

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РЕСПУБЛИКИ КАЗАХСТАН

КАЗАХСКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени К.И. САТПАЕВА

О.В. Киселева

КОМПЬЮТЕРНЫЕ ВЫЧИСЛЕНИЯ

Рекомендовано Республиканским учебно-методическим
объединением в качестве учебного пособия

Алматы 2005

УДК 519.682
ББК 32.973 Я73

Киселева О.В. Компьютерные вычисления: Учеб. пособие.
К.44 – Алматы: КазНТУ, 2005. 109 с.
ISBN 9965 – 758 – 18 – 2

Учебное пособие содержит систематическое изложение вопросов, связанных с выполнением математических расчетов с помощью компьютера. Значительное место уделено методике компьютерного вычисления с использованием редактора «Matlab».

Учебное пособие предназначено для студентов младших курсов и магистрантов, специализирующихся в области компьютерных вычислений.

Табл. 4. Библиогр.–13 назв.

Рецензент *М.К. Шукаев*, д-р техн.наук, проф.

Печатается по плану издания Министерства образования и науки Республики Казахстан на 2005 г.

К $\frac{1602120000}{00(05)-05}$

ББК 32. 973 Я73

ISBN 9965 – 758 – 18 –2

© КазНТУ, 2005

О.В. КИСЕЛЕВА

КОМПЬЮТЕРНЫЕ ВЫЧИСЛЕНИЯ

ВВЕДЕНИЕ

Учебное пособие предназначено для изучения студентами дисциплины «Компьютерные вычисления», согласно рабочей программе, данная дисциплина включает в себя изучение редактора «Matlab».

Данное учебное пособие содержит систематическое изложение вопросов, связанных с выполнением математических расчетов с помощью компьютера. В методике компьютерного вычисления значительное место уделено использованию редактора «Matlab».

В учебном пособии рассмотрены принципы реализации численных методов алгебры и анализа, методы, используемые при работе с интегрированными пакетами математических вычислений, возможности реализации статистической обработки и использования численных методов, а также наиболее актуальные темы для компьютерного изучения. Работа с графиками; численные методы; статистическая обработка данных; работа с векторами и массивами полностью раскрыты для лучшего усвоения материала студентами.

Учебное пособие предназначено для углубленного изучения студентами по специальностям:

3701 – Компьютерные системы обработки информации и управления,

3704 – Программное и аппаратное обеспечение в вычислительной технике,

0913 - Информационные системы по отраслям.

В наши дни компьютерная математика получила должную известность и интенсивно развивается как передовое научное направление на стыке математики и информатики.

Программируемые микрокалькуляторы и персональные компьютеры уже давно применяются для математических расчетов. Для подготовки программ использовались различные универсальные языки программирования. В начале 90-х гг. на смену им пришли специализированные системы компьютерной математики.

MATLAB представляет собой хорошо апробированную и надежную СКМ, рассчитанную на решение самого широкого круга математических задач с представлением данных в универсальной (но не навязываемой пользователям) матричной форме.

Система MATLAB предлагается в системе военно-промышленного комплекса, в аэрокосмической отрасли и автомобилестроении, язык программирования *высокого уровня* для технических вычислений с большим числом стандартных пакетов. прикладных программ. Система MATLAB 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. Около миллиона легально зарегистрированных пользователей уже применяют эту систему. Ее охотно используют в своих научных проектах ведущие университеты и научные центры мира. Популярности системы способствует ее мощное расширение Simulink, предоставляющее удобные и простые средства, в том числе визуальное объектно-ориентированное программирование, для моделирования линейных и нелинейных динамических систем, а также множество других пакетов расширения системы.

