяемые алгоритмы



В рамках пакета Optimization Toolbox все задачи оптимизации делятся на две группы: задачи малой и средней размерности и задачи большой размерности. Соответственно, такое же деление принято для алгоритмов решения данных задач. Разумеется, это не означает, что для решения задач средней размерности нельзя применять алгоритмы большой размерности и наоборот. Просто алгоритмы той или иной группы более эффективны для задач своей размерности.

В пакете Optimization Toolbox реализован широкий набор алгоритмов для решения задач оптимизации средней и малой размерности. Основными для задач без ограничений являются симплексный метод Нелдера—Мида и квазиньютоновские методы.

Для решения задач с ограничениями, минимакса, достижения цели и полубесконечной оптимизации использованы алгоритмы квадратичного программирования.

Задачи, сводящиеся к нелинейным МНК, решаются с помощью алгоритмов Ньютона—Рафсона и Левенберга—Марквардта.

Вспомогательные процедуры одномерной (скалярной) оптимизации используют алгоритмы квадратичной (параболической) и кубической интерполяции.

Рассмотрим перечисленные алгоритмы подробнее.

## Общая формулировка задачи параметрической оптимизации

Задача параметрической оптимизации формулируется как задача нахождения набора параметров $\mathbf{x} = \{x_1, x_2, ..., x_n\}$, который является оптимальным в смысле некоторого критерия. В простейшем случае такая задача сводится к минимизации или максимизации некоторой (целевой) функции без каких-либо ограничений. В более сложных ситуациях на отмеченные параметры могут быть наложены некоторые ограничения в виде равенств $g_i(\mathbf{x}) = 0$ ($i = 1, 2, ..., m_e$), неравенств $g_i(\mathbf{x}) \le 0$ ($i = m_e+1, ..., m$) и/или параметрических границ $x_L$, $x_U$.

Общая формулировка задачи параметрической оптимизации представляется следующим образом: требуется найти вектор $x$, обеспечивающий

$$\min_{\mathbf{x} \in R^n} f(\mathbf{x})$$

при ограничениях

$$g_i(x) = 0,\ i = 1, 2, ..., m_e,$$

$$g_i(\mathbf{x}) \le 0,\ i = m_e+1, ..., m,$$

$$\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U,$$

где $\mathbf{x}$ — вектор оптимизируемых параметров ($\mathbf{x} \in R^n$), $f(\mathbf{x})$ — скалярная целевая функция (критерий) векторного аргумента ($f(\mathbf{x})$: $R^n \to R$), $g_i(\mathbf{x})$ — также некоторые скалярные функции векторного аргумента (заметим, что задача максимизации сводится к задаче минимизации заменой $f(\mathbf{x})$ на $-f(\mathbf{x})$).

Эффективность и точность решения данной задачи зависит как от числа параметров и ограничений, так и от вида целевой функции. При линейных ограничениях и целевой функции приведенная задача оптимизации называется задачей линейного программирования, при линейных ограничениях, но при квадратичной (по аргументам) целевой функции — задачей квадратичного программирования, в общем случае это задача нелинейного программирования.

## Безусловная оптимизация

Существующие алгоритмы безусловной оптимизации могут быть разделены на две группы — алгоритмы, базирующиеся на использовании производных минимизируемой функции (градиентные и методы второго порядка), и алгоритмы, использующие только значения функции (безградиентные).

*Безградиентные методы* (например, симплексный метод Нелдера–Мида) более пригодны для задач, где минимизируемая функция является существенно нелинейной или имеет разрывы. Градиентные методы (методы первого порядка) обычно эффективны в случаях целевых функций, непрерывных вместе с первыми производными. Методы второго порядка, такие как метод Ньютона, применяются реже, поскольку требуют больших вычислительных затрат для расчета матриц вторых производных.

*Градиентные методы* используют информацию о наклоне функции для выбора направления поиска экстремума. В одном из таких методов – наискорейшего спуска – на каждой итерации движение к точке минимума осуществляется в направлении $-\nabla f(\mathbf{x})$ (где $\nabla f(\mathbf{x})$ – вектор-градиент целевой функции $f(\mathbf{x})$). Этот метод весьма неэффективен в ситуациях, когда поверхность целевой функции имеет узкие «овраги», как, например, у известной функции Розенброка

$$f(\mathbf{x})=100(x_1 - x_2^2)^2 + (1 - x_1)^2.$$

Поверхность этой функции приведена на рис. 7.1.

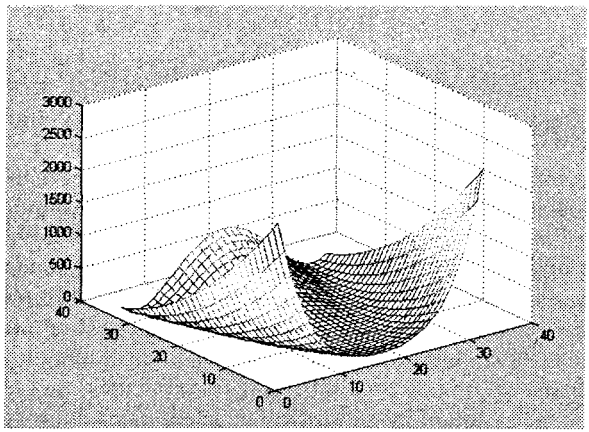


**Рис. 7.1.** Графическое представление функции Розенброка

Минимальное значение данной функции, как нетрудно видеть, равно нулю при $x_1 = x_2 = 1$. Между тем численные эксперименты показывают, что зачастую метод наискорейшего спуска не обеспечивает нахождение точки экстремума даже после сотен и тысяч итераций.

Отметим, что указанную функцию из-за своеобразной формы ее линий равного уровня часто называют «банановой» функцией (как видно, банановыми бывают не только республики!) и используют как тестовую при проверке эффективности различных оптимизационных алгоритмов.

*Квазиньютоновские алгоритмы.* Среди алгоритмов, использующих информацию о градиенте, наиболее распространенными являются квазиньютоновские. В этих (итерационных) алгоритмах целевая функция в окрестностях произвольной точки аппроксимируется квадратичной функцией, при этом на каждой итерации решается задача локальной минимизации

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x} + b,$$

где $\mathbf{H}$ — симметричная и положительно определенная матрица вторых частных и смешанных производных (матрица Гессе, или гессиан), $\mathbf{c}$ — постоянный вектор, $b$ — константа.

Оптимальное решение приведенной задачи соответствует нулевым значениям первых производных, то есть

$$\nabla f(\mathbf{x}^*) = \mathbf{H}\mathbf{x}^* + \mathbf{c} = 0,$$

откуда

$$\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{c}.$$

## Ньютоновские алгоритмы

Ньютоновские алгоритмы (в отличие от квазиньютоновских) непосредственно вычисляют $\mathbf{H}$ (как отмечалось, прямое вычисление матрицы $\mathbf{H}$ требует больших вычислительных затрат) и осуществляют движение в рассчитанном на очередной итерации направлении уменьшения целевой функции до достижения минимума (с использованием методов одномерного поиска). В квазиньютоновских алгоритмах такое вычисление не производится, а используется некоторая аппроксимация $\mathbf{H}$.

Среди подобных алгоритмов одним из наиболее популярных и используемым в пакете Optimization Toolbox является так называемый BFGS-алгоритм, получивший свое название по фамилиям предложивших его авторов (Broyden, Fletcher, Goldfarb, Shanno), в котором аппроксимация $\mathbf{H}$ производится итерационно, по формуле

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{q}_k\mathbf{q}_k^T}{\mathbf{q}_k^T\mathbf{s}_k} - \frac{\mathbf{H}_k\mathbf{s}_k\mathbf{s}_k^T\mathbf{H}_k}{\mathbf{s}_k^T\mathbf{H}_k\mathbf{s}_k},$$

где

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k,$$
$$\mathbf{q}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$$

Заметим, что более удобно иметь аппроксимацию не матрицы **H**, а обратной к ней матрицы $\mathbf{H}^{-1}$; приведенное рекуррентное соотношение подобную замену допускает, при этом сам алгоритм становится практически идентичным хорошо известному отечественным исследователям *алгоритму Давидона—Флетчера—Пауэлла*, за тем исключением, что в последнем векторы **q** заменены на векторы **s** и наоборот.

## Алгоритмы Ньютона—Гаусса и Левенберга—Марквардта

Алгоритмы Ньютона—Гаусса и Левенберга—Марквардта используются в функциях рассматриваемого пакета, предназначенных для решения задачи нелинейного метода наименьших квадратов (МНК). При отсутствии ограничений указанная задача формулируется следующим образом:

$$\min_{\mathbf{x}} \frac{1}{2}\|F(\mathbf{x})\|_2^2 = \frac{1}{2}\sum_i F_i^2(\mathbf{x}).$$

Скалярная оптимизация в данных алгоритмах производится, соответственно, вдоль направлений

- $d_k = -(\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k))^{-1}\mathbf{J}(\mathbf{x}_k)\mathbf{F}(\mathbf{x}_k)$ — для алгоритма Ньютона—Гаусса,
- $d_k = -(\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \lambda_k \mathbf{I})^{-1}\mathbf{J}(\mathbf{x}_k)\mathbf{F}(\mathbf{x}_k)$ — для алгоритма Левенберга—Марквардта,

где $\mathbf{J}(\mathbf{x})$ — матрица-якобиан размером $m \times n$ (в документации к пакету Optimization Toolbox под якобианом понимается матрица первых частных производных вектор-функции $\mathbf{F}(\mathbf{x})$ по векторному аргументу **x**, а не определитель этой матрицы, как обычно принято в математической литературе; будем придерживаться терминологии документации пакета), $\lambda_k$ — параметр алгоритма, определяемый в процессе линейной (скалярной) оптимизации вдоль выбранного направления.

## Минимизация при наличии ограничений

В задачах оптимизации с ограничениями (см. табл. 7.1) обычный подход к нахождению решения состоит в замене исходной задачи

с ограничениями на задачу без ограничений (задачу безусловной оптимизации), например, с помощью метода штрафных функций. В настоящее время, однако, более эффективным считается применение так называемых уравнений Куна—Таккера, которые на основании вышеприведенной формулировки задачи параметрической оптимизации и при некоторых дополнительных предположениях о характере ограничений записываются в виде

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\mathbf{x}^*) = \mathbf{0},$$

$$\nabla g_i(\mathbf{x}^*) = \mathbf{0}, \quad i = 1, 2, ..., m_e,$$

$$\lambda_i^* = 0, \quad i = m_e+1, ..., m,$$

где $\lambda_i$ — множители Лагранжа.

Для решения данных уравнений в пакете Optimization Toolbox использован алгоритм так называемого последовательного квадратичного программирования (в оригинале — Sequential Quadratic Programming, или SQP), представляющий собой, по сути, разновидность квазиньютоновского метода. Основная идея SQP заключается в применении квадратичной аппроксимации функции Лагранжа (учитывающей ограничения)

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\mathbf{x}),$$

так что на каждой итерации решается задача оптимизации

$$\min_{d \in R^n} \frac{1}{2} \mathbf{d}^T \mathbf{H}_k \mathbf{d} + \nabla f^T(\mathbf{x}_k)\mathbf{d},$$

$$\nabla g_i^T(\mathbf{x}_k)\mathbf{d} + g_i(\mathbf{x}_k) = 0, \quad i = 1, 2, ..., m_e,$$

$$\nabla g_i^T(\mathbf{x}_k)\mathbf{d} + g_i(\mathbf{x}_k) \le 0, \quad i = m_e+1, ..., m.$$

В последней формулировке данная задача может быть решена любым методом решения задач квадратичного программирования. В пакете Optimization Toolbox для этой цели использован комбинированный алгоритм, объединяющий алгоритм BFGS и так называемый метод проекций.

## Многокритериальная оптимизация

Достаточно часто в реальных ситуациях качество работы исследуемого объекта или системы оценивается не единственным критерием или показателем качества, а совокупностью таких критериев, причем представляющихся одинаково значимыми. Это приводит к

задаче оптимизации с векторной целевой функцией $\mathbf{F(x)} = \{\mathbf{F}_1(\mathbf{x}), \mathbf{F}_2(\mathbf{x}),..., \mathbf{F}_m(\mathbf{x})\}$ и формулировкой

$$\min_{\mathbf{x}\in R^n} \mathbf{F(x)}$$

при ограничениях

$$g_i(\mathbf{x}) = 0, \quad i = 1, 2, ..., m_e,$$

$$g_i(\mathbf{x}) \le 0, \quad i = m_e+1, ..., m,$$

$$\mathbf{x}_L \le \mathbf{x} \le \mathbf{x}_U,$$

получившей название задачи многокритериальной, или векторной, оптимизации.

Известно, что решение подобной задачи сводится к нахождению множества точек неулучшаемых решений (Парето-множества), для чего используются такие, например, методы, как метод взвешенной суммы частных критериев или метод ε-ограничения.

В рассматриваемом пакете для решения задачи многокритериальной оптимизации применен так называемый *метод достижения цели*, предложенный Gembicki и математически описываемый соотношениями

$$\min_{\gamma\in R,\ \mathbf{x}\in\Omega} \gamma$$

при ограничениях

$$F_i(\mathbf{x}) - w_i\gamma \le F_i^*, \qquad i = 1, 2, ..., m,$$

где $\Omega$ — область допустимых значений $\mathbf{x}$, $w_i$ — некоторые весовые коэффициенты, $F_i^*$ — некоторые устанавливаемые «цели» (goals).

В приведенной формулировке задача подобна задаче однокритериальной оптимизации и может решаться с помощью перечисленных алгоритмов.

## Алгоритмы большой размерности

Задачи оптимизации с большим числом оптимизируемых факторов (десятки, сотни, тысячи) и/или сложным, нелинейным характером целевой функции достаточно плохо решаются рассмотренными методами ввиду существенных вычислительных затрат (например, при определении аппроксимации матрицы Гессе), что привело к выделению в отдельную группу алгоритмов, оказывающихся эффективными в подобных ситуациях, — алгоритмов большой размерности.

Данные алгоритмы основаны на идее так называемой области доверия — области $N$ вблизи некоторой точки $\mathbf{x}$, в которой рассматриваемая целевая функция $f(\mathbf{x})$ может быть адекватно аппроксимирована более простой функцией $q(\mathbf{s})$, так что исходная задача оптимизации сводится к задаче

$$\min_{\mathbf{s}\in N} q(\mathbf{s}).$$

Текущая точка $\mathbf{x}$ обновляется, то есть заменяется точкой $\mathbf{x} + \mathbf{s}$, если $f(\mathbf{x} + \mathbf{s}) < f(\mathbf{x})$, и не изменяется в противоположном случае.

В стандартном подходе предполагается, что как функция $q(\mathbf{s})$, так и область доверия описываются квадратичными функциями, что приводит к задаче оптимизации

$$\min_{\mathbf{s}} \frac{1}{2}\mathbf{s}^T\mathbf{H}_k\mathbf{s} + \mathbf{s}^T\mathbf{G}$$

при условии

$$\|\mathbf{D}\,\mathbf{s}\| \le \Delta,$$

где $\mathbf{G} = \nabla f(\mathbf{x})$ — градиент в текущей точке $\mathbf{x}$, $\mathbf{D}$ — некоторая диагональная матрица весовых коэффициентов, $\|\bullet\|$ — обозначение нормы, $\Delta$ — положительный скаляр.

В пакете Optimization Toolbox используется двумерный вектор $\mathbf{s} = [\mathbf{s}_1, \mathbf{s}_2]$, в котором элемент $\mathbf{s}_1$ соответствует направлению градиента $\mathbf{G}$, а элемент $\mathbf{s}_2$ — направлению, выбираемому по одному из соотношений

$$\mathbf{H}\mathbf{s}_2 = -\mathbf{G}, \quad \mathbf{s}_2^T\mathbf{H}\mathbf{s}_2 < 0.$$

Очевидно, если направления $\mathbf{s}_1$ и $\mathbf{s}_2$ установлены, локальная оптимизация производится в двумерной области, что не вызывает никаких проблем с нахождением соответствующего решения. Наибольшую сложность здесь вызывает именно определение указанных направлений. Решение данной вспомогательной задачи производится модификацией метода сопряженных градиентов, так называемого метода PCG (Preconditioned Conjugate Gradients).

Для решения частных задач большой размерности (задачи с линейными ограничениями, линейного и нелинейного МНК, задач линейного и квадратичного программирования) в рассматриваемом пакете применены специальные модификации изложенного подхода, в том числе использующие операции с разреженными матрицами, подробности о которых можно выяснить в справочной системе MATLAB.

# Функции пакета Optimization Toolbox

Функции пакета разделяются на пять групп: функции минимизации, решения уравнений, наименьших квадратов (подбора кривых), утилиты и демонстрационные.

## Функции минимизации

В данную группу входят 9 функций.

### Функция fgoalattain

Функция `fgoalattain` служит для решения задачи векторной оптимизации методом «достижения цели». Она записывается в следующем виде:

- `x = fgoalattain(fun,x0,goal,weight)` — возвращает решение задачи многомерной оптимизации при заданных векторе целевых функций `fun`, начальном приближении `x0`, заданных векторе целей `goal` и векторе весов `weight`;
- `x = fgoalattain(fun,x0,goal,weight,A,b)` — то же, что и предыдущая функция, но при наличии ограничений в форме линейных неравенств $\mathbf{A}\,\mathbf{x} \le \mathbf{b}$;
- `x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq)` — то же, что и предыдущая функция, но при наличии дополнительных ограничений в форме равенств $\mathbf{Aeq}\,\mathbf{x} = \mathbf{beq}$; если ограничения в форме неравенств отсутствуют, задаются `A=[ ]` и `b=[ ]`;
- `x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub)` — то же, что и предыдущая функция, но при наличии дополнительных граничных ограничений $\mathbf{lb} \le \mathbf{x} \le \mathbf{ub}$;
- `x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub,nonlcon)` — то же, что и предыдущая функция, но при наличии дополнительных ограничений в форме нелинейных неравенств или равенств $\mathbf{c}(\mathbf{x}) \le \mathbf{0}$, $\mathbf{ceq}(\mathbf{x}) = \mathbf{0}$, при отсутствии граничных ограничений задаются `lb=[ ]` и/или `ub=[ ]`;
- `x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,...lb,ub,nonlcon,options)` — то же, что и предыдущая функция, но при задании (изменении) опций (см. ниже);
- `x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,...lb,ub,nonlcon,options,P1,P2,...)` — то же, что и предыдущая функция, но при задании параметров `P1`, `P2`, ..., относящихся к функциям-аргументам;

- `[x,fval] = fgoalattain(...)` — возвращается не только оптимальное значение векторного аргумента, но и значение целевой функции в точке минимума `fval`;
- `[x,fval,attainfactor] = fgoalattain(...)` — то же, что и предыдущая функция, но возвращается еще и коэффициент достижения цели `attainfactor`;
- `[x,fval,attainfactor,exitflag] = fgoalattain(...)` — то же, что и предыдущая функция, но возвращается еще информация о характере завершения вычислений `exitflag`;
- `[x,fval,attainfactor,exitflag,output] = fgoalattain(...)` — то же, что и предыдущая функция, но возвращается еще информация о результатах оптимизации (выходная структура) `output`;
- `[x,fval,attainfactor,exitflag,output,lambda] = fgoalattain(...)` — то же, что и предыдущая функция, но возвращаются еще множители Лагранжа `lambda`.

Аргументы функции:

- `fun` — векторная функция векторного аргумента. Должна быть задана либо с помощью функции `inline`, например:

```
» fun = inline('sin(x.*x)');
```

либо как m-файл, например:

```
function F = myfun(x)
F = ...
```

Если задано вычисление градиента (функцией `options = optimset('GradObj', 'on')`), то m-файл должен возвращать не только значение функции **F**, но и значения градиентов **G**:

```
function [F,G] = myfun(x)
F = ... % Вычисление векторной функции
G = ... % Вычисление градиента
```

- `goal` — вектор задаваемых целевых значений той же размерности, что и вектор `fun`;
- `weight` — вектор весов той же размерности, что и вектор целей, часто принимается равным `abs(goal)`.
- `nonlcon` — функция, возвращающая значения функций-ограничений, а при необходимости (если задано `options = optimset('GradConstr','on')`) и их градиентов; должна быть оформлена в виде m-файла, например:

```
function [c,ceq] = mycon(x)
c = ...  % Вычисление левых частей нелинейных неравенств
```

```
ceq = ... % Вычисление левых частей нелинейных равенств
function [c,ceq,GC,GCeq] = mycon(x)
c = ... % Вычисление левых частей нелинейных неравенств
ceq = ... % Вычисление левых частей нелинейных равенств
GC = ... % Градиенты неравенств
GCeq = ... % Градиенты равенств
```

- `Options` — опции (их можно изменять, используя функцию `optimset`):
  - `DerivativeCheck` — задает проверку соответствия производных, определенных пользователем, их вычисленным оценкам в виде первых разностей;
  - `Diagnostics` — вывод диагностической информации о минимизируемой функции;
  - `DiffMaxChange` — максимальные значения изменений переменных при определении первых разностей;
  - `DiffMinChange` — минимальные значения изменений переменных при определении первых разностей;
  - `Display` — уровень отображения: `'off'` — вывод информации отсутствует, `'iter'` — вывод информации о поиске решения на каждой итерации, `'final'` — вывод только итоговой информации;
  - `GoalExactAchieve` — определяет количество целей, которые должны быть достигнуты «точно»;
  - `GradConstr` — использование градиентов для ограничений (опция имеет смысл в случае применения аргумента `nonlcon`, см. выше), возможные значения — `'off'` и `'on'`;
  - `GradObj` — использование градиента для целевой функции, определяемого пользователем (возможные значения — `'off'` и `'on'`);
  - `MaxFunEvals` — максимальное число вычислений функции;
  - `MaxIter` — максимальное допустимое число итераций;
  - `MeritFunction` — устанавливает вид функции оценки качества достижения цели (возможные значения `'multiobj'` или `'singleobj'`);
  - `TolCon` — допуск останова вычислений при нарушении ограничений;
  - `TolFun` — допуск останова вычислений по величине изменений функции;
  - `TolX` — допуск останова вычислений по величине изменений `x`;

- `attainfactor` — коэффициент достижения цели, усредненное значение несоответствий заданным целям, выраженное в долевом (процентном) виде. Если данный коэффициент отрицательный, цели были «перекрыты», если положительный — цели не достигнуты;
- `exitflag` — информация о характере завершения вычислений: если эта величина положительна, то вычисления завершились нахождением решения x, если она равна нулю, то останов произошел в результате выполнения предельного числа итераций, если данная величина отрицательна, то решение не найдено;
- `lambda` — множители Лагранжа, соответственно, для различных типов ограничений:
  - `lambda.lower` — для нижней границы lb;
  - `lambda.upper` — для верхней границы ub;
  - `lambda.ineqlin` — для линейных неравенств;
  - `lambda.eqlin` — для линейных равенств;
  - `lambda.ineqnonlin` — для нелинейных неравенств;
  - `lambda.eqnonlin` — для нелинейных равенств;
- `output` — информация о результатах оптимизации:
  - `output.iterations` — число выполненных итераций;
  - `output.funcCount` — число вычислений функции;
  - `output.algorithm` — используемый алгоритм.



### Функция fminbnd

`fminbnd` — функция скалярной нелинейной минимизации с ограничениями вида $x_1 < x < x_2$. Алгоритм базируется на методе золотого сечения и квадратичной (параболической) интерполяции. Запись функции:

```
x = fminbnd(fun,x1,x2)
x = fminbnd(fun,x1,x2,options)
x = fminbnd(fun,x1,x2,options,P1,P2,...)
[x,fval] = fminbnd(...)
[x,fval,exitflag] = fminbnd(...)
[x,fval,exitflag,output] = fminbnd(...)
```

Аргументы и возвращаемые величины практически аналогичны рассмотренным для предыдущей функции за тем исключением, что число опций здесь меньше: возможны только опции `Display`, `MaxFunEvals`, `MaxIter` и `TolX`.

Приведем несколько примеров.

Точка минимума функции $\sin(x)$ в интервале $(0, 2\pi)$ определяется следующим образом:

```
» x = fminbnd('sin',0,2*pi)
Optimization terminated successfully:
 the current x satisfies the termination criteria using OPTIONS.TolX
 of 1.000000e-004
x =
    4.7124
```

Значение функции при этом

```
» y = sin(x)
y =
   -1.0000
```

Для минимизации функции $f(x) = (x - 3)^2 - 1$ на интервале $(0, 5)$ сначала необходимо представить данную функцию в виде m-файла:

```
function f = myfun(x)
f = (x-3).^2-1;
```

а затем использовать функцию **fminbnd**:

```
» x = fminbnd('myfun',0,5)
Optimization terminated successfully:
 the current x satisfies the termination criteria using OPTIONS.TolX
 of 1.000000e-004
x =
     3
```



### Функция fmincon

**fmincon** — функция поиска минимума скалярной функции многих переменных при наличии ограничений вида

$$\mathbf{c(x)} \le 0, \quad \mathbf{ceq(x)} = 0,$$

$$\mathbf{A\,x} \le \mathbf{b}, \quad \mathbf{Aeq\,x} = \mathbf{beq},$$

$$\mathbf{lb} \le \mathbf{x} \le \mathbf{ub}$$

(задача нелинейного программирования). Функция записывается в виде

```
x = fmincon(fun,x0,A,b)
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2, …)
[x,fval] = fmincon(…)
```

```
[x,fval,exitflag] = fmincon(…)
[x,fval,exitflag,output] = fmincon(…)
[x,fval,exitflag,output,lambda] = fmincon(…)
[x,fval,exitflag,output,lambda,grad] = fmincon(…)
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(…)
```

Аргументы и возвращаемые величины практически аналогичны рассмотренным для функции `fgoalattain` за следующими исключениями:

1) имеется дополнительная возвращаемая величина `grad` — градиент функции в точке минимума;
2) имеется возможность задания вычисления гессиана **H** (вводом функции `options = optimset('Hessian','on')`, что должно быть отражено в m-файле:

```
function [f,g,H] = myfun(x)
f = … % Вычисление целевой функции
g = … % Вычисление градиента
H = … % Вычисление гессиана
```

3) имеется дополнительная возвращаемая величина `hessian` — гессиан **H** функции в точке минимума;
4) возможны дополнительные опции и имеются различия в их использовании:
   - `LargeScale` — может принимать значения `'off'` (по умолчанию) и `'on'`. В первом случае используется алгоритм средней размерности, во втором — алгоритм большой размерности.

   Следующие опции используются только при работе с алгоритмом средней размерности (описание см. выше):
   - `DerivativeCheck`;
   - `DiffMaxChange`;
   - `DiffMinChange`;
   - `LineSearchType` — задание вида алгоритма одномерной оптимизации.

   Опции, используемые только в алгоритме большой размерности:
   - `Hessian` — гессиан (в случае матрицы Гессе, задаваемой пользователем, — см. выше);
   - `HessPattern` — задание гессиана как разреженной матрицы (это может привести к существенному ускорению поиска минимума);
   - `MaxPCGIter` — максимальное число итераций PCG-алгоритма (preconditioned conjugate gradient, см. выше);

- `PrecondBandWidth` — верхняя величина начальных условий для PCG-алгоритма;
- `TolPCG` — допуск на завершение PCG-итераций;
- `TypicalX` — типовые величины `x`;

5) возвращаемая величина `output` в данном случае имеет дополнительные компоненты:
   - `output.cgiterations` — число PCG-итераций (только при использовании алгоритма большой размерности);
   - `output.stepsize` — величина конечного шага поиска (только при использовании алгоритма средней размерности);
   - `output.firstorderopt` — мера оптимальности первого порядка (норма вектора градиента в точке минимума) — только при использовании алгоритма большой размерности).

**Пример.** Пусть требуется найти минимум функции $f(\mathbf{x}) = -x_1x_2x_3$ при начальном значении $\mathbf{x} = [10; 10; 10]$ и при наличии ограничений $0 \le x_1 + 2x_2 + 2x_3 \le 72$.

*Решение.* Вначале создадим m-файл, определяющий целевую функцию:

```
function f=myfun(x)
f =-x(1)*x(2)*x(3);
```

Затем запишем ограничения в виде неравенств:

$$-x_1 - 2x_2 - 2x_3 \le 0,$$

$$x_1 + 2x_2 + 2x_3 \le 72,$$

или в матричной форме:

$$\mathbf{A}\,\mathbf{x} \le \boldsymbol{b},$$

где

$$\mathbf{A} = \begin{bmatrix} -1 & -2 & -2 \\ 1 & 2 & 2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 0 \\ 72 \end{bmatrix}.$$

Теперь нахождение решения представляется следующим образом:

```
» A=[-1 -2 -2; 1 2 2];
» b=[0;72];
» x0 = [10; 10; 10]; % Стартовое значение
» [x,fval] = fmincon('myfun',x0,A,b)
x =
   24.0000
   12.0000
   12.0000
```

```
fval =
 -3.4560e+003
```

**Функция fminimax**

fminimax — функция для решения минимаксных задач (см. табл. 7.1):

$$\min_{\mathbf{x}} \max_{\{F_i\}} \{F_i(\mathbf{x})\} \quad \text{при наличии ограничений}$$

$$\mathbf{c}(\mathbf{x}) \le \mathbf{0}, \quad \mathbf{ceq}(\mathbf{x}) = \mathbf{0},$$

$$\mathbf{A}\,\mathbf{x} \le \mathbf{b}, \quad \mathbf{Aeq}\,\mathbf{x} = \mathbf{beq},$$

$$\mathbf{lb} \le \mathbf{x} \le \mathbf{ub}.$$

Запись функции:

- x = fminimax(fun,x0) — возвращает значение x, минимизирующее максимальное значение функций из заданного набора fun при стартовой точке поиска x0.

Другие представления аналогичны рассмотренным для функции fgoalattain:

```
x = fminimax(fun,x0,A,b)
x = fminimax(fun,x0,A,b,Aeq,beq)
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub)
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon) -
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,...)
[x,fval] = fminimax(...) -
[x,fval,maxfval] = fminimax(...)
[x,fval,maxfval,exitflag] = fminimax(...)
[x,fval,maxfval,exitflag,output] = fminimax(...)
[x,fval,maxfval,exitflag,output,lambda] = fminimax(...)
```

Аргументы функции идентичны рассмотренным для функции fgoalattain.

**Пример.** Пусть требуется решить минимаксную задачу для набора из 5 функций $[f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x}), f_4(\mathbf{x}), f_5(\mathbf{x})]$, где

$$f_1(x) = 2x_1^2 + x_2^2 - 48x_1 - 40x_2 + 304,$$
$$f_2(x) = -x_1^2 - 3x_2^2,$$
$$f_3(x) = x_1 + 3x_2 - 18,$$
$$f_4(x) = -x_1 - x_2,$$
$$f_5(x) = x_1 + x_2 - 8.$$

Для получения решения сначала представим данные функции в виде m-файла

```
function f = myfun(x)
f(1)= 2*x(1)^2+x(2)^2-48*x(1)-40*x(2)+304;
f(2)= -x(1)^2-3*x(2)^2;
f(3)= x(1)+3*x(2)-18;
f(4)= -x(1)-x(2);
f(5)= x(1)+x(2)-8;
```

Затем используем функцию **fminimax**:

```
» x0 = [0.1; 0.1]; % Стартовое значение
» [x,fval] = fminimax('myfun',x0)
```

Результаты вычислений:

```
Optimization terminated successfully:
 Magnitude of directional derivative in search direction
  less than 2*options.TolFun and maximum constraint violation
  is less than options.TolCon
Active Constraints:
     1
     5
x =
    4.0000
    4.0000
fval =
    0.0000  -64.0000   -2.0000   -8.0000   -0.0000
```

Решение — **x = [4, 4]**.

7

### Функция fminsearch

Функция **fminsearch** позволяет найти минимум функции нескольких переменных без ограничений, то есть решение задачи безусловной оптимизации с использованием симплексного метода.

Формы записи, аргументы и возвращаемые величины аналогичны рассмотренным ранее:

```
x = fminsearch(fun,x0)
x = fminsearch(fun,x0,options)
x = fminsearch(fun,x0,options,P1,P2,…)
[x,fval] = fminsearch(…)
[x,fval,exitflag] = fminsearch(…)
[x,fval,exitflag,output] = fminsearch(…)
```

**Пример.** Ниже демонстрируется нахождение минимума функции одного аргумента $f(x) = \sin(x) + 3$:

```
» f = inline('sin(x)+3');
» x = fminsearch(f,2)
```

```
Optimization terminated successfully:
 the current x satisfies the termination criteria using OPTIONS.TolX
 of 1.000000e-004
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun
 of 1.000000e-004
x =
    4.7124
```

### Функция fminunc

Функция fminunc предназначена для тех же целей, что и предыдущая функция, но, в отличие от последней, имеет большее число представлений:

```
x = fminunc(fun,x0)
x = fminunc(fun,x0,options)
x = fminunc(fun,x0,options,P1,P2,...)
[x,fval] = fminunc(...)
[x,fval,exitflag] = fminunc(...)
[x,fval,exitflag,output] = fminunc(...)
[x,fval,exitflag,output,grad] = fminunc(...)
[x,fval,exitflag,output,grad,hessian] = fminunc(...)
```

Набор опций в данном случае такой же, как и у функции fmincon, то есть здесь возможно использование как алгоритма средней размерности, так и алгоритма большой размерности.

**Пример 1.** Найдем минимум функции

$$f(\mathbf{x}) = 3x_1^2 + 2x_1x_2 + x_2^2.$$

*Способ 1.* Создадим m-файл

```
function f = myfun(x)
f = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;
```

Затем используем рассматриваемую функцию при начальном значении [1, 1]. Результат возвращается буквально после пары итераций:

```
» x0 = [1,1];
» [x,fval] = fminunc('myfun',x0)
x =
  1.0e-008 *
   -0.8356    0.2899
fval =
  1.6943e-016
```

Как видно, найденное значение достаточно близко к истинной точке минимума [0, 0].

*Способ 2.* Проделаем то же самое, но с заданием информации о градиенте целевой функции. Вот соответствующий m-файл:

```
function [f,g] = myfun(x)
f = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;
if nargout > 1 % Проверка количества аргументов функции
g(1) = 6*x(1)+2*x(2);
g(2) = 2*x(1)+2*x(2);
```

Решение задачи:

```
» % Разрешение использования градиента пользователя
» options = optimset('GradObj','on');
» x0 = [1,1];% Стартовое значение
» [x,fval] = fminunc('myfun',x0,options) % Нахождение решения
x =
  1.0e-015 *
    0.1110    0.4441
fval =
  3.3280e-031
```

Как видите, в этом случае решение найдено более точно.

**Пример 2.** Найдем минимум функции $f(x) = \sin(x) + 3$:

```
» f = inline('sin(x)+3');
» x = fminunc(f,4)
x =
    4.7124
```

Здесь результат получился совпадающим с найденным при помощи функции **fminsearch** (см. выше).



#### Функция fseminf

Функция **fseminf** предназначена для нахождения решения задачи полубесконечной минимизации с ограничениями (см. табл. 7.1).

Запись:

```
x = fseminf(fun,x0,ntheta,seminfcon)
x = fseminf(fun,x0,ntheta,seminfcon,A,b)
x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq)
x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq,lb,ub)
x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq,lb,ub,options)
x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq,…lb,ub,options,P1,P2,…)
[x,fval] = fseminf(…)
[x,fval,exitflag] = fseminf(…)
[x,fval,exitflag,output] = fseminf(…)
[x,fval,exitflag,output,lambda] = fseminf(…)
```

*Описание.* Аргументы и возвращаемые величины данной функции в основном аналогичны ранее рассмотренным. Добавлены лишь аргументы, отражающие специфику задачи:

- ntheta — число полубесконечных ограничений вида $K_i(\mathbf{x}, w_i) \le 0$, где переменные $w_i$ — некоторые задаваемые векторы;
- seminfcon — функция (имя функции), возвращающая значения векторов ограничений при заданном x и оформленная в виде соответствующего m-файла, например так (файл этого примера должен иметь имя myinfcon):

```
function [c,ceq,K1,K2,...,Kntheta,S] = myinfcon(x,S)
if isnan(S(1,1)),% Начальный интервал дискретизации

S = ...% S имеет ntheta строк и 2 столбца
end
w1 = ...% Набор дискретных значений аргумента w1
w2 = ...% Набор дискретных значений аргумента w2
...
wntheta = ... % Набор дискретных значений аргумента wntheta
K1 = ... % Первое полубесконечное ограничение, зависящее от x и w1
K2 = ... % Второе полубесконечное ограничение, зависящее от x и w2
...
% Последнее полубесконечное ограничение, зависящее от x и wntheta
Kntheta = ...
c = ... % Вычисление левой части нелинейного неравенства
ceq = ... % Вычисление левой части нелинейного равенства
```

Здесь S — рекомендуемый интервал дискретизации (для расчета левых частей полубесконечных ограничений), соответственно, для $w_1$, $w_2$, ..., $w_{ntheta}$; может быть опущен.

**Пример.** Рассмотрим задачу минимизации функции

$$f(\mathbf{x}) = (x_1 - 0{,}5)^2 + (x_2 - 0{,}5)^2 + (x_3 - 0{,}5)^2$$

при наличии ограничений

$$K_1(\mathbf{x}, w_1) = \sin(w_1 x_1)\cos(w_1 x_2) - \frac{1}{1000}(w_1 - 50)^2 - \\ - \sin(w_1 x_3) - x_3 - 1 \le 0,$$

$$K_2(\mathbf{x}, w_2) = \sin(w_2 x_2)\cos(w_2 x_1) - \frac{1}{1000}(w_2 - 50)^2 - \\ - \sin(w_2 x_3) - x_3 - 1 \le 0,$$

где $w_1$ и $w_2$ принадлежат отрезку [1, 100].