Возможности новейшей версии MATLAB 6

Новейшая версия системы MATLAB 6 не только имеет перечисленные выше возможности предшествующих версий, но и характеризуется целым рядом новых и важных возможностей:

- доведенное до более чем 600 число встроенных функций и команд;
- новый интерфейс с набором инструментов для управления средой, включающий в себя окно команд (Command Window), окно истории команд (Command History), браузер рабочей области (Workspace Browser) и редактор массивов (Array Editor);
- новые инструменты, позволяющие при помощи мыши интерактивно редактировать и форматировать графики, оптимизировать их коды и затраты памяти на графические команды и атрибуты;
- улучшенные алгоритмы на основе оптимизированной библиотеки LAPACK;
- новая библиотека FFTW (быстрых преобразований Фурье) Массачусетского технологического института Кембриджского университета (США);
- ускоренные методы интегральных преобразований;
- новые, более мощные и точные алгоритмы интегрирования дифференциальных уравнений и квадратур;
- новые современные функции визуализации: вывод на экран двумерных изображений, поверхностей и объемных фигур в виде прозрачных объектов;
- новая инструментальная панель Camera для управления перспективой и ускорения вывода графики с помощью OpenGL;

- новый интерфейс для вызова Java-процедур и использования Java-объектов непосредственно из MATLAB;
- новые, современные инструменты проектирования графического пользовательского интерфейса.

ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС MATLAB

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.

Перечислим назначение всех кнопок панели инструментов:

- New M-file (Новый m-файл) — выводит пустое окно редактора m-файлов;
- Open file (Открыть файл) — открывает окно для загрузки m-файла;
- Cut (Вырезать) — вырезает выделенный фрагмент и помещает его в буфер;
- Copy (Копировать) — копирует выделенный фрагмент в буфер;
- Paste (Вставить) — переносит фрагмент из буфера в текущую строку ВВОДА;
- Undo (Отменить) — отменяет предшествующую операцию;
- Redo (Повторить) — восстанавливает последнюю отмененную операцию;
- Simulink — открывает окно браузера библиотек Simulink;
- Help (Помощь) — открывает окно справки.

Кнопка New M-file открывает окно редактора/отладчика m-файлов. По умолчанию файлу дается имя Untitled (безымянный), которое впоследствии (при записи файла) можно изменить на другое, отражающее тему задачи.

В левой части окна системы MATLAB 6.0 имеется окно специального браузера рабочей области — Workspace Browser. Оно служит для просмотра ресурсов рабочей области памяти. Браузер дает наглядную визуализацию содержимого рабочей области. Окно браузера рабочей области выполняет и другие важные функции: позволяет просматривать существующие в памяти объекты, редактировать их содержимое и удалять объекты из памяти. Для вывода содержимого объ-

екта достаточно выделить его имя с помощью мыши и щелкнуть на кнопке Open (Открыть). Объект можно открыть и двойным щелчком на его имени в списке. Откроется окно редактирования массива Array Editor. Окно редактирования матрицы дает удобный доступ для редактирования любого элемента матрицы по правилам, принятым при работе с электронными таблицами. Основное из них — быстрый доступ к любому элементу матрицы. Можно также менять тип значений элементов, выбирая его из списка, предоставляемого меню Numeric format (Формат чисел). В окне также выводятся данные о числе строк и столбцов матрицы.

Просмотр рабочей области возможен и в командном режиме, без обращения к браузеру Workspace Browser. Команда who выводит список определенных переменных, а команда whos — список неременных с указанием их размера и объема занимаемой памяти.

Для просмотра файловой структуры MATLAB служит специальный браузер файловой системы (Path Browser), который запускается при обычной загрузке системы. Если был установлен упрощенный интерфейс, то для запуска браузера файловой системы используется окно Current Directory (Текущая папка). Нетрудно заметить, что браузер файловой системы построен по типу общеизвестного Проводника (Explorer). Окно дает возможность просмотра файловой системы ПК и выбора любого файла. Исполнив команду Open (Открыть) из контекстного меню правой клавиши мыши или дважды щелкнув по выделенной строке с именем файла, можно ввести этот файл в окно редактора/отладчика m-файлов. При этом редактор запустится автоматически и его окно с готовым для редактирования выбранным файлом появится на экране.