Для решения задачи создадим два m-файла, первый (с именем myfun) — для вычислений целевой функции, второй (с именем mycon) — для вычислений левых частей ограничений:

```
function f = myfun(x,s)
% Целевая функция
f = sum((x-0.2).^2);
function [c,ceq,K1,K2,s] = mycon(x,s)
% Начальный интервал дискретизации
if isnan(s(1,1)),
s = [0.2 0; 0.2 0];
end
% Дискретные значения
w1 = 1:s(1,1):100;
w2 = 1:s(2,1):100;
% Полубесконечные ограничения
K1 = sin(w1*X(1)).*cos(w1*X(2)) - 1/1000*(w1-50).^2 -...
sin(w1*X(3))-X(3)-1;
K2 = sin(w2*X(2)).*cos(w2*X(1)) - 1/1000*(w2-50).^2 -...
sin(w2*X(3))-X(3)-1;
% Другие ограничения отсутствуют
c = []; ceq=[];
plot(w1,K1,'-',w2,K2,':'),title('Полубесконечные ограничения')
drawnow
```

Решение задачи иллюстрируется следующим образом:

```
» % Стартовые значения
» x0 = [0.5; 0.2; 0.3];
» % Нахождение решения
» [x,fval] = fseminf('myfun',x0,2,'mycon')
x =
    0.6535
    0.2821
    0.4013
```

На рис. 7.2 приведен графический вид функций в правых частях полубесконечных ограничений, возвращаемый m-файлом mycon для точки минимума. Как видно из рисунка, ограничения удовлетворяются.

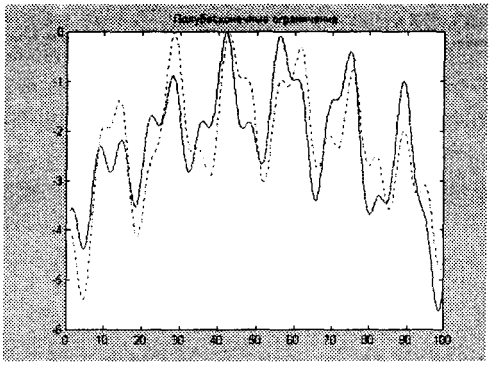

**Рис. 7.2.** *Графический вид функций в правых частях полубесконечных ограничений*

### Функция linprog

Функция `linprog` обеспечивает решение задачи линейного программирования (см. табл. 7.1).

Запись:

```
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
[x,fval] = linprog(...)
[x,fval,exitflag] = linprog(...)
[x,fval,exitflag,output] = linprog(...)
[x,fval,exitflag,output,lambda] = linprog(...)
```

Аргументы и возвращаемые величины здесь аналогичны рассмотренным ранее для других функций за одним исключением: здесь введен дополнительный аргумент `f` — вектор коэффициентов линейной целевой функции. Функция может использовать алгоритм большой размерности lipsol или алгоритм средней размерности (метод проекций).

**Пример.** Требуется найти решение задачи линейного программирования, описывающейся соотношениями

$$f(\mathbf{x}) = -5x_1 - 4x_2 - 6x_3,$$

$$x_1 - x_2 + x_3 \le 20,$$

$$3x_1 + 2x_2 + 4x_3 \le 42,$$

$$3x_1 + 2x_2 \le 42,$$

$$0 \le x_1,\ 0 \le x_2,\ 0 \le x_3.$$

Решение задачи приведено ниже.

```
» f=[-5; -4; -6]; % Вектор коэффициентов линейной целевой функции
» % Матрица коэффициентов ограничений-неравенств
» A=[1 -1 1;3 2 4;3 2 0];
» b=[20; 42; 30];  % Вектор ограничений-неравенств
» % Задание нижних границ переменных (нулей)
» lb = zeros(3,1);
» % Поиск решения
» [x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb)
Optimization terminated successfully.
x =
    0.0000
   15.0000
    3.0000
fval =
  -78.0000
exitflag =
     1
output =
      iterations: 6
      cgiterations: 0
      algorithm: 'lipsol'
lambda =
      ineqlin: [3x1 double]
      eqlin: [0x1 double]
      upper: [3x1 double]
      lower: [3x1 double]
```

Из возвращенной информации, в частности, следует:

- что оптимальное решение `x = [0.0000 15.0000 3.0000]`;
- минимальное значение целевой функции равно `-78.0000`;
- вычисления завершились нахождением решения (`exitflag>0`);
- всего было выполнено 6 итераций;
- был использован алгоритм lipsol

и т. п.

### Функция quadprog

Функция `quadprog` возвращает решение задачи квадратичного программирования. *Может использовать алгоритм как средней, так и* большой размерности.

Запись:

```
x = quadprog(H,f,A,b)
x = quadprog(H,f,A,b,Aeq,beq)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
[x,fval] = quadprog(…)
[x,fval,exitflag] = quadprog(…)
[x,fval,exitflag,output] = quadprog(…)
[x,fval,exitflag,output,lambda] = quadprog(…)
```

Аргументы (за исключением матрицы H) и возвращаемые величины аналогичны рассмотренным для функции fmincon.

**Пример.** Найдем решение задачи квадратичного программирования, имеющей описание

$$f(\mathbf{x}) = \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2,$$

$$x_1 + x_2 \le 2,$$
$$-x_1 + 2x_2 \le 2,$$
$$2x_1 + x_2 \le 3,$$
$$0 \le x_1,\ 0 \le x_2.$$

В данном случае

$$\mathbf{H} = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}, \qquad \mathbf{f} = \begin{bmatrix} -2 \\ -6 \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

и решение задачи представляется в следующем виде:

```
» H=[1 -1; -1 2];
» f=[-2; -6];
» A=[1 1; -1 2; 2 1];
» b=[2; 2; 3];
» lb = zeros(2,1);
» [x,fval,exitflag,output,lambda] = quadprog(H,f,A,b,[],[],lb)
Optimization terminated successfully.
x =
    0.6667
    1.3333
fval =
   -8.2222
exitflag =
     1
```

```
output =
      iterations: 3
      algorithm: 'medium-scale: active-set'
      firstorderopt: []
      cgiterations: []
lambda =
      lower: [2x1 double]
      upper: [2x1 double]
      eqlin: [0x1 double]
      ineqlin: [3x1 double]
```

Из возвращаемой информации, в частности, следует, что оптимальное значение `x = [0.6667 1.3333]`, минимальное значение целевой функции `-8.2222`, количество выполненных итераций — 3, использован алгоритм средней размерности.

## Функции решения уравнений

Данная группа состоит из трех функций.

### Функция mldivide

Функция обратного деления матриц (справа налево) `mldivide` (или \), строго говоря, относится к основным функциям системы MATLAB и применяется при решении систем линейных уравнений вида **Cx = d** при числе уравнений, равном числу неизвестных. Более подробная информация о функции может быть получена при запросе справочной информации путем набора в командной строке `help mldivide`.



**Пример:**

```
» C=[1 1; -1 2; 2 1];
» d=[2; 2; 3];
» x=C\d
x =
    0.7714
    1.3714
```

### Функция fsolve

Функция `fsolve` возвращает решение системы нелинейных уравнений **F(x)** = 0. Использует алгоритм средней или большой размерности.

Запись:

```
x = fsolve(fun,x0)
x = fsolve(fun,x0,options)
x = fsolve(fun,x0,options,P1,P2, … )
```

```
[x,fval] = fsolve(…)
[x,fval,exitflag] = fsolve(…)
[x,fval,exitflag,output] = fsolve(…)
[x,fval,exitflag,output,jacobian] = fsolve(…)
```

*Описание.* Возвращаемые величины, аргументы, опции соответствуют рассмотренным ранее, добавлена лишь возвращаемая величина `jacobian` — якобиан вектора целевых функций в точке найденного решения (напомним, что в терминах рассматриваемого пакета якобиан — это транспонированная матрица, составленная из столбцов-градиентов частных целевых функций).

**Пример 1.** Найдем решение системы нелинейных уравнений

$$2x_1 - x_2 - e^{-x_1} = 0,$$
$$-x_1 + 2x_2 - e^{-x_2} = 0$$

при начальном приближении (стартовом значении) `x0 = [-5 -5]`.

Составим вначале соответствующий m-файл с именем `myfun` для вычисления значений функций в левых частях уравнений:

```
function F = myfun(x)
F = [2*x(1) - x(2)- exp(-x(1));-x(1) + 2*x(2) - exp(-x(2))];
```

Затем организуем процесс вычислений:

```
» x0=[-5; -5]; % Стартовое значение
» % Задание вывода информации на каждой итерации
» options=optimset('Display','iter');
» [x,fval] = fsolve('myfun',x0,options) % Поиск решения
                                                  Norm of    First-order
Iteration  Func-count     f(x)         step      optimality CG-iterations
    1          4        47071.2            1       2.29e+004      0
    2          7        6527.47      1.45207       3.09e+003      1
    3         10        918.372      1.49186             418      1
    4         13         127.74      1.55326            57.3      1
    5         16        14.9153      1.57591            8.26      1
    6         19       0.779051      1.27662            1.14      1
    7         22     0.00372453     0.484658          0.0683      1
    8         25   9.21617e-008    0.0385552        0.000336      1
    9         28   5.66133e-017   0.000193707        8.34e-009     1
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
x =
    0.5671
    0.5671
```

```
fval =
  1.0e-008 *
   -0.5320
   -0.5320
```

Выводимая информация сообщает, что процесс поиска решения прошел нормально. Само решение приводится в последних строках листинга.

**Пример 2.** Найдем матрицу **x**, удовлетворяющую уравнению

$$\mathbf{x}\,\mathbf{x}\,\mathbf{x} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

со стартовым значением `x0 = [1, 1; 1, 1]`.

Для нахождения решения, как и раньше, составим m-файл (с именем `myfun`) для вычисления значений функции:

```
function F = myfun(x)
F = x*x*x-[1,2;3,4];
```

Решение затем определяется с использованием следующей программы:

```
» x0 = ones(2,2);% Задание стартового значения
» % Отключение вывода текущей информации
» options = optimset('Display','off');
» [x,Fval,exitflag] = fsolve('myfun',x0,options) % Нахождение решения
```

Результаты вычислений:

```
x =
   -0.1291    0.8602
    1.2903    1.1612
Fval =
  1.0e-003 *
    0.1541   -0.1163
    0.0109   -0.0243
exitflag =
     1
```

Судя по итоговому сообщению (`exitflag=1`), решение достигнуто и определяется выведенными матрицами `x` и `Fval` (эта матрица в данном случае характеризует ошибку решения).

### Функция fzero

Функция `fzero` предназначена для поиска корней (нулей) функции одной переменной, то есть для решения задачи

$a: f(a) = 0.$

Функция реализует одноименный метод, объединяющий методы бисекции и секущих, предложенный ван Вайнгаарденом, Деккером и другими сотрудниками Математического центра в Амстердаме.

Запись:

```
x = fzero(fun,x0)
x = fzero(fun,x0,options)
x = fzero(fun,x0,options,P1,P2,…)
[x,fval] = fzero(…)
[x,fval,exitflag] = fzero(…)
[x,fval,exitflag,output] = fzero(…)
```

**Примеры.**

1. Найдем оценку значения числа $\pi$ как нуля функции $\sin(x)$, расположенного вблизи точки x0 = 3:

```
» x = fzero('sin',3)
Zero found in the interval: [2.8303, 3.1697].
x =
    3.1416
```

2. Найдем корень функции $\cos(x)$ в интервале [1, 2]:

```
» x = fzero('cos', [1 2])
Zero found in the interval: [1, 2].
x =
    1.5708
```

3. Найдем ноль функции $f(x) = x^3 - 2x - 5$ вблизи точки x0 = 2. Для этого составим m-файл с именем f:

```
function y = f(x)
y = x.^3 - 2*x - 5;
```

Нахождение решения записывается следующим образом:

```
» z = fzero('f',2)
Zero found in the interval: [1.8869, 2.1131].
z =
    2.0946
```

Сохраняя полное уважение к рассматриваемой функции, заметим, что значительно более полный результат (нахождение всех корней полинома) можно получить, применяя функцию MATLAB roots. В условиях примера при этом будем иметь

```
» roots([1 0 -2 -5])
ans =
   2.0946
  -1.0473 + 1.1359i
  -1.0473 - 1.1359i
```

## Функции наименьших квадратов (подбора кривых)

Для решения задач аппроксимации, связанных с применением метода наименьших квадратов (МНК), применяются следующие пять функций.

### Функция mldivide

Функция обратного деления матриц (справа налево) `mldivide` (или `\`) используется для реализации линейного МНК в следующей постановке. Имеется переопределенная линейная система из $m$ уравнений с $n$ переменными ($m > n$)

$$\mathbf{C}\,\mathbf{x} = \mathbf{d}.$$

Требуется найти вектор решения $\mathbf{x}$.

Такое решение, получаемое с использованием линейного МНК, определяется формулой

$$\mathbf{x} = (\mathbf{C}^{\mathrm{T}}\mathbf{C})^{-1}\mathbf{C}^{\mathrm{T}}\mathbf{d},$$

которая в системе MATLAB реализуется операцией `x=C\d`. Пример:

```
» C=[1 2;2 3;3 4];
» d=[1; 2; 3];
» x=C\d
x =
    1.0000
    0.0000
```

### Функция lsqlin

Функция `lsqlin` возвращает решение задачи линейного МНК при наличии ограничений (см. табл. 7.1). Использует как алгоритм средней, так и алгоритм большой размерности.

Запись:

```
x = lsqlin(C,d,A,b)
x = lsqlin(C,d,A,b,Aeq,beq)
x = lsqlin(C,d,A,b,Aeq,beq,lb,ub)
x = lsqlin(C,d,A,b,Aeq,beq,lb,ub,x0)
x = lsqlin(C,d,A,b,Aeq,beq,lb,ub,x0,options)
[x,resnorm] = lsqlin(…)
[x,resnorm,residual] = lsqlin(…)
[x,resnorm,residual,exitflag] = lsqlin(…)
[x,resnorm,residual,exitflag,output] = lsqlin(…)
[x,resnorm,residual,exitflag,output,lambda] = lsqlin(…)
```

*Описание.* Аргументы и опции соответствуют ранее рассмотренным. Возвращаемые величины:

- x — найденное решение;
- resnorm — сумма квадратов невязок $\|\mathbf{Cx} - \mathbf{d}\|_2^2$ (для найденного значения $x$);
- residual — вектор невязок $\mathbf{Cx} - \mathbf{d}$ (также для конечного значения $x$);
- exitflag, output, lambda — см. выше.

**Пример.** Найдем с использованием МНК решение переопределенной системы линейных уравнений

$$\mathbf{Cx} = \mathbf{d}$$

при наличии ограничений

$$\mathbf{Ax} \le \mathbf{b}, \qquad \mathbf{lb} \le \mathbf{x} \le \mathbf{ub}.$$

Решение приведено ниже.

```
» C = [
        0.9501   0.7620   0.6153   0.4057
        0.2311   0.4564   0.7919   0.9354
        0.6068   0.0185   0.9218   0.9169
        0.4859   0.8214   0.7382   0.4102
        0.8912   0.4447   0.1762   0.8936];
» d = [
       0.0578
       0.3528
       0.8131
       0.0098
       0.1388];
» A =[
       0.2027   0.2721   0.7467   0.4659
       0.1987   0.1988   0.4450   0.4186
       0.6037   0.0152   0.9318   0.8462];
» b =[
      0.5251
      0.2026
      0.6721];
» lb = -0.1*ones(4,1);
» ub = 2*ones(4,1);
» [x,resnorm,residual,exitflag,output,lambda] = ...
   lsqlin(C,d,A,b,[ ],[ ],lb,ub);
Optimization terminated successfully.
```

```
» x
x =
   -0.1000
   -0.1000
    0.2152
    0.3502
```

Судя по сообщению `Optimization terminated successfully` (Оптимизация успешно закончена), возвращенное решение является «правильным».

### Функция lsqcurvefit

Функция `lsqcurvefit` возвращает решение задачи аппроксимации (нелинейной «подгонки» кривой, см. табл. 7.1) под имеющиеся экспериментальные данные `xdata`, `ydata`, иначе говоря, возвращает параметры функции заданного вида, обеспечивающие ее наименьшее среднеквадратичное отклонение от указанных данных.

Запись:

```
x = lsqcurvefit(fun,x0,xdata,ydata)
x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub)
x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,options)
x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,options,P1,P2,…)
[x,resnorm] = lsqcurvefit(…)
[x,resnorm,residual] = lsqcurvefit(…)
[x,resnorm,residual,exitflag] = lsqcurvefit(…)
[x,resnorm,residual,exitflag,output] = lsqcurvefit(…)
[x,resnorm,residual,exitflag,output,lambda] = lsqcurvefit(…)
[x,resnorm,residual,exitflag,output,lambda,jacobian] =
lsqcurvefit(…)
```

Аргументы, опции, возвращаемые величины — такие же, как у ранее рассмотренных функций.

Пример использования функции `lsqcurvefit` приведен ниже.

М-файл:

```
function F = myfun(x,xdata)
F = x(1)*xdata.^2 + x(2)*sin(xdata) + x(3)*xdata.^3;
```

Основная программа:

```
» xdata = [3.6 7.7 9.3 4.1 8.6 2.8 1.3 7.9 10.0 5.4];
» ydata = [16.5 150.6 263.1 24.7 208.5 9.9 2.7 163.9 325.0 54.3];
» x0 = [10, 10, 10];  % стартовое значение
» % Нахождение решения
» [x,resnorm] = lsqcurvefit('myfun',x0,xdata,ydata)
```

```
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
x =
    0.2269    0.3385    0.3021
resnorm =
    6.2950
```

Судя по итоговому сообщению, оптимизация здесь проведена успешно несмотря на то, что итоговая сумма квадратов невязок оказалась *не слишком близкой к нулю — в условиях примера она отражает* действие чисто случайной (шумовой) составляющей, содержащейся в экспериментальных данных.

### Функция lsqnonlin

Функция `lsqnonlin` возвращает решение задачи нелинейного МНК:

$$\min_{\mathbf{x}} f(\mathbf{x}) = f_1^2(\mathbf{x}) + f_2^2(\mathbf{x}) + \ldots + f_m^2(\mathbf{x}) + \text{const} =$$

$$= \min_{\mathbf{x}} \frac{1}{2}\|F(\mathbf{x})\|_2^2 = \frac{1}{2}\sum_i f_i^2(\mathbf{x}).$$

Использует алгоритмы средней или большой размерности.

Запись:

```
x = lsqnonlin(fun,x0)
x = lsqnonlin(fun,x0,lb,ub)
x = lsqnonlin(fun,x0,lb,ub,options)
x = lsqnonlin(fun,x0,options,P1,P2, … )
[x,resnorm] = lsqnonlin(…)
[x,resnorm,residual] = lsqnonlin(…)
[x,resnorm,residual,exitflag] = lsqnonlin(…)
[x,resnorm,residual,exitflag,output] = lsqnonlin(…)
[x,resnorm,residual,exitflag,output,lambda] = lsqnonlin(…)
[x,resnorm,residual,exitflag,output,lambda,jacobian] = lsqnonlin(…)
```

**Пример.** Найдем значение вектора $x$, минимизирующее функцию

$$\sum_{k=1}^{10}\left(2 + 2k - e^{kx_1} - e^{kx_2}\right)^2$$

при начальном значении `x0 = [0.3, 0.4]`.

Для решения задачи вначале составим соответствующий m-файл:

```
function F = myfun(x)
k = 1:10;
F = 2 + 2*k - exp(k*x(1)) - exp(k*x(2));
```

а затем — собственно программу минимизации:

```
» x0 = [0.3 0.4]; % Стартовое значение
» [x,resnorm] = lsqnonlin('myfun',x0)  % Поиск решения
```

Результаты расчета:

```
Optimization terminated successfully:
 Norm of the current step is less than OPTIONS.TolX
x =
    0.2578    0.2578
resnorm =
  124.3622
```

В данном примере итерационный процесс успешно закончился нахождением решения `x = [0.2578 0.2578]` при сумме квадратов невязок, равной 124,3622.

### Функция lsqnonneg

Функция `lsqnonneg` возвращает решение так называемой задачи неотрицательного линейного МНК (см. табл. 7.1).

Запись:

```
x = lsqnonneg(C,d)
x = lsqnonneg(C,d,x0)
x = lsqnonneg(C,d,x0,options)
[x,resnorm] = lsqnonneg(…)
[x,resnorm,residual] = lsqnonneg(…)
[x,resnorm,residual,exitflag] = lsqnonneg(…)
[x,resnorm,residual,exitflag,output] = lsqnonneg(…)
[x,resnorm,residual,exitflag,output,lambda] = lsqnonneg(…)
```

**Пример.** Ниже приведены программа и результаты ее вычислений, позволяющие сравнить решения задачи минимизации, возвращаемые с использованием линейного МНК без ограничений и с использованием рассматриваемой функции:

```
» C=[
        0.0372    0.2869
        0.6861    0.7071
        0.6233    0.6245
        0.6344    0.6170];
» d=[
        0.8587
        0.1781
        0.0747
        0.8405];
```

```
» [C\d, lsqnonneg(C,d)]
ans =
   -2.5627         0
    3.1108    0.6929
» [norm(C*(C\d)-d), norm(C*lsqnonneg(C,d)-d)]
ans =
    0.6674    0.9118
```

Видно, что *линейный* МНК обеспечил результат с меньшими невязками, зато применение функции `lsqnonneg` позволило получить (наилучшее) неотрицательное решение `x = [3.1108 0.6929]`.

### Функции-утилиты

К утилитам относятся две следующие функции пакета.

- **optimget** — функция, возвращающая значение установленных опций. Запись:

```
val = optimget(options,'param')
val = optimget(options,'param',default)
```

- **optimset** — функция для создания или редактирования опций решения задачи минимизации. Запись:

```
options = optimset('param1',value1,'param2',value2,...)
optimset
options = optimset
options = optimset(optimfun)
options = optimset(oldopts,'param1',value1,...)
options = optimset(oldopts,newopts)
```

*Описание.* Аргументы утилиты:

  - **'param1', value1, 'param2', value2,...** — имена и задаваемые значения опций;
  - **optimfun** — имя одной из функций оптимизации;
  - **oldopts, newopts** — имена структур опций («старое» и «новое»).

Имена и значения опций представлены в табл. 7.2 (значения по умолчанию приведены в фигурных скобках).

**Таблица 7.2.** *Имена и значения опций функций пакета Optimization Toolbox*

| Имя | Значения |
|---|---|
| Опции, используемые для алгоритмов как средней, так и большой *размерности* | |
| **Diagnostics** | **[ on \| {off} ]** |

| Имя | Значения |
|---|---|
| Display | [ off \| iter \| {final} ] |
| GradObj | [ on \| {off} ] |
| Jacobian | [ on \| {off} ] |
| LargeScale | [ {on} \| off ] |
| MaxFunEvals | [целое положительное] |
| MaxIter | [целое положительное] |
| TolCon | [положительная константа] |
| TolFun | [положительная константа] |
| TolX | [положительная константа] |
| Опции, используемые только для алгоритмов большой размерности | |
| Hessian | [ on \| {off} ] |
| HessPattern | [разреженная матрица] |
| JacobPattern | [разреженная матрица] |
| MaxPCGIter | [целое положительное] |
| PrecondBandWidth | [целое положительное\| Inf ] |
| TolPCG | [положительная константа\| {0.1}] |
| TypicalX | [вектор] |
| Опции, используемые только для алгоритмов средней размерности | |
| DerivativeCheck | [ on \| {off} ] |
| DiffMaxChange | [положительная константа \| {1e-1}] |
| DiffMinChange | [положительная константа \| {1e-8}] |
| GoalsExactAchieve | [ целое положительное \| {0} ] |
| GradConstr | [ on \| {off} ] |
| HessUpdate | [ {bfgs} \| dfp \| gillmurray \| steepdesc ] |
| LevenbergMarquardt | [ on \| off ] |
| LineSearchType | [ cubicpoly \| {quadcubic} ] |
| MeritFunction | [ singleobj \| {multiobj} ] |
| MinAbsMax | [ целое положительное \| {0} ] |

## Демонстрационные функции

Данную группу составляют следующие функции:

- демонстрации алгоритмов большой размерности:
  - circustent — решение задачи квадратичного программирования для нахождения оптимальной формы тента купола цирка;

- `molecule` — задача моделирования (определения пространственной структуры) молекулы с использованием нелинейной оптимизации без ограничений;
- `optdeblur` — решение задачи повышения качества изображения (фотографии) с применением линейного МНК с ограничениями.

- демонстрации алгоритмов средней размерности и другие функции:
  - `bandemo` — минимизация «банановой» функции;
  - `dfildemo` — проектирование фильтра;
  - `goaldemo` — задача достижения цели;
  - `optdemo` — меню демонстрационных программ;
  - `tutdemo` — электронный курс обучения работе с функциями пакета.

В качестве примеров на рис. 7.3 и рис. 7.4 приведены демонстрации, соответственно отражающие задачи нахождения оптимальной формы тента купола цирка и минимизации (различными методами) «банановой» функции.

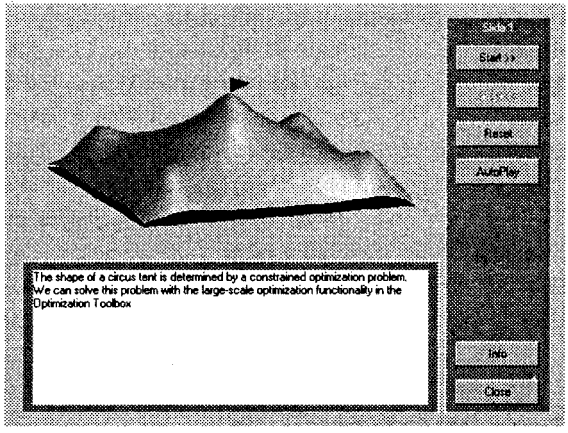


**Рис. 7.3.** Иллюстрация задачи нахождения оптимальной формы тента купола цирка

Более подробное знакомство с приведенными функциями легко осуществить через главное меню MATLAB (команда **Help ▸ Examples and Demos**, раздел **Toolboxes/Optimization**).

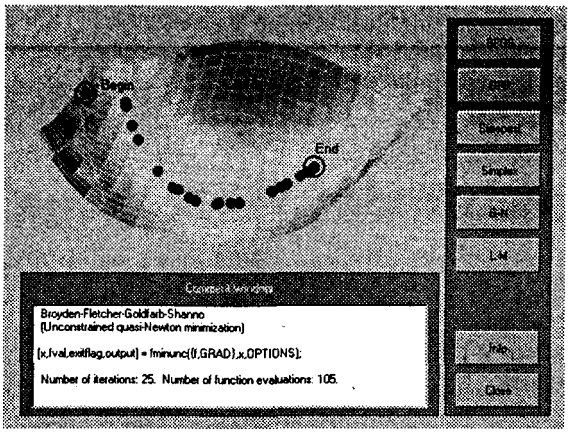


**Рис. 7.4.** Минимизация «банановой» функции

# Примеры решения оптимизационных задач

## Минимизация без ограничений

Рассмотрим задачу нахождения значений переменных $x_1$ и $x_2$, обеспечивающих решение задачи минимизации

$$\min_{\mathbf{x}} f(\mathbf{x}) = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 1).$$

Нахождение решения производится в соответствии со следующими этапами.

**Этап 1.** Составление m-файла (с именем objfun), реализующего вычисление значения целевой функции:

```
function  f = objfun(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
```

**Этап 2.** Составление программы с использованием подходящей функции пакета (в данном случае — функции fminunc):

```
» % Стартовое значение
» x0=[-1,1];
» % Задание опции использования алгоритма средней размерности
» options = optimset('LargeScale','off');
» % Поиск решения
» [x,fval,exitflag,output] = fminunc('objfun',x0,options)
```

Выполнение программы приведет к следующему результату:

```
Optimization terminated successfully:
 Current search direction is a descent direction, and magnitude of
 directional derivative in search direction less than 2*options.TolFun
x =
    0.5000   -1.0000
fval =
  1.3031e-010
exitflag =
     1
output =
        iterations: 7
        funcCount: 40
        stepsize: 1
        firstorderopt: 8.1995e-004
        algorithm: 'medium-scale: Quasi-Newton line search'
```

Значения `x = [0.5000 -1.0000]` и `fval = 1.3031e-010` — искомое решение задачи. Значение `exitflag=1` дает информацию, что найдена точка минимума (возможно, локального).

*Выходная структура (информация о результатах оптимизации)* определяется идентификатором `output`:

- число выполненных итераций (`iterations`): 7;
- число вычислений функции (`funcCount`): 40;
- шаг поиска (`stepsize`): 1;
- степень оптимальности найденного решения (`firstorderopt`) — норма вектора-градиента в точке найденного решения: 8.1995e-004;
- использованный алгоритм (`algorithm`): квазиньютоновский с одномерной оптимизацией (`'medium-scale: Quasi-Newton line search'`), относящийся к числу алгоритмов средней размерности.

## Минимизация с ограничениями в форме нелинейных неравенств

Теперь рассмотрим задачу минимизации той же целевой функции, но при наличии ограничений в форме нелинейных неравенств:

$$x_1x_2 - x_1 - x_2 \le -1{,}5,$$

$$x_1x_2 \ge -10.$$

Данная задача (см. табл. 7.1) может быть решена с применением функции `fmincon` в соответствии с теми же этапами, что и предыдущая. Поскольку при использовании данной функции нелинейные

ограничения должны быть представлены в виде $\mathbf{c}(\mathbf{x}) \le 0$, перепишем их соответствующим образом:

$$x_1x_2 - x_1 - x_2 + 1{,}5 \le 0,$$
$$-x_1x_2 - 10 \le 0.$$

**Этап 1.** Составление m-файла (с именем confun), возвращающего значения левых частей ограничивающих неравенств:

```
function [c, ceq] = confun(x)
% Нелинейные ограничения в форме неравенств
c = [1.5 + x(1)*x(2) - x(1) - x(2); -x(1)*x(2) - 10];
% Нелинейные ограничения в форме равенств
ceq = [];
```

**Этап 2.** Составление программы минимизации:

```
» x0=[-1,1]; % Задание начальных значений
» options = optimset('LargeScale','off');  % Задание опций
» % Поиск решения
» [x, fval]=fmincon('objfun',x0,[],[],[],[],[],[],'confun',options)
```

Результаты вычислений сообщают о найденном решении, количестве ограничений и т. п.:

```
Optimization terminated successfully:
 Search direction less than 2*options.TolX and
  maximum constraint violation is less than options.TolCon
Active Constraints:
     1
     2
x =
   -9.5474    1.0474
fval =
    0.0236
```

## Минимизация с дополнительными ограничениями на диапазоны изменения переменных

Функция fmincon может быть применена и для поиска решения в задаче минимизации, в которой допустимые значения переменных ограничены некоторыми диапазонами: $\mathbf{lb} \le \mathbf{x} \le \mathbf{ub}$.

Продолжим рассмотрение примера, введя дополнительные ограничения $x_1 \ge 0$, $x_2 \ge 0$, что эквивалентно заданию lb = [0 0], ub = [ ].

Программа оптимизации и результаты вычислений для данного случая приведены ниже:

```
» x0 = [-1,1]; % Задание начальных значений
» lb = [0, 0]; % Задание нижних границ переменных
» ub = [ ]; % Сверху переменные не ограничены
» options = optimset('LargeScale','off');  % Задание опций
» % Поиск решения
» [x,fval] = fmincon('objfun',x0,[],[],[],[],lb,ub,'confun',options)
Optimization terminated successfully:
 Search direction less than 2*options.TolX and
  maximum constraint violation is less than options.TolCon
Active Constraints:
     1
     3
x =
         0    1.5000
fval =
    8.5000
```

Найденное решение удовлетворяет ограничениям, наложенным на диапазоны изменения переменных. Проверим теперь выполнение ограничений в форме неравенств:

```
» % Проверка выполнения ограничений в форме неравенств
» [c, ceq] = confun(x)
c =
     0
   -10
ceq =
     []
```

Как следует из приведенного результата, ограничения в форме неравенств выполнены (ограничения в форме равенств отсутствуют).

## Использование вектора-градиента, аналитически задаваемого пользователем

По умолчанию функции пакета заменяют точные значения производных целевой функции и ограничений (в требуемых случаях) их оценками в виде первых и/или вторых разностей. Иногда целесообразно задать аналитическое вычисление данных производных — это может привести к ускорению процесса поиска решения. Рассмотрим такой подход при задании векторов первых производных — векторов-градиентов.

Продолжая решение в условиях приведенного примера, получим следующие аналитические выражения:

- градиент целевой функции:

$$\frac{df(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1) + e^{x_1}(8x_1 + 4x_2) \\ e^{x_1}(4x_1 + 4x_2 + 2) \end{bmatrix},$$

- матрица, столбцы которой являются градиентами функций в левых частях ограничений-неравенств:

$$\begin{bmatrix} \frac{dc_1}{dx_1} & \frac{dc_2}{dx_1} \\ \frac{dc_1}{dx_2} & \frac{dc_2}{dx_2} \end{bmatrix} = \begin{bmatrix} x_2 - 1 & -x_2 \\ x_1 - 1 & -x_1 \end{bmatrix}.$$

Используя данные выражения, создадим требуемые m-файлы.

**Этап 1.** Создание m-файла (с именем objfungrad) для расчета значений целевой функции и ее градиента:

```
function [f,G] = objfungrad(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
% Градиент целевой функции
t = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
G = [ t + exp(x(1)) * (8*x(1) + 4*x(2)), exp(x(1))*(4*x(1)+4*x(2)+2)];
```

**Этап 2.** Создание m-файла для нелинейных ограничений и их градиента:

```
function [c,ceq,DC,DCeq] = confungrad(x)
% Нелинейные ограничения-неравенства
c(1) = 1.5 + x(1) * x(2) - x(1) - x(2);
c(2) = -x(1) * x(2) - 10;
% Градиент ограничений-неравенств:
DC= [x(2) - 1, -x(2); x(1) - 1, -x(1)];
% Нелинейные ограничения-равенства отсутствуют
ceq=[];
DCeq = [ ];
```

**Этап 3.** Создание программы оптимизации.

```
» x0 = [-1,1];
» options = optimset('LargeScale','off');
» % Разрешение использования градиентов
» options = optimset(options,'GradObj','on','GradConstr','on');
» lb = [ ]; ub = [ ]; % Границы диапазонов не заданы
» % Поиск решения
» [x,fval] = fmincon('objfungrad',x0,[],[],[],[],lb,ub,...
              'confungrad',options)
```

Результаты вычислений:

```
Optimization terminated successfully:
 Search direction less than 2*options.TolX and
  maximum constraint violation is less than options.TolCon
Active Constraints:
     1
     2
x =
   -9.5474    1.0474
fval =
    0.0236
```

Проверка выполнения ограничений:

```
» c
c =
     0
   -10
```

Как видно, в точке решения заданные ограничения выполняются.

## Задача достижения цели

Рассмотрим следующий пример. Пусть некоторая замкнутая линейная динамическая система управления 3-го порядка описывается уравнениями

$$\dot{x} = (\mathbf{A} + \mathbf{BKC})x + \mathbf{B}u,$$
$$y = \mathbf{C}x,$$

где матрицы

$$\mathbf{A} = \begin{bmatrix} -0{,}5 & 0 & 0 \\ 0 & -2 & 10 \\ 0 & 1 & -2 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 0 \\ -2 & 2 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

отражающие динамические свойства объекта управления, являются заданными, а матрица **K** регулятора — изменяемой.

Как известно, качество работы подобных систем (качество переходных процессов) определяется расположением на комплексной плоскости собственных чисел матрицы **A** + **BKC**. Поставим задачу оптимизации таким образом: при задании диапазона возможных изменений элементов матрицы **K** от −4 до +4 подобрать эти элементы таким образом, чтобы указанные собственные числа равнялись величинам [−5, −3, −1]. В подобной формулировке задача является многокри-

териальной (3 критерия) и соответствует задаче достижения цели, приведенной в табл. 7.1.

Решение задачи, как и раньше, проведем поэтапно.

**Этап 1.** Создадим m-файл (с именем eigfun) для вычисления и упорядочивания по величине собственных чисел матрицы **A + BKC**:

```
function F = eigfun(K,A,B,C) % Нахождение собственных чисел
F = sort(eig(A+B*K*C)); % Упорядочивание собственных чисел
```

**Этап 2.** Составление оптимизирующей программы:

```
» A = [-0.5 0 0; 0 -2 10; 0 1 -2];
» B = [1 0; -2 2; 0 1];
» C = [1 0 0; 0 0 1];
» K0 = [-1 -1; -1 -1];  % Задание начальных условий
» % Задание вектора целей
» goal = [-5,-3,-1];
» weight = abs(goal); % Задание вектора весовых коэффициентов
» lb = -4*ones(size(K0)); % Нижние границы элементов матрицы K
» ub = 4*ones(size(K0)); % Верхние границы элементов матрицы K
» % Установка опции вывода информации
» options = optimset('Display','iter');
» [K,fval,attainfactor] = fgoalattain('eigfun',K0,...
        goal,weight,[],[],[],[],lb,ub,[],options,A,B,C)
```

Результаты вычислений:

| Iter | F-count | Attainment factor | Step-size | Directional derivative | Procedure |
|---|---|---|---|---|---|
| 1 | 6 | 1.885 | 1 | 1.03 | |
| 2 | 13 | 1.061 | 1 | -0.679 | |
| 3 | 20 | 0.4211 | 1 | -0.523 | Hessian modified |
| 4 | 27 | -0.06352 | 1 | -0.053 | Hessian modified twice |
| 5 | 34 | -0.1571 | 1 | -0.133 | |
| 6 | 41 | -0.3489 | 1 | -0.00768 | Hessian modified |
| 7 | 48 | -0.3643 | 1 | -4.25e-005 | Hessian modified |
| 8 | 55 | -0.3645 | 1 | -0.00303 | Hessian modified twice |
| 9 | 62 | -0.3674 | 1 | -0.0213 | Hessian modified |
| 10 | 69 | -0.3806 | 1 | 0.00266 | |
| 11 | 76 | -0.3862 | 1 | -2.73e-005 | Hessian modified twice |
| 12 | 83 | -0.3863 | 1 | -1.2e-013 | Hessian modified twice |

```
Optimization terminated successfully:
 Search direction less than 2*options.TolX and
  maximum constraint violation is less than options.TolCon
Active Constraints:
     1
     2
     4
     9
    10
K =
   -4.0000   -0.2564
   -4.0000   -4.0000
fval =
   -6.9313
   -4.1588
   -1.4099
attainfactor =
   -0.3863
```

Выведенная информация имеет следующий характер:

- `Iter` — номер итерации;
- `F-count` — количество вычислений функции;
- `Attainment factor` — коэффициент (уровень) достижения целей;
- `Step-size` — шаг поиска;
- `Directional derivative` — норма градиента;
- `Procedure` — выполненная процедура (`Hessian modified` — гессиан модифицирован, `Hessian modified twice` — гессиан модифицирован дважды);
- `Optimization terminated successfully` — оптимизация проведена успешно;
- конечное значение матрицы

```
  K =
-4.0000   -0.2564
-4.0000   -4.0000
```

- итоговые значения собственных чисел

```
fval =
   -6.9313
   -4.1588
   -1.4099
```

- итоговое значение коэффициента достижения целей attainfactor = -0.3863, что говорит о «перевыполнении» заданных целей в среднем на 38 % (сравнение итоговых значений собственных чисел с заданными это подтверждает).

Полученные результаты не слишком близки к целевым. Поэтому для получения более приемлемого результата внесем коррекцию в программу оптимизации — зададим опцию точного достижения всех трех целей:

```
» options = optimset('GoalsExactAchieve',3);
```

после чего повторим поиск оптимального решения:

```
» [K,fval,attainfactor] = fgoalattain(...
  'eigfun',K0,goal,weight,[],[],[],[],lb,ub,[],options,A,B,C)
Optimization terminated successfully:
 Magnitude of directional derivative in search direction
  less than 2*options.TolFun and maximum constraint violation
  is less than options.TolCon
Active Constraints:
     9
    10
    12
    14
K =
   -1.5954    1.2040
   -0.4201   -2.9046
fval =
   -5.0000
   -3.0000
   -1.0000
attainfactor =
     0
```

Очевидно, теперь результат является вполне удовлетворительным. Отметим, что подробную демонстрацию приведенного примера можно получить, используя функцию goaldemo.

## Решение системы нелинейных уравнений с заданием якобиана

Перейдем к решению задач на основе алгоритмов большой размерности. Рассмотрим задачу нахождения решения системы $\mathbf{F(x)} = 0$ из $n = 1000$ уравнений вида

$$F(1) = 3x_1 - 2x_1^2 - 2x_2 + 1,$$

...

$$F(i) = 3x_i - 2x_i^2 - x_{i-1} - 2x_{i+1} + 1,$$

...

$$F(n) = 3x_n - 2x_n^2 - 2x_{n-1} + 1,$$

используя функцию fsolve с применением алгоритма большой размерности (заметим, что якобиан данной системы функций, как нетрудно показать, является разреженной матрицей).

**Этап 1.** Составим m-файл (под именем nlsf1) для вычисления целевых функций и якобиана:

```
function [F,J] = nlsf1(x); % Вычисление векторной функции
n = length(x);
F = zeros(n,1);
i = 2:(n - 1);
F(i) = (3 - 2*x(i)).*x(i) - x(i -1) - 2*x(i+1)+ 1;
F(n) = (3 - 2*x(n)).*x(n) - x(n -1) + 1;
F(1) = (3 - 2*x(1)).*x(1) - 2*x(2) + 1;
% Вычисление якобиана
d = - 4*x + 3*ones(n,1); D = sparse(1:n,1:n,d,n,n);
c = - 2*ones(n-1,1); C = sparse(1:n -1,2:n,c,n,n);
e = - ones(n-1,1); E = sparse(2:n,1:n -1,e,n,n);
J = C + D + E;
```

**Этап 2.** Составление программы поиска решения:

```
» xstart = -ones(1000,1); % Вектор начальных значений
» fun = 'nlsf1'; % Имя целевой функции (M-файла)
» options = optimset('Display','iter','Jacobian','on');
» [x,fval,exitflag,output] = fsolve(fun,xstart,options);
```

Результаты вычислений:

```
                                          Norm of    First-order
Iteration  Func-count  f(x)          step   ptimality  CG-iterations
    1          2      1011          1        19            0
    2          3      16.1942       7.91898  2.35          3
    3          4      0.0228027     1.33142  0.291         3
    4          5      0.000103359   0.0433329 0.0201       4
    5          6      7.3792e-007   0.0022606 0.000946     4
    6          7      4.02299e-010  0.000268381 4.12e-005  5
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
```

Для получения всех 1000 элементов решений в командной строке необходимо набрать x и нажать клавишу Enter. Заметим, что в данном случае алгоритм большой размерности применяется по умолчанию, в то время как использование пользовательского якобиана задано с помощью функции optimset.

## Решение системы нелинейных уравнений с представлением оценки якобиана в виде разреженной матрицы

Иногда выражения для элементов якобиана столь сложны, что лучше доверить нахождение их оценок (в виде первых разностей) самой функции минимизации. Для сокращения вычислений в алгоритмах большой размерности в этом случае задается специальная опция — опция представления разреженного образа якобиана.

Данный разреженный образ есть матрица того же размера, что и якобиан, при этом часть элементов — ненулевые, остальные — нулевые. Наличие *ij*-го ненулевого элемента означает, что при выполнении расчетов будет находиться оценка соответствующего элемента якобиана; для нулевых элементов такие расчеты производиться не будут. Указанную матрицу следует заранее подготовить. Будем полагать, продолжая рассмотрение предыдущего примера, что такая матрица под именем Jstr подготовлена и сохранена в файле nlsdat1.mat.

**Этап 1.** Создание m-файла (под именем nlsf1a)для вычисления значений векторной функции:

```
function F = nlsf1a(x);% Вычисление векторной функции
n = length(x);
F = zeros(n,1);
i = 2:(n -1);
F(i) = (3 - 2*x(i)).*x(i) - x(i -1) - 2*x(i+1) + 1;
F(n) = (3 - 2*x(n)).*x(n) - x(n -1) + 1.
F(1) = (3 - 2*x(1)).*x(1) - 2*x(2) + 1;
```

**Этап 2.** Создание оптимизирующей программы:

```
» xstart = -ones(1000,1); % Начальные значения
» fun = 'nlsf1a'; % Имя функции (M-файла)
» load nlsdat1 % Загрузка матрицы Jstr
» % Задание необходимых опций
» options = optimset('Display','iter','JacobPattern',Jstr,...
      'PrecondBandWidth',1);
» % Поиск решения
» [x,fval,exitflag,output] = fsolve(fun,xstart,options);
```

Результаты вычислений:

```
                                                Norm of      First-order
Iteration  Func-count     f(x)          step    optimality   CG-iterations
    1          6           1011             1          19          0
    2         11        16.0839       7.92496        1.92          1
    3         16      0.0458181        1.3279       0.579          1
    4         21    0.000101184     0.0631898      0.0203          2
    5         26   3.16615e-007    0.00273698     0.00079          2
    6         31   9.72482e-010    0.00018111    5.82e-005         2
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
```

Данные результаты по точности и по времени счета сопоставимы с полученными в предыдущем случае (но здесь якобиан не задавался).

## Нелинейный МНК с вычислением оценок всех элементов якобиана

Алгоритмы большой размерности, применяемые в функциях `lsqnonlin`, `lsqcurvefit` и `fsolve`, могут использоваться для решения задач малой и средней размерности вообще без задания якобиана пользователя или разреженного образа якобиана. Такую возможность рассмотрим на примере задачи минимизации суммы 10 функций, зависящих от 2 переменных,

$$\sum_{k=1}^{10}\left(2+2k-e^{kx_1}-e^{kx_2}\right)^2,$$

ранее уже приведенном для иллюстрации функции `lsqnonlin` (см. выше).

Поскольку якобиан не вычисляется в m-файле `myfun.m` и по умолчанию его использование не задано в опциях, функция `lsqnonlin` автоматически запускает алгоритм большой размерности с заданием разреженного образа якобиана в виде матрицы `Jstr = sparse(ones(10,2))`, то есть матрицы, все элементы которой — единицы. Когда подпрограмма определения оценки якобиана вызывается первый раз, она обнаруживает, что матрица `Jstr` не является разреженной, то есть нет никакой необходимости рассматривать и хранить в памяти данную матрицу как разреженную, и в дальнейшем все вычисления производятся как для обычных матриц.

## Минимизация нелинейной функции с использованием градиента и гессиана

Пусть требуется найти минимум функции вида

$$f(\mathbf{x}) = \sum_{i=1}^{n-1}\left[\left(x_i^2\right)^{x_{i+1}^2+1} + \left(x_{i+1}^2\right)^{x_i^2+1}\right]$$

при $n = 1000$.

В соответствии с табл. 7.1 данная задача является задачей безусловной оптимизации, и ее целесообразно решать с помощью функции `fminunc`.

**Этап 1.** Составление m-файла (с именем `brownfgh`) для вычислений значений целевой функции, ее градиента и разреженной трехдиагональной матрицы Гессе:

```
function [f,g,H] = brownfgh(x)
% Вычисление функции
  n=length(x); y=zeros(n,1);
  i=1:(n-1);
  y(i)=(x(i).^2).^(x(i+1).^2+1)+(x(i+1).^2).^(x(i).^2+1);
  f=sum(y);
% Вычисление градиента
i=1:(n-1); g = zeros(n,1);
g(i)= 2*(x(i+1).^2+1).*x(i).*((x(i).^2).^(x(i+1).^2))+...
            2*x(i).*((x(i+1).^2).^(x(i).^2+1)).*log(x(i+1).^2);
g(i+1)=g(i+1)+...
             2*x(i+1).*((x(i).^2).^(x(i+1).^2+1)).*log(x(i).^2)+...
             2*(x(i).^2+1).*x(i+1).*((x(i+1).^2).^(x(i).^2));
% Вычисление (разреженной, симметричной) матрицы Гессе
 v=zeros(n,1);
 i=1:(n-1);
 v(i)=2*(x(i+1).^2+1).*((x(i).^2).^(x(i+1).^2))+...
             4*(x(i+1).^2+1).*(x(i+1).^2).*(x(i).^2).*...
             ((x(i).^2).^((x(i+1).^2)-1))+...
             2*((x(i+1).^2).^(x(i).^2+1)).*(log(x(i+1).^2));
 v(i)=v(i)+4*(x(i).^2).*((x(i+1).^2).^(x(i).^2+1)).*...
             ((log(x(i+1).^2)).^2);
 v(i+1)=v(i+1)+...
             2*(x(i).^2).^(x(i+1).^2+1).*(log(x(i).^2))+...
             4*(x(i+1).^2).*((x(i).^2).^(x(i+1).^2+1)).*...
             ((log(x(i).^2)).^2)+...
             2*(x(i).^2+1).*((x(i+1).^2).^(x(i).^2));
 v(i+1)=v(i+1)+...
             4*(x(i).^2+1).*(x(i+1).^2).*(x(i).^2).*...
             ((x(i+1).^2).^(x(i).^2-1));
 v0=v;
```

```
v=zeros(n-1,1);
v(i)=4*x(i+1).*x(i).*((x(i).^2).^(x(i+1).^2))+...
            4*x(i+1).*(x(i+1).^2+1).*x(i).*((x(i).^2).^...
            (x(i+1).^2)).*log(x(i).^2);
v(i)=v(i)+4*x(i+1).*x(i).*((x(i+1).^2).^(x(i).^2)).*log(x(i+1).^2);
v(i)=v(i)+4*x(i).*((x(i+1).^2).^(x(i).^2)).*x(i+1);
v1=v;
i=[(1:n)';(1:(n-1))'];
j=[(1:n)';(2:n)'];
s=[v0;2*v1];
H=sparse(i,j,s,n,n);
H=(H+H')/2;
```

**Этап 2.** Составление оптимизирующей программы:

```
» xstart = -ones(n,1);
» xstart(2:2:n,1) = 1;
» % Задание использования градиента и гессиана пользователя
» options = optimset('GradObj','on','Hessian','on');
» % Поиск решения
» [x,fval,exitflag,output] = fminunc('brownfgh',xstart,options);
```

Результаты вычислений свидетельствуют о получении корректного решения:

```
Optimization terminated successfully:
 First-order optimality less than OPTIONS.TolFun, and no negative/zero
 curvature
» exitflag
exitflag =
     1
» fval
fval =
  2.8709e-017
» output.iterations
ans =
     8
» output.cgiterations
ans =
     7
» output.firstorderopt
ans =
  4.7948e-010
```

## Нелинейная оптимизация с использованием разреженных образов градиента и гессиана

Следующий пример иллюстрирует решение задачи минимизации большой размерности с заданием пользовательского градиента, но с приближенным вычислением гессиана, при котором для ускорения расчетов использован его разреженный образ. Данный образ должен быть заранее подготовлен в виде некоторой разреженной матрицы (в рассматриваемом примере это матрица Hstr, сохраненная в файле brownhstr.mat).

Необходимо также (с помощью функции optimset) установить значение 'on' для опции GradObj, поскольку градиент задается пользователем (вычисляется в m-файле).

**Этап 1.** Создание m-файла (с именем brownfg) для расчета значений минимизируемой функции и ее градиента.

```
function [f,g] = brownfg(x,dummy)
% Вычисление функции
n=length(x); y=zeros(n,1);
i=1:(n-1);
y(i)=(x(i).^2).^(x(i+1).^2+1) + (x(i+1).^2).^(x(i).^2+1);
f=sum(y);
% Вычисление градиента
i=1:(n-1); g = zeros(n,1);
g(i) = 2*(x(i+1).^2+1).*x(i).* ...
((x(i).^2).^(x(i+1).^2))+ ...
2*x(i).*((x(i+1).^2).^(x(i).^2+1)).* ...
log(x(i+1).^2);
g(i+1) = g(i+1) + ...
2*x(i+1).*((x(i).^2).^(x(i+1).^2+1)).* ...
log(x(i).^2) + ...
2*(x(i).^2+1).*x(i+1).* ...
((x(i+1).^2).^(x(i).^2));
```

**Этап 2.** Создание программы поиска решения:

```
» fun = 'brownfg'; % Задание имени функции (M-файла)
» load brownhstr % Загрузка разреженного образа гессиана
» % Графическое представление разреженной матрицы
» spy(Hstr)
» n = 1000;
» xstart = -ones(n,1);
» xstart(2:2:n,1) = 1;
```

```
» % Задание опций
» options = optimset('GradObj','on','HessPattern',Hstr);
» [x,fval,exitflag,output] = fminunc(fun,xstart,options);
```

Результаты расчетов:

```
Optimization terminated successfully:
 First-order optimality less than OPTIONS.TolFun,
 and no negative/zero curvature detected
» exitflag
exitflag =
     1
» fval
fval =
  7.4739e-017
» output.iterations
ans =
     8
» output.firstorderopt
ans =
  7.9822e-010
```

Данные результаты говорят об успешном решении задачи. Команда `spy(Hstr)` позволяет графически представить структуру разреженной матрицы (рис. 7.5).

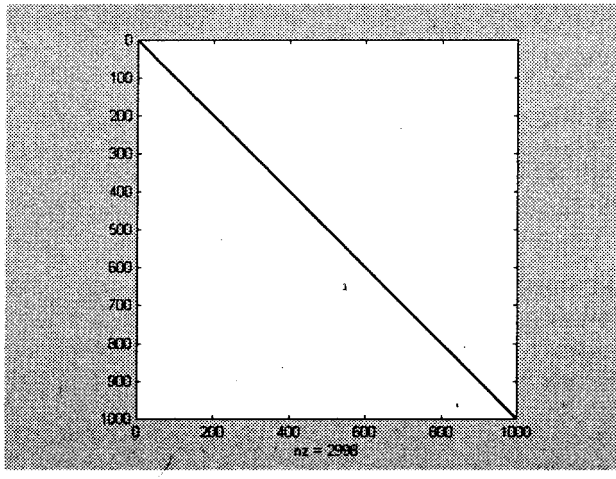


**Рис. 7.5.** Структура разреженной матрицы Hstr

## Нелинейная минимизация с ограничениями в виде линейных равенств

Продолжим рассмотрение задачи с минимизацией введенной функции, добавив в нее ограничения в виде 100 линейных равенств

$$\mathbf{Aeq}\ \mathbf{x} = \mathbf{beq},$$

то есть матрица **Aeq** имеет размер 100×1000.

Будем полагать, что матрица **Aeq** и вектор **beq** содержатся в файле browneq.mat.

**Этап 1.** В данном случае будем использовать ранее подготовленный m-файл brownfgh.

**Этап 2.** Составление программы для поиска решения:

```
» fun = 'brownfgh';
» load browneq % Загрузка матрицы Aeq и вектора beq
» n = 1000;
» xstart = -ones(n,1); xstart(2:2:n) = 1;  % Начальные значения
» options = optimset('GradObj','on','Hessian','on', ...
'PrecondBandWidth', inf);
» % Поиск решения
» [x,fval,exitflag,output] = ...
fmincon('brownfgh',xstart,[],[],Aeq,beq,[],[],[],options);
```

Результаты расчета представляются удовлетворительными (об этом можно судить, например, по величине нормы невязки norm(Aeq*x-beq)):

```
» exitflag
exitflag =
  1
» fval
fval =
  205.9313
» output
output =
  iterations: 16
  funcCount: 16
  cgiterations: 14
  firstorderopt: 2.1434e-004
  algorithm: 'large-scale: projected trust-region Newton'
  The linear equalities are satisfied at x
  norm(Aeq*x-beq)
```

```
ans =
  1.1913e-012
```

### Задача квадратичного программирования при наличии ограничений на диапазоны изменений переменных

Для минимизации квадратичной формы, зависящей от большого числа переменных, следует использовать функцию quadprog. Рассмотрим подобную задачу при числе переменных 400 и симметричной трехдиагональной матрице **H**, сохраненной в файле qpbox1.mat.

Соответствующая программа и результаты вычислений приведены ниже.

```
» load qpbox1 % Загрузка матрицы H
» % Задание граничных значений
» lb = zeros(400,1); lb(400) = -inf;
» ub = 0.9*ones(400,1); ub(400) = inf;
» f = zeros(400,1); f([1 400]) = -2;
» xstart = 0.5*ones(400,1);  % Начальные значения
» % Поиск решения
» [x,fval,exitflag,output] = ...
quadprog(H,f,[],[],[],[],lb,ub,xstart);
» exitflag
exitflag =
  1
» output
output =
  firstorderopt: 7.8435e-006
  iterations: 20
  cgiterations: 1809
  algorithm: 'large-scale: reflective trust-region'
```

## Решение задачи линейного программирования

Рассмотрим задачу линейного программирования в общем виде (см. табл. 7.1), полагая, что все исходные данные содержатся в файле sc50b.mat. Задача включает 48 переменных, 30 неравенств и 20 равенств, так что вполне может быть отнесена к задачам большой размерности.

Текст программы с использованием функции linprog и полученные результаты приведены ниже:

```
» load sc50b
» [x,fval,exitflag,output] = ...
  linprog(f,A,b,Aeq,beq,lb,[],[],optimset('Display','iter'));
```

| Residuals: | Primal Infeas A*x-b | Dual Infeas A'*y+z-f | Duality Gap x'*z | Total Rel Error |
|---|---|---|---|---|
| Iter 0: | 1.50e+003 | 2.19e+001 | 1.91e+004 | 1.00e+002 |
| Iter 1: | 1.15e+002 | 2.94e-015 | 3.62e+003 | 9.90e-001 |
| Iter 2: | 1.16e-012 | 2.21e-015 | 4.32e+002 | 9.48e-001 |
| Iter 3: | 3.23e-012 | 5.16e-015 | 7.78e+001 | 6.88e-001 |
| Iter 4: | 5.78e-011 | 7.61e-016 | 2.38e+001 | .69e-001 |
| Iter 5: | 9.31e-011 | 1.84e-015 | 5.05e+000 | 6.89e-002 |
| Iter 6: | 2.96e-011 | 1.62e-016 | 1.64e-001 | 2.34e-003 |
| Iter 7: | 1.51e-011 | 2.74e-016 | 1.09e-005 | 1.55e-007 |
| Iter 8: | .51e-012 | .37e-016 | 1.09e-011 | 1.51e-013 |

```
Optimization terminated successfully.
» exitflag
exitflag =
     1
» fval
fval =
  -70.0000
» output
output =
      iterations: 8
    cgiterations: 0
       algorithm: 'lipsol'
```

# Некоторые рекомендации по использованию функций пакета

## Использование inline-функции вместо m-файла

В не очень сложных ситуациях вместо создания m-файла для задания целевой функции можно применять inline-функцию, например:

```
f = inline('1./((x - 0.3).^2 + 0.01) + 1./((x - 0.9).^2 + 0.04) - 6');
```

Значение такой функции можно вычислить:

```
» f(2.0)
ans =
   -4.8552
```

или найти ее минимум, например, в диапазоне [3, 4]:

```
» [x, fmin] = fminbnd(f, 3, 4)
x =
    3.9999
fmin =
   -5.8234
```

С помощью inline-функции можно задавать функции не только одного аргумента:

```
» f= inline('sin(x).*xdata +(x.^2).*cos(xdata)','x','xdata')
f =
     Inline function:
     f(x,xdata) = sin(x).*xdata +(x.^2).*cos(xdata)
» x = pi; xdata = pi*[4;2;3];
» f(x, xdata)
ans =
    9.8696
    9.8696
   -9.8696
```

и решать затем соответствующие оптимизационные задачи:

```
» % Предположим, вектор ydata задан
» x = lsqcurvefit(f,x,xdata,ydata)
```

Следующий пример иллюстрирует использование inline-техники для описания задачи оптимизации с произвольным числом аргументов:

```
x = fsolve(inline('x*x*x - [1,2;3,4]'),ones(2,2))
```

## Решение задач максимизации

Функции пакета fminbnd, fminsearch, fminunc, fmincon, fgoalattain, fminimax, lsqcurvefit и lsqnonlin решают задачу *минимизации* целевой функции $f(\mathbf{x})$. Максимизация достигается, если решать задачу минимизации для целевой функции, взятой со знаком «минус», то есть $-f(\mathbf{x})$. Аналогичный результат достигается в задачах линейного и квадратичного программирования: там со знаком «минус» необходимо применять матрицу **H** и вектор **f** (то есть использовать $-\mathbf{H}$ и $-\mathbf{f}$).

## Приведение ограничений-неравенств к стандартному виду

В пакете Optimization Toolbox предполагается некоторая стандартная форма записи ограничений-неравенств: $c_i(\mathbf{x}) \le 0$.

Неравенства иного вида должны быть приведены к данному (умножением обеих частей на $-1$, переносом констант в левую часть и т. п.), например:

$$c_i(\mathbf{x}) \geq 0 \quad \rightarrow \quad -c_i(\mathbf{x}) \leq 0,$$
$$c_i(\mathbf{x}) \geq b \quad \rightarrow \quad -c_i(\mathbf{x}) + b \leq 0.$$

### Введение дополнительных аргументов (глобальные переменные)

В списке аргументов рассмотренных функций пакета могут присутствовать дополнительные, обозначенные через P1, P2 и т. д., например:

```
[x,fval] = fsolve('objfun',x0,options,P1,P2,...)
```

Эти дополнительные аргументы выполняют роль глобальных переменных для вызываемых целевых функций.

Пусть, например, требуется найти нули эллиптической функции Якоби ellipj(u,m). Данная функция зависит от аргумента u и параметра m (указываемого как второй аргумент). Поиск ее нуля вблизи значения $u_0 = 3$ и при $m = 0{,}5$ может быть организован следующим образом:

```
» m = 0.5;
» % Отключение вывода текущей информации
» options = optimset('Display','off');
» x = fsolve('ellipj',3,options,m) % Поиск решения
x =
    3.7081
```

Легко убедиться в том, что найденное решение корректно:

```
» f = ellipj(x,m)
f =
 -3.0042e-008
```

то есть оно действительно является нулем рассматриваемой функции.

Следующий пример относится к использованию функции fgoalattain:

```
» % Задание целевой функции
» fun = inline('sort(eig(A+B*x*C))','x','A','B','C');
» % Поиск решения
» x = fgoalattain(fun, -ones(2,2), [-5,-3,-1], [5, 3, 1],...
  [ ], [ ], [ ], [ ], -4*ones(2), 4*ones(2), [ ], [ ], A, B, C);
```

## Соответствия между версиями пакета 1.5 и 2.0

Версия пакета Optimization Toolbox 2.0, используемая в системе MATLAB 5.3.1, отличается от версии пакета 1.5, применявшегося в составе MATLAB более ранних версий, как по именам функций,

так и по наборам их аргументов и опций, хотя назначение функций аналогично. Соответствия между функциями версий приведены в табл. 7.3; функции optimget и optimset в версии 1.5 отсутствуют. Более подробную информацию по версии 1.5 можно получить, используя справочную систему MATLAB.

**Таблица 7.3.** Соответствия между функциями пакета Optimization версий 1.5 и 2.0

| Имя функции в версии 1.5 | Имя функции в версии 2.0 |
|---|---|
| attgoal | fgoalattain |
| conls | lsqlin |
| constr | fmincon |
| curvefit | lsqcurvefit |
| fmin | fminbnd |
| fmins | fminsearch |
| fminu | fminunc |
| fsolve | fsolve (имя не изменилось) |
| fzero | fzero (имя не изменилось) |
| leastsq | lsqnonlin |
| minimax | fminimax |
| nnls | lsqnonneg |
| lp | linprog |
| qp | quadprog |
| seminf | fseminf |

# Глава 8
# Пакет Statistics Toolbox

## Назначение пакета Statistics Toolbox

Пакет Statistics Toolbox (пакет статистических вычислений) представляет собой набор программ, позволяющих выполнять различные статистические расчеты в рамках системы MATLAB. Данный пакет ориентирован на весьма широкий спектр задач: от генерации случайных чисел и подбора кривых под экспериментальные данные до планирования экспериментов и задач промышленного статистического контроля. Инструментальные средства пакета позволяют использовать как его систему команд в режиме командной строки, так и набор графических интерактивных программ (графический интерфейс пользователя).

В систему MATLAB 5.3.1 входит обновленная версия 2.2 рассматриваемого пакета.

Пакет Statistics Toolbox сам по себе, наверное, уступает специализированным статистическим системам, таким, например, как STATISTICA или STATGRAPHICS, но в основном это касается удобства использования. В то же время нельзя не отметить достаточно серьезные достоинства Statistics Toolbox — это весьма обширный набор его функций (их около 200) и, главное, возможность (с использованием других средств MATLAB) организовывать моделирование стохастических объектов и процессов, в том числе с изменяющимися во времени характеристиками. Классы задач, охватываемые пакетом, видны из содержания этого раздела.

## Распределения вероятностей

Отметим, что необходимые теоретические сведения об используемых статистических методах, понятиях и терминологии широко представлены в литературе, например в следующих монографиях:

1. Айвазян С. А., Мхитарян В. С. Прикладная статистика и основы эконометрики. — М.: ЮНИТИ, 1998. — 1022 с.

2. Афифи Ф., Эйзен С. Статистический анализ: Подход с использованием ЭВМ. — М.: Мир, 1982. — 488 с.

В пакете используются 20 различных типов *распределения вероятностей*, в том числе нормальный, Стьюдента, Фишера, экспоненциальный и т. д. С каждым *законом распределения* связаны:

- плотность распределения вероятностей или функция плотности вероятности (probability density function — pdf) $f(x)$;
- интегральная функция распределения (cumulative distribution function — cdf), $F(x)$;
- функция, обратная к интегральной функции распределения, $F^{-1}(x)$;
- генерация случайных чисел;
- среднее значение и дисперсия как функции параметров распределения;
- функции оценки параметров закона распределения.

## Функции плотности вероятности

Функции данной группы в пакете Statistics Toolbox имеют некоторый стандартный формат, отличающийся только по имени используемого распределения. Следующий пример относится к нормальному закону:

```
» x = [-3:0.5:3];
» f = normpdf(x,0,1)
f =
  Columns 1 through 7
    0.0044    0.0175    0.0540    0.1295    0.2420    0.3521    0.3989
  Columns 8 through 13
    0.3521    0.2420    0.1295    0.0540    0.0175    0.0044
```

В данном случае функция `normpdf(x,0,1)` для заданного `x` возвращает значение функции плотности вероятности со следующими параметрами: математическое ожидание — 0, среднеквадратичное отклонение (корень квадратный из дисперсии) — 1.

Аналогичным образом используются функции, относящиеся к другим распределениям, — необходимо только знать (и задавать) соответствующие параметры выбранного закона. Функции данной подгруппы представлены в табл. 8.1. Их имена образованы путем слияния первых букв названия закона (естественно, по-английски) и «окончания» `pdf`.

**Таблица 8.1.** Функции плотности распределения вероятностей

| Имя функции | Закон распределения |
|---|---|
| betapdf | Бета-распределение |
| binopdf | Биномиальный |
| chi2pdf | Хи-квадрат |
| exppdf | Экспоненциальный |
| fpdf | Фишера |
| gampdf | Гамма-распределение |
| geopdf | Геометрический |
| hygepdf | Гипергеометрический |
| lognpdf | Логнормальный |
| nbinpdf | Отрицательный биномиальный |
| ncfpdf | Нецентральное (смещенное) распределение Фишера |
| nctpdf | Нецентральное (смещенное) распределение Стьюдента |
| ncx2pdf | Нецентральное (смещенное) хи-квадрат распределение |
| pdf | Закон распределения задается как аргумент функции |
| poisspdf | Пуассона |
| raylpdf | Рэлея |
| tpdf | Стьюдента |
| unidpdf | Дискретное равномерное распределение |
| unifpdf | Равномерный (прямоугольный) |
| weibpdf | Вейбулла |



# Функции распределения вероятностей

Функции данной подгруппы возвращают значение связанной с выбранным законом функции распределения вероятностей.

Следующий пример иллюстрирует нахождение $F(x)$ для экспоненциального закона распределения с параметрами MU=1 и x=2:

```
» expcdf(2,1)
ans =
    0.8647
```

Вообще, имена функций этой подгруппы практически идентичны именам функций, приведенным в табл. 8.1, и отличаются только окончанием: не pdf, а cdf.

## Функции, обратные к интегральным функциям распределения

Функции этой подгруппы возвращают значение $F^{-1}(x)$. Имя функции образуется из имени закона распределения и окончания inv.

Пример для экспоненциального распределения:

```
» expinv(0.8647,1)
ans =
    2.0003
```

## Генерация случайных чисел

Команды данной подгруппы возвращают случайные числа с заданным законом распределения. Имя функции образовано путем слияния первых букв названия закона и окончания rnd.

**Примеры:**

1. Генерация одного случайного числа, подчиняющегося экспоненциальному закону распределения с параметром MU=1:

```
» exprnd(1)
ans =
    1.0417
```

2. Генерация группы случайных чисел (подчиняющихся тому же закону распределения), образующих матрицу с 3 строками и 4 столбцами:

```
» exprnd(1,[3 4])
ans =
    0.2068    1.5957    1.3013    0.2920
    4.6191    1.6158    1.6154    0.8095
    1.9741    0.5045    4.1816    0.0706
```

3. То же, что в предыдущем случае, но с несколько иной формой записи команды:

```
» exprnd(1,3,4)
ans =
    0.7636    0.6441    0.1766    0.9690
    0.8707    1.5963    3.9302    0.1842
    0.1670    0.3973    0.3838    0.6875
```

## Среднее и дисперсия как функции распределения

Функции данной подгруппы возвращают математические ожидание и дисперсию указанного распределения в зависимости от его заданных параметров. Имя функции данной подгруппы образовано путем слияния названия закона и окончания stat. Следующий пример иллюстрирует нахождение математического ожидания и дисперсии экспоненциального закона распределения для четырех значений его параметра — 1, 10, 100 и 1000:

```
» [m,v] = expstat([1 10 100 1000])
m =
          1          10         100        1000
v =
          1         100       10000     1000000
```

## Функции оценки параметров закона распределения

Функции данной подгруппы возвращают оценки параметров законов распределений по экспериментальным данным. Имена функций образованы путем слияния названия закона и окончания fit.

**Пример.** Оценивание параметра экспоненциального закона производится следующим образом:

```
» x=exprnd(1,100,1); % Генерация 100 элементов выборки
% Нахождение оценки параметра (точное значение - 1)
» expfit(x)
ans =
    0.9157
```

## Дескриптивная статистика

Функции данной группы позволяют определять набор сводных характеристик исходного множества обрабатываемых данных (выборки): оценки центра группирования значений случайной величины, степени рассеяния, коэффициента корреляции, эксцесса и т. п., иначе говоря, характеристики *описательной* или *дескриптивной статистики*. Полный перечень функций дескриптивной статистики пакета Statistics Toolbox приведен в табл. 8.2. Некоторые из них, вообще говоря, являются встроенными функциями MATLAB (например, corrcoef, cov, mean, median, std).

**Таблица 8.2.** Функции дескриптивной статистики

| Имя функции | Возвращаемая величина |
|---|---|
| corrcoef | Оценка коэффициента корреляции |
| cov | Оценка матрицы ковариаций |
| geomean | Среднее геометрическое |
| harmmean | Среднее гармоническое |
| iqr | Оценка разности между 75- и 25-процентными точками (процентилями), иначе — разность между 3-й и 1-й квартилями |
| kurtosis | Оценка коэффициента эксцесса (от обычно используемого в отечественной литературе параметра $\beta_2$ отличается на величину 3, то есть kurtosis = $\beta_2$ + 3) |
| mad | Среднее абсолютное отклонение от среднего значения |
| mean | Арифметическое среднее |
| median | Медиана |
| moment | Оценка центрального момента произвольного (задаваемого как аргумент) порядка |
| nanmax | Максимальное значение, находимое с игнорированием нечисловых элементов |
| nanmean | Арифметическое среднее, находимое с игнорированием нечисловых элементов |
| nanmedian | Медиана, находимая с игнорированием нечисловых элементов |
| nanmin | Минимальное значение, находимое с игнорированием нечисловых элементов |
| nanstd | Оценка среднеквадратичного отклонения, находимая с игнорированием нечисловых элементов |
| nansum | Сумма, находимая с игнорированием нечисловых элементов |
| prctile | Выборочная процентная точка (процентиль) |
| range | Размах выборки |
| skewness | Оценка коэффициента асимметрии |
| std | Оценка среднеквадратичного отклонения |
| trimmean | Оценка среднего, находимая с игнорированием заданного процента минимальных и максимальных элементов выборки |
| var | Оценка дисперсии |

**Примеры:**

1. В данном примере сначала моделируется (в виде матрицы размером 100×100) совокупность случайных чисел, подчиненных нормальному закону распределения с параметрами, соответственно, 0 и 1, затем по данной выборке находятся оценки среднеквадратичного отклонения и средние абсолютных отклонений от среднего значения (по 100 значений), после чего оценивается относительная эффективность (efficiency) данных величин как оценок степени рассеяния элементов выборки (для корректности подобного сравнения среднее значение абсолютных отклонений умножается на множитель 1,3). Конечный результат говорит о том, что при нормальном законе распределения более эффективной оценкой степени рассеяния является оценка среднеквадратичного отклонения.

```
» x = normrnd(0,1,100,100); % Генерация элементов выборки
» % Нахождение оценки среднеквадратичного отклонения
» s = std(x);
» s_MAD = 1.3 * mad(x);
» efficiency = (norm(s - 1)./norm(s_MAD - 1)).^2
efficiency =
    0.7383
```

2. Следующий пример иллюстрирует нахождение среднего значения для массивов данных, в которых некоторые элементы не определены (пропущены). Исходная матрица данных — «магический квадрат» 3×3, в котором затем 1-й, 6-й и 8-й элементы сделаны нечисловыми.

```
» m = magic(3);
» m([1 6 8]) = [NaN NaN NaN]
m =
   NaN     1     6
     3     5   NaN
     4   NaN     2
» nmean = nanmean(m)
nmean =
    3.5000    3.0000    4.0000
```

3. В данном примере по генерируемой выборке случайных чисел, подчиняющихся нормальному закону с нулевым средним и единичной дисперсией, определяются оценки коэффициентов эксцесса и асимметрии.

```
» X = randn([5 4])
X =
   -0.1096   -0.2009    1.0306   -3.5968
   -0.1121   -0.7846    0.2014   -1.8284
    0.8969    0.8279   -1.2392   -0.6119
    0.7632   -0.2592   -0.5260   -0.7429
    0.1649   -1.5061    0.6314    0.0440
» k = kurtosis(X)
k =
    1.2783    2.1848    1.7258    2.2550
» y = skewness(X)
y =
    0.2855    0.1403   -0.3298   -0.7890
```

## Кластерный анализ

В данную группу входят следующие 9 функций, реализующих ряд операций кластерного анализа.

- Функция `pdist` возвращает парные расстояния между объектами (векторами).

  Запись:

  ```
  Y = pdist(X)
  Y = pdist(X,'metric')
  Y = pdist(X,'minkowski',p)
  ```

  *Описание.* Аргументы функции:

  - `X` — матрица данных, имеющая $m$ строк и $n$ столбцов, рассматриваемая как совокупность $m$ векторов с $n$ элементами каждый;
  - `'metric'` — строковая переменная, принимающая возможные значения:

    `'Euclid'` — евклидово расстояние;

    `'SEuclid'` — нормализованное евклидово расстояние;

    `'Mahal'` — расстояние Махаланобиса;

    `'CityBlock'` — расстояние по Манхэттену (расстояние Хэмминга);

    `'Minkowski'` — расстояние в метрике Минковского;
  - `p` — некоторое фиксированное число, показатель метрики Минковского (по умолчанию — 2).

  Возвращаемая величина `Y` — вектор с $(m-1)m/2$ элементами, которые являются расстояниями, соответственно, между вектором

1 и вектором 2, вектором 1 и вектором 3... вектором 1 и вектором $m$... вектором $m-1$ и вектором $m$.

В следующем примере находятся 6 взаимных евклидовых расстояний между 4 векторами:

```
» X = [1 2; 1 3; 2; 3 1]
X =
     1     2
     1     3
     2     2
     3     1
» Y = pdist(X)
Y =
    1.0000    1.0000    2.2361    1.4142    2.8284    1.4142
```

- Функция `zscore(D)` осуществляет масштабирование (нормализацию) по столбцам элементов матрицы `D`.

  Запись:

  ```
  Z = zscore(D)
  ```

  *Описание.* Расчеты выполняются по формуле `Zi = (Di - mean(Di))./(std(Di))`, где `Zi`, `Di` — столбцы матриц `Z` и `D`.

  Функция используется для предварительного преобразования экспериментальных данных.

- `squareform(Y)` — данная функция преобразует вектор `Y`, возвращаемый функцией `pdist`, в симметричную квадратную матрицу.

  Пример:

  ```
  » Y = [1.0000 1.0000 2.2361 1.4142 2.8284 1.4142];
  » squareform(Y)
  ans =
         0    1.0000    1.0000    2.2361
    1.0000         0    1.4142    2.8284
    1.0000    1.4142         0    1.4142
    2.2361    2.8284    1.4142         0
  ```

- Функция `linkage(Y)` возвращает иерархическое дерево кластеров, используя по умолчанию алгоритм «ближайшего соседа». Является исходной для ряда других функций, в частности для функции построения дендрограммы `dendrogram` (см. ниже).

  Запись:

  ```
  Z = linkage(Y)
  Z = linkage(Y,'method')
  ```

*Описание.* Аргументами функции являются вектор `Y` расстояний между $m$ объектами, возвращаемый функцией `pdist`, и строковая переменная `'method'` (метод), которая задает метод кластеризации и может принимать одно из следующих значений:

- `'single'` — алгоритм «ближайшего соседа»;
- `'complete'` — алгоритм «дальнего соседа»;
- `'average'` — алгоритм «средней связи»;
- `'centroid'` — центроидный алгоритм, использующий расстояние по «центрам тяжести» групп;
- `'ward'` — пошаговый алгоритм.

Функция возвращает матрицу `Z`, имеющую $m - 1$ строку и 3 столбца и содержащую информацию об иерархическом дереве кластеров. Нижний уровень иерархии образован $m$ исходными объектами (векторами), которые затем объединяются попарно (в зависимости от расстояния между ними), образуя новые кластеры, которые также могут объединяться попарно (в зависимости от взаимного расстояния), и т. д. Каждому новому формируемому кластеру, соответствующему $i$-й строке матрицы `Z`, присваивается индекс $m+i$ ($m$ — число исходных объектов). Столбцы 1 и 2 матрицы `Z` содержат индексы объектов (кластеров), которые были объединены в пары при образовании нового кластера, а столбец 3 — расстояние между данными объектами. Всего подобным образом будет организован $m - 1$ кластер.

Пример.

```
» X = [3 1.7; 1 1; 2 3; 2 2.5; 1.2 1; 1.1 1.5; 3 1];
» Y = pdist(X);
» Z = linkage(Y)
Z =
   2.0000    5.0000    0.2000
   3.0000    4.0000    0.5000
   8.0000    6.0000    0.5099
   1.0000    7.0000    0.7000
  11.0000    9.0000    1.2806
  12.0000   10.0000    1.3454
```

Результаты расчетов показывают, что кластеры образованы объектами (векторами) 2 и 5, 3 и 4, 1 и 7, новым объектом с индексом 8 и объектом 6 и т. д.

Функция `dendrogram(Z)` возвращает графическое отображение результата выполнения функции `linkage`.

Пример использования функции приведен на рис. 8.1.

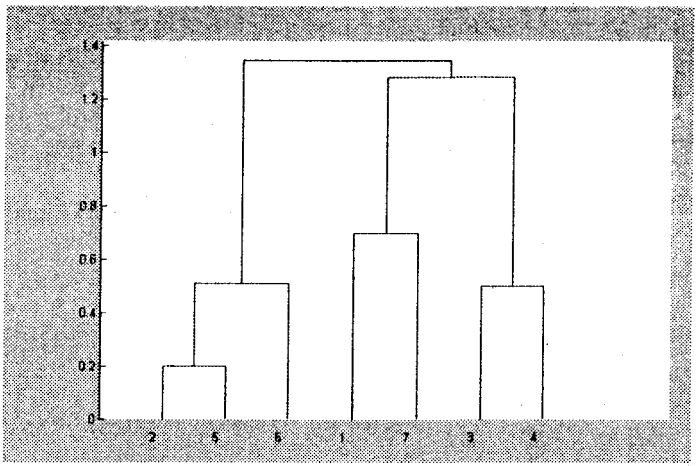


**Рис. 8.1.** Пример дендрограммы

- Функция `cophenet(Z,Y)`, где `Y` и `Z` — матрицы, возвращаемые, соответственно, функциями `pdist` и `linkage`, возвращает аналог коэффициента корреляции, характеризующий качество (чем ближе к 1, тем лучше) разбиения исходных объектов на дерево кластеров.

  Пример. Применение данной функции к условиям предыдущего примера дает результат, который можно оценить как удовлетворительный:

```
» cophenet(Z,Y)
ans =
    0.7514
```

- Функция `inconsistent(Z)` возвращает значения так называемых коэффициентов несовместимости для каждой связи иерархического дерева кластеров. Аргумент — матрица `Z`, возвращаемая функцией `linkage`. Может использоваться для оценки качества разбиения на кластеры исходного множества объектов.
- Функция `cluster` осуществляет разбиение иерархического дерева кластеров на отдельные кластеры.

  Запись:

```
T = cluster(Z,cutoff)
T = cluster(Z,cutoff,depth)
```

*Описание.* Аргументы: `Z` — матрица, возвращаемая функцией `linkage`; `cutoff` при значении от 0 до 1 воспринимается как пороговая величина для выделения отдельных кластеров путем исключения связей дендрограммы, если соответствующие коэффициенты несовместимости превышают данный порог, а при значении больше 1 — как задаваемое число кластеров; `depth` — аргумент, определяющий «глубину» иерархии, для которой подсчитываются коэффициенты несовместимости.

Возвращаемая величина — вектор `T`, элементы которого являются номерами кластеров, к которым отнесены исходные объекты.

Пример. В условиях ранее рассмотренного примера имеем

```
» z=[2.0000 5.0000 0.2000;
     3.0000 4.0000 0.5000;
     8.0000 6.0000 0.5099;
     1.0000 7.0000 0.7000;
    11.0000 9.0000 1.2806;
    12.0000 10.0000 1.3454];
» T = cluster(z,3)
T =
     1
     3
     2
     2
     3
     3
```

Здесь из полученного иерархического дерева выделено 3 кластера, при этом вектор 1 отнесен к первому кластеру, вектор 2 — к третьему и т. д.

- Функция `T = clusterdata(X,cutoff)` осуществляет кластеризацию аналогично предыдущей функции, но по отношению к исходной матрице данных `X`.

## Линейные модели

Функции данной группы (в табл. 8.3 приведены основные функции) реализуют процедуры одно- и двухфакторного дисперсионного анализа и линейного (по параметрам модели) регрессионного анализа.

**Таблица 8.3.** Функции линейных моделей

| Имя функции | Реализуемая процедура |
|---|---|
| anova1 | Однофакторный регрессионный анализ |
| anova2 | Двухфакторный регрессионный анализ |
| lscov | Линейная регрессия при заданной матрице ковариаций (встроенная функция MATLAB) |
| polyconf | Определения доверительных интервалов для линии регрессии, возвращаемой функцией polyfit |
| polyfit | Полиномиальная регрессия (встроенная функция MATLAB) |
| polyval | Прогноз с использованием полиномиальной регрессии (встроенная функция MATLAB) |
| regress | Множественная линейная регрессия |
| ridge | Линейная регрессия с применением гребневых оценок (ридж-оценок) |
| rstool | Интерактивный подбор и визуализация поверхности отклика |
| stepwise | Пошаговая регрессия (графический интерфейс пользователя) |

Большинство приведенных функций реализуют хорошо известные статистические методы обработки данных, поэтому остановимся подробнее только на двух наиболее интересных последних функциях.

## Функция rstool

Запись:

```
rstool(x,y)
rstool(x,y,'model')
rstool(x,y,'model',alpha,'xname','yname')
```

*Описание.* Данная функция строит и вызывает графическое отображение линейной по параметрам (но не по факторам!) регрессионной модели. Аргументы функции:

- x, y — матрица и вектор экспериментальных данных, соответственно, для факторов и отклика;
- 'model' — строковая переменная, задающая вид начальной модели регрессии и принимающая значения 'interaction' (модель содержит константу, линейные члены и парные взаимодействия факторов), 'quadratic' (модель содержит парные взаимодействия и квадратичные члены), 'purequadratic' (модель содержит константу, линейные члены, парные взаимодействия и квадратичные члены);

- `alpha` — параметр, задающий доверительный коридор для прогнозируемых значений, равный 100(1 − `alpha`) %;
- `'xname'`, `'yname'` — строковые переменные, задающие наименования осей графика.

Используя меню появляющегося графического окна, можно менять вид модели, экспортировать выбранные параметры в рабочее пространство MATLAB и т. п.

Пример использования функции иллюстрируется приведенным ниже фрагментом MATLAB-программы и рис. 8.2 (в данном случае речь идет о восстановлении квадратичной зависимости).

```
» X=[-1; -0.5; 0; 0.5; 1];
» y=[1; 0.25; 0; 0.25; 1];
» rstool(X,y,'purequadratic')
```

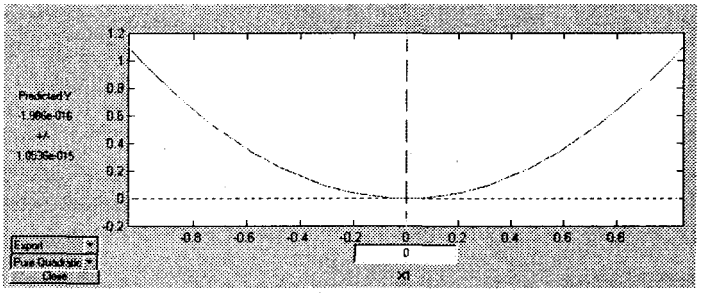


**Рис. 8.2.** Графический интерфейс функции rstool

## Функция stepwise

Функция `stepwise` реализует пошаговую регрессию с использованием средств графического интерфейса.

Запись:

```
stepwise(X,y)
stepwise(X,y,inmodel)
stepwise(X,y,inmodel,alpha)
```

*Описание.* Аргументы `X`, `y` и `alpha` имеют тот же смысл, что и для предыдущей функции, `inmodel` — вектор, элементы которого указывают номера переменных (столбцов матрицы `X`), включенных в регрессию. При выполнении функции открываются три графических окна (коэффициенты регрессии, таблица значений, график истории по-

строения модели), позволяющих контролировать качество регрессионной модели, введение и удаление отдельных факторов.

Следующий фрагмент MATLAB-программы и рис. 8.3 иллюстрируют пример применения данной функции:

```
» load hald
» stepwise(ingredients,heat)
```

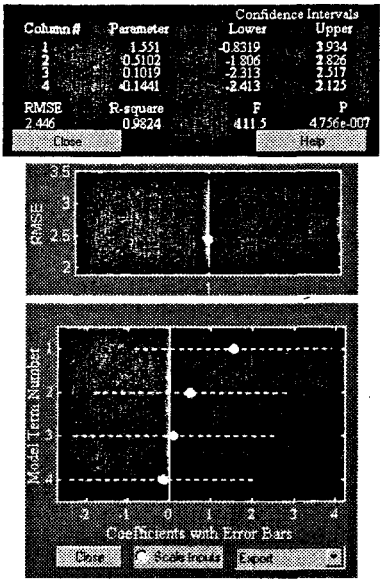


**Рис. 8.3.** Графический интерфейс функции stepwise

Заметим, что «включение» в модель и «выключение» из нее отдельных регрессоров производится в окне «Коэффициенты регрессии» (нижнее окно на рис. 8.3) с использованием мыши — выделением линии, соответствующей выбранному регрессору (вначале введенными в модель считаются все факторы). В окне «История построения модели» каждое изменение в наборе регрессоров (то есть каждый шаг построения наилучшей модели) отображается точкой, характеризующей величину ошибки модели; адекватные изменения происходят и в окне «Таблица значений». Пошаговый интерактивный отбор регрессоров в конце концов приводит к получению наиболее экономной и точной модели (при умеренном числе исходных факторов).

## Нелинейные регрессионные модели

Функции данной группы позволяют определять оценки параметров нелинейных моделей, постулируемых для имеющихся экспериментальных данных. Группа содержит следующие функции:

- `nlinfit` — реализует нелинейный метод наименьших квадратов — МНК (метод Гаусса—Ньютона);
- `nlintool` — возвращает график прогнозируемых значений (прямой аналог функции `rstool`);
- `nlparci` — возвращает вектор доверительных интервалов для оценок параметров модели;
- `nlpredci` — возвращает прогнозируемые значения нелинейной модели;
- `nnls` — метод МНК, возвращающий только неотрицательные значения параметров (*встроенная функция* MATLAB).

В качестве примера рассмотрим одну из функций — `nlinfit`.

Запись:

```
[beta,r,J] = nlinfit(X,y,'model',beta0)
```

*Описание.* Аргументы:

- `X`, `y` — исходные наборы экспериментальных данных;
- `'model'` — задаваемая пользователем функция вида $y = f(\mathbf{beta}, x)$;
- `beta0` — начальное значение вектора параметров модели `beta`.

Возвращаемые значения:

- `beta` — вектор параметров модели;
- `r` — остатки модели;
- `J` — якобиан (для использования в функции `nlintool`).

## Проверка гипотез

Следующие 6 функций позволяют реализовывать определенный набор процедур проверки статистических гипотез:

- `ranksum` — реализует ранговый критерий Вилкоксона для проверки однородности двух генеральных совокупностей;
- `signrank` — реализует знаковый критерий Вилкоксона для проверки гипотезы о равенстве медиан двух выборок;
- `signtest` — реализует проверку гипотезы о равенстве медиан двух выборок;

- ztest — реализует проверку гипотезы, что математическое ожидание и дисперсия выборки равны заданным величинам (предполагается нормальный закон распределения для элементов выборки);
- ttest — реализует проверку гипотезы, что математическое ожидание выборки равно заданному значению (дисперсия — неизвестна, для элементов выборки предполагается нормальный закон распределения);
- ttest2 — реализует проверку гипотезы о равенстве математических ожиданий двух выборок при неизвестных, но одинаковых дисперсиях (предполагается нормальный закон распределения).

Следующий пример иллюстрирует применение функции ranksum.

```
» x = poissrnd(5,10,1); % Генерация первой выборки
» y = poissrnd(2,20,1); % Генерация второй выборки
» % Проверка гипотезы об однородности выборок
» [p,h] = ranksum(x,y,0.05)
p =
    0.0010
h =
     1
```

В данном случае возвращаемые величины: p — вероятность того, что выборки однородны, h — индикатор принятия гипотезы (его значение, равное 1, говорит о том, что нуль-гипотезу об однородности выборок следует отвергнуть при заданном уровне значимости 0,05).

## Многомерные статистики

Данная группа состоит из следующих двух функций:

- d = mahal(Y,X). Функция возвращает матрицу, элементами которой являются расстояния Махаланобиса от векторов (строк) матрицы Y до векторов (строк) матрицы X. Число строк данных матриц может быть различным, но число столбцов должно совпадать; число строк матрицы X должно превышать число столбцов.

  Расстояние Махаланобиса часто используется как мера близости (сходства) объектов в задачах кластеризации и распознавания образов.

- class = classify(sample,training,group). Функция реализует процедуру линейного дискриминантного анализа.

  *Описание.* Аргументы:

  - sample — матрица анализируемых объектов (векторов-строк);

- training — матрица, вектор-строки которой разбиты на несколько классов; число столбцов этой матрицы совпадает с числом столбцов матрицы sample;
- group — вектор, элементы которого указывают на принадлежность каждого вектора матрицы training к определенному классу; *число его элементов (строк) должно совпадать с числом* строк матрицы training.

Возвращаемая величина — вектор class с числом элементов, равным числу строк матрицы sample. Элементы этого вектора указывают на принадлежность объектов к тому или иному классу.

## Метод главных компонент

*Четыре функции данной группы реализуют процедуры статистического* метода главных компонент.

- pcacov — функция реализует метод главных компонент по заданной матрице ковариаций.

  Запись:

```
pc = pcacov(X)
[pc,latent,explained] = pcacov(X)
```

  *Описание.* Аргумент X — матрица ковариаций экспериментальных данных; возвращаемые величины:

  - pc — *матрица, образованная векторами — главными компонентами;*
  - latent — вектор, элементами которого являются собственные числа X, соответствующие данным компонентам;
  - explained — вектор, элементами которого являются процентные *доли общей дисперсии, объясняемые главными компонентами.*

  Пример.

```
» load hald % Загрузка экспериментальных данных
» covx = cov(ingredients); % Вычисление матрицы ковариаций
» [pc,variances,explained] = pcacov(covx)  % Реализация метода
pc =
    0.0678   -0.6460    0.5673   -0.5062
    0.6785   -0.0200   -0.5440   -0.4933
   -0.0290    0.7553    0.4036   -0.5156
   -0.7309   -0.1085   -0.4684   -0.4844
```

```
variances =
  517.7969
   67.4964
   12.4054
    0.2372
explained =
   86.5974
   11.2882
    2.0747
    0.0397
```

- `princomp` — функция подобна предыдущей, но реализует метод главных компонент по исходной матрице данных.
- `residuals = pcares(X,ndim)` — функция возвращает остаток после удаления `ndim` главных компонент. Здесь `X` — исходная матрица данных, величина `ndim` не должна превышать числа столбцов этой матрицы.
- `barttest` — функция реализует тест Бартлетта определения размерности (числа главных компонент) для объяснения неслучайных вариаций в исходных экспериментальных данных.

# Статистические графики

Функции данной группы позволяют графически отображать различные вероятностные характеристики. Список функций приведен в табл. 8.4.

**Таблица 8.4.** Функции статистических графиков

| Имя функции | Реализуемая операция |
|---|---|
| `boxplot` | Построение графика в виде «ящика с усами» (box and whisker plot) |
| `fsurfht` | Интерактивная операция построения контурного графика заданной функции |
| `gline` | Рисование линии в текущей графической фигуре |
| `gname` | Нанесение меток на график |
| `lsline` | Нанесение линии, наиболее близкой (в смысле МНК) к точкам графика |
| `normplot` | Нормальный вероятностный график |
| `qqplot` | График типа «квантиль—квантиль» для двух выборок |
| `refcurve` | Наносит полиномиальную кривую на текущий график |

| Имя функции | Реализуемая операция |
|---|---|
| refline | Наносит прямую линию на текущий график |
| surfht | Контурный график по матрице данных |
| weibplot | Вероятностный график Вейбулла |

Примеры.

1. Использование функции boxplot.

```
» x = normrnd(6,1,100,1); % Генерация выборки
» boxplot(x) % Построение графика
```

Результат использования функции приведен на рис. 8.4.

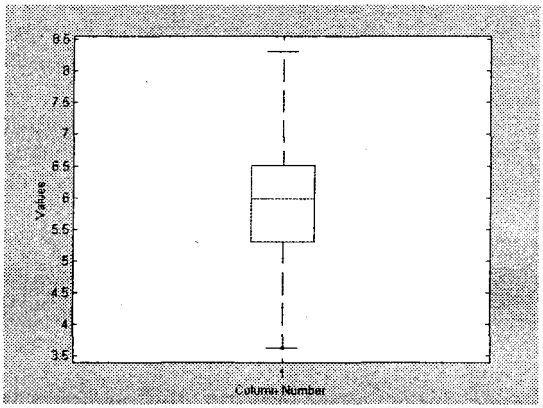


**Рис. 8.4.** График в виде «ящика с усами»

На графике нижняя сторона «ящика» соответствует 25-процентной точке, верхняя — 75-процентной, горизонтальная линия внутри «ящика» — медиане, нижний «ус» — минимальному элементу выборки, верхний — ее максимальному элементу.

2. Использование функции lsline.

```
% Задание экспериментальных точек
» y = [2 3.4 5.6 8 11 12.3 13.8 16 18.8 19.9]';
» plot(y,'+'); % График точек
» % Нанесение аппроксимирующей прямой
» lsline;
```

Результат использования функции иллюстрируется рис. 8.5.

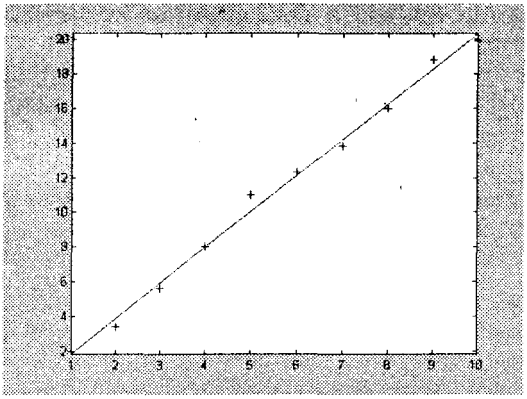

**Рис. 8.5.** *Результат использования функции lsline*

3. Использование функции qqplot. График, возвращаемый данной функцией, является удобным инструментом для выяснения ответа на вопрос, принадлежат ли две выборки к одному и тому же (не обязательно гауссовому) распределению.

```
» x = poissrnd(10,50,1); % Генерация первой выборки
» y = poissrnd(5,100,1); % Генерация второй выборки
» qqplot(x,y); % Построение графика
```

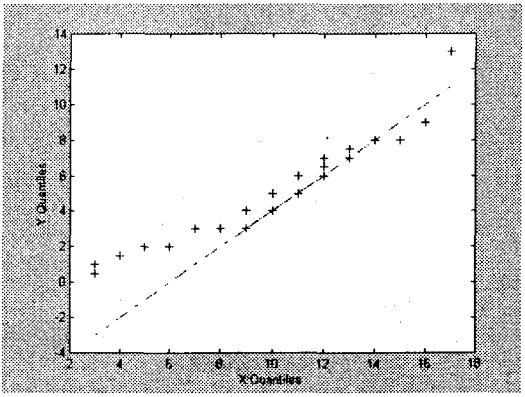


**Рис. 8.6.** Результат использования функции qqplot

На графике крестиками обозначены точки, соответствующие одинаковым процентилям обеих выборок. Сплошной линий со-

единены 25-процентные и 75-процентные процентили. Пунктирная линия является продолжением данной сплошной. Чем ближе все точки графика к данным прямым, тем с большей уверенностью можно утверждать, что выборки соответствуют одинаковому закону распределения. В рассматриваемом случае, несмотря на различия в параметрах закона и в числе элементов выборок, данное утверждение может быть принято.

## Статистический контроль в промышленности

Функции данной группы реализуют известные статистические процедуры контроля качества продукции (см., например, книгу «Контроль качества с помощью персональных компьютеров» / Т. Макино и др. — М.: Машиностроение, 1991. — 224 с.):

- `histfit` — функция возвращает гистограмму, построенную по элементам выборки с наложенной на нее кривой плотности вероятности нормального закона распределения.

  Пример:

```
» r = normrnd(10,1,100,1);
» histfit(r)
```

  Рисунок 8.7 отображает результат использования функции.

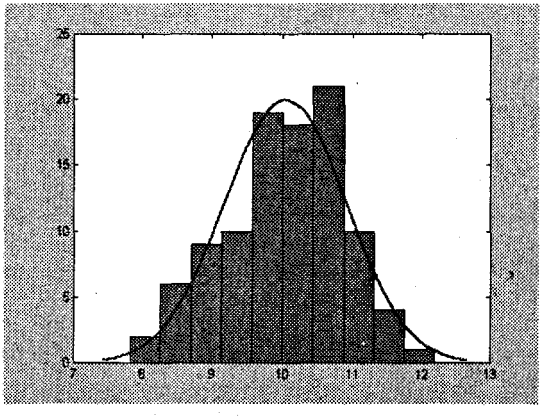


**Рис. 8.7.** Результат использования функции histfit

- `capable` — функция возвращает оценку вероятности нахождения элементов выборки вне заданного интервала, а также оценку дисперсии, реализуя технику контрольных карт.
- `capaplot` — функция «подгоняет» имеющиеся экспериментальные данные под нормальный закон распределения, возвращает соответствующий график и вероятность того, что любое последующее наблюдение находится в заданном интервале.

  Пример:

```
» data = normrnd(1,1,30,1); % Генерация выборки
» p = capaplot(data,[-3 3]) % Построение графика и расчет вероятности
p =
    0.9948
```

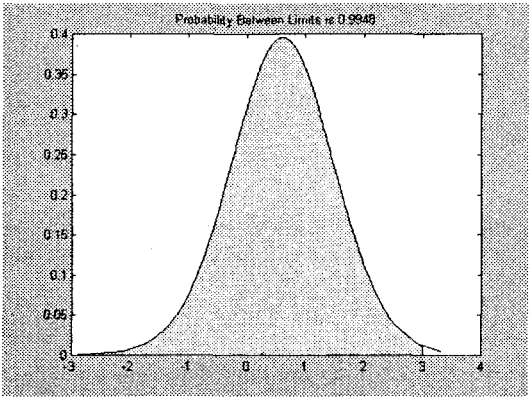


**Рис. 8.8.** График, возвращаемый функцией capaplot

- `normspec` — функция возвращает график плотности распределения *нормального закона с заданными параметрами и для заданного* диапазона изменения аргумента.
- `ewmaplot` — возвращает график экспоненциально сглаженных значений исходной выборки, которые рассматриваются как значения дискретной последовательности.
- `schart` — возвращает график изменения во времени среднеквадратичного отклонения изучаемой последовательности, представленной выборкой данных; разновидность контрольной карты.
- `xbarplot` — функция аналогична предыдущей, но относительно среднего значения.

## Планирование эксперимента

Зачастую можно обеспечить существенную экономию временных и финансовых затрат, организуя активный изучающий (идентифицирующий) эксперимент на объекте некоторым оптимальным образом, заранее планируя его (см. книгу Хартман К. и др. Планирование эксперимента в исследовании технологических процессов. — М.: Мир, 1977. — 552 с.). Для быстрого автоматического нахождения такого наилучшего плана предназначены функции рассматриваемой группы.

- `ff2n` — формирует план полного факторного эксперимента при факторах, каждый из которых задан на отрезке.

  Пример использования (для трех входных факторов):

```
» ff2n(3)
ans =
     0     0     0
     0     0     1
     0     1     0
     0     1     1
     1     0     0
     1     0     1
     1     1     0
     1     1     1
```

  Возвращаемая величина — матрица плана эксперимента.

- `fullfact` — формирует план полного факторного эксперимента при произвольном (задаваемом) числе уровней факторов.

  Пример (для двух входных факторов, каждый из которых имеет по три уровня):

```
» d = fullfact([3 3])
d =
     1     1
     2     1
     3     1
     1     2
     2     2
     3     2
     1     3
     2     3
     3     3
```

Здесь возвращена матрица плана эксперимента d, каждая строка которой соответствует одному из опытов плана, а столбцы — факторам. Например, четвертая строка матрицы соответствует опыту, в котором 1-й фактор должен располагаться на своем первом уровне, а 2-й — на втором.

- hadamard — данная функция возвращает матрицу Адамара, которая соответствует плану дробного факторного эксперимента в случае факторов, каждый из которых задан на отрезке [−1, 1], и построения линейной регрессионной модели.

Пример:

```
» X = hadamard(8)
X =
     1     1     1     1     1     1     1     1
     1    -1     1    -1     1    -1     1    -1
     1     1    -1    -1     1     1    -1    -1
     1    -1    -1     1     1    -1    -1     1
     1     1     1     1    -1    -1    -1    -1
     1    -1     1    -1    -1     1    -1     1
     1     1    -1    -1    -1    -1     1     1
     1    -1    -1     1    -1     1     1    -1
```

Следующие четыре функции связаны с построением так называемых D-оптимальных планов эксперимента:

- rowexch — функция нахождения точного D-оптимального плана эксперимента.

Запись:

```
settings = rowexch(nfactors,nruns)
[settings,X] = rowexch(nfactors,nruns)
[settings,X] = rowexch(nfactors,nruns,'model')
```

*Описание.* Аргументы: nfactors — число факторов, nruns — число опытов (число строк матрицы плана эксперимента), 'model' — строковая переменная, задающая тип регрессионной модели и принимающая одно из следующих возможных значений: 'interaction' (неполная квадратичная модель), 'quadratic' (в модель включаются взаимодействия факторов и квадратичные члены) и 'purequadratic' (в модель включаются константа, линейные и квадратичные члены).

Возвращаемые величины: settings — матрица изменений уровней факторов, X — полная матрица эксперимента.

Пример:

```
» [s,X]=rowexch(3,8,'interaction')
s =
    -1     1    -1
     1    -1    -1
     1    -1     1
     1     1     1
    -1     1     1
     1     1    -1
    -1    -1     1
    -1    -1    -1
X =
     1    -1     1    -1    -1     1    -1
     1     1    -1    -1    -1    -1     1
     1     1    -1     1    -1     1    -1
     1     1     1     1     1     1     1
     1    -1     1     1    -1    -1     1
     1     1     1    -1     1    -1    -1
     1    -1    -1     1     1    -1    -1
     1    -1    -1    -1     1     1     1
```

В условиях примера задано 3 фактора, 8 опытов, постулируемая регрессионная модель содержит константу, три линейных слагаемых, два парных произведения факторов и одно тройное произведение. В частности, в первом опыте 1-й фактор должен быть установлен на нижнем уровне, 2-й — на верхнем, 3-й — на нижнем и т. п.

- `cordexch` — функция нахождения точного D-оптимального плана эксперимента, практически аналогична предыдущей за исключением алгоритма нахождения плана.

  Пример. Нахождение матрицы плана для двухфакторной квадратичной модели

```
» settings = cordexch(2,9,'quadratic')
settings =
     1     1
     1    -1
    -1    -1
     0    -1
     1     0
     0     0
    -1     1
     0     1
    -1     0
```

- daugment — функция возвращает матрицу плана, дополняющую матрицу заданного плана до D-оптимального.
- dcovary — функция для построения D-оптимальных блочных планов.

# Демонстрационные примеры

Для ознакомления с некоторыми возможностями пакета Statistics Toolbox можно использовать следующие функции (команды):

- disttool — возвращает графический интерфейс пользователя, позволяющий выводить графики функций плотности вероятности и функций распределения вероятностей для 19 теоретических законов распределения (рис. 8.9).
- randtool — возвращает графический интерфейс пользователя для генерации случайных чисел с заданным законом распределения и построения соответствующей гистограммы (рис. 8.10).

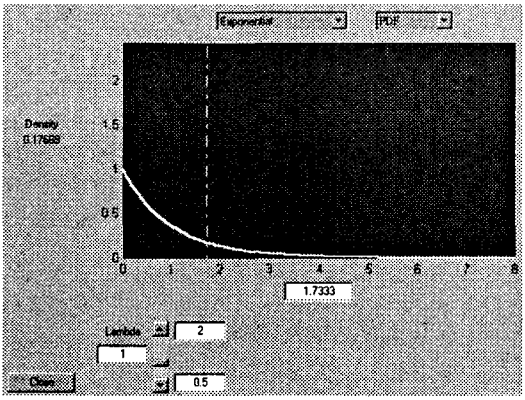


**Рис. 8.9.** Графический интерфейс функции disttool

- polytool — возвращает графический интерфейс пользователя, позволяющий подбирать аппроксимирующую кривую (полином) для заданных экспериментальных данных.

  Пример:

```
» x=[-1; -0.5; 0; 0.5; 1]; % Задание значений аргумента
» y=[1; 0.25; 0; 0.25; 1]; % Задание значений функции
» polytool(x,y) % Вызов графического интерфейса подбора кривых
```

  Рисунок 8.11 иллюстрирует результат использования функции в условиях примера.

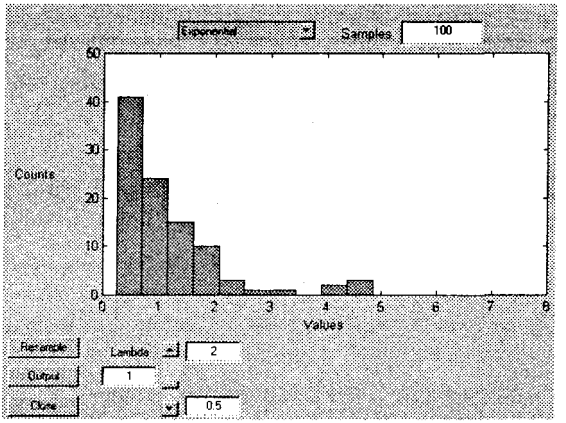


**Рис. 8.10.** Графический интерфейс функции randtool

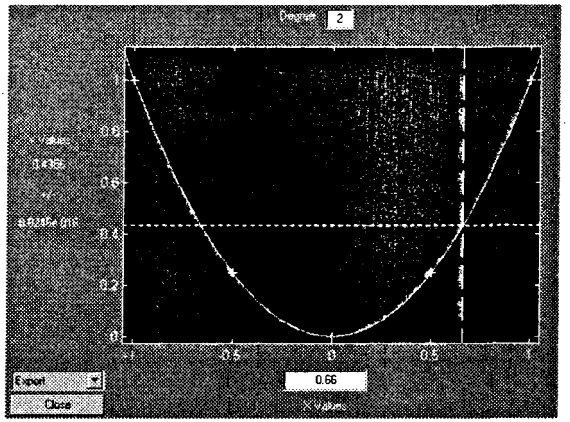


**Рис. 8.11.** Графический интерфейс функций polytool

- `rsmdemo` — функция моделирования функционирования и построения (в режиме графического интерфейса) нелинейной регрессионной модели химической реакции.

## Функции записи/чтения файлов данных

В данную группу входят следующие функции:

- `tblread` — чтение данных в табличном формате.

Запись:

```
[data,varnames,casenames] = tblread
[data,varnames,casenames] = tblread('filename')
[data,varnames,casenames] = tblread('filename','delimiter')
```

*Описание.* Аргументы: `'filename'` — имя файла данных, `'delimiter'` — строковая переменная, принимающая значения `'tab'` (разделители — табуляторы), `'space'` (разделители — пробелы) или `'comma'` (разделители — запятые).

Возвращаемые величины: `data` — числовая матрица данных, `varnames` — вектор с именами переменных, `casenames` — вектор с именами строк (опытов или случаев).

Пример:

```
» [data,varnames,casenames] = tblread('sat.dat')
data =
   470   530
   520   480
varnames =
Male
Female
casenames =
Verbal
Quantitative
```

В данном случае матрица данных отображает две переменных с именами `Male` и `Female` и два «опыта» или «случая» с именами `Verbal` и `Quantitative`.

- `tblwrite` — функция записи данных в файл; является «обратной» к предыдущей функции.

  Использование данной функции проиллюстрируем продолжением предыдущего примера:

```
» tblwrite(data,varnames,casenames,'sattest.dat')
» type sattest.dat
                Male  Female
Verbal          470       530
Quantitative    520       480
```

- `caseread` — функция чтения данных из файла; возвращает матрицу данных вместе с соответствующими именами.

  Пример:

```
» caseread('sattest.dat')
```

```
ans =
                Male  Female
Verbal          470       530
Quantitative    520       480
```

- `casewrite` — функция записи данных в файл вместе с соответствующими именами; является обратной к предыдущей.

# Алфавитный указатель

## F



## H



## I



## J



## L



## M

## N



## O



## P



## Q



## R



## S

## T



## U



## V



## W



## Z



## А



## В

## Г



## Д



## З



## И



## К



## Л



## М

## Н



## О

## П



## Р

## С



## Т



## Ф

## Ц—Ч

# Краткое содержание

# Содержание