Основное меню:

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

Подменю File содержит ряд операций и команд для работы с файлами:

- New - открывает подменю с позициями:
- M-file — открытие окна редактора/отладчика m-файлов;
- Figure — открытие пустого окна графики;

- Model — открытие пустого окна для создания Simulink-модели;
 - GUI — открытие окна разработки элементов графического интерфейса пользователя;
 - Open — открывает окно загрузки файла;
 - Close Command Windows — закрывает окно командного режима работы (оно при этом исчезает с экрана);
 - Import data — открывает окно импорта файлов данных;
 - Save Workspace As... — открывает окно записи рабочей области в виде файла с заданным именем;
 - Set Path — открывает окно установки путей доступа файловой системы;
 - Preferences... — открывает окно настройки элементов интерфейса;
 - Print... — открывает окно печати всего текущего документа.
 - Print Selection... — открывает окно печати выделенной части документа;
 - Exit — завершает работу с системой;
- Меню Edit содержит операции и команды редактирования;
- Undo (Отменить) — отмена результата предшествующей операции;
 - Redo (Повторить) — отмена действия последней операции Undo;
 - Cut (Вырезать) — вырезание выделенного фрагмента и перенос его в буфер;
 - Copy (Копировать) — копирование выделенного фрагмента в буфер;
 - Paste (Вставить) — вставка фрагмента из буфера в текущую позицию курсора;
 - Clear (Очистить) — операция очистки выделенной области;
 - Select All (Выделить) — выделение всей сессии;
 - Delete (Стереть) — уничтожение выделенного объекта;
 - Clear Command Windows (Очистить командное окно) — очистка текста сессии (с сохранением созданных объектов);
 - Clear Command History (Очистить окно истории команд) — очистка окна истории; О Clear Workspace — очистка окна браузера рабочей области.

Набор команд меню View существенно расширен и теперь с помощью этого меню можно существенно менять вид пользовательского интерфейса. Меню Window активно только в случае, если в систему

загружены файлы. При этом оно имеет единственную команду Close All (закрыть все окна) и открывающийся список всех загруженных файлов. Он позволяет выбрать окно указанного пользователем файла и сделать его открытым.

Для подготовки, редактирования и отладки m-файлов служит специальный многооконный редактор. 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. Перед запуском файла его необходимо записать на диск, используя команду File > Save as в меню редактора. После записи файла на диск можно заметить, что команда Run в меню Tools (Инструменты) редактора становится активной (до записи файла на диск она пассивна) и позволяет произвести запуск файла. Запустив команду Run, можно наблюдать исполнение m-файла.

Редактор/отладчик позволяет создать m-файл (программу) без той многочисленной «шелухи», которая сопровождает работу в командном режиме. Далее мы убедимся, что текст такого файла подвергается тщательной синтаксической проверке, в ходе которой выявляются и отсеиваются многие ошибки пользователя. Таким образом, редактор обеспечивает синтаксический контроль файла. Редактор имеет и другие важные отладочные средства: он позволяет устанавливать в тексте файла специальные метки, именуемые точками прерывания (breakpoints). 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. Для удобства работы с редактором/отладчиком строки программы в нем нумеруются в последовательном порядке. Редактор является многооконным. Окно каждой программы оформляется как вкладка.

Редактор/отладчик m-файлов выполняет синтаксический контроль программного кода 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.

m-файлы, создаваемые редактором/отладчиком, делятся на два класса:

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

Файл-сценарий (Script-файлом) не имеет списка входных параметров и является примером простой процедуры без параметров. Он использует *глобальные переменные*, т. е. такие 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. Используемые в файле-функции переменные являются *локальными переменными*, изменение значений которых в теле функции никоим образом не влияет на значения, которые те же самые переменные могут иметь за пределами функции.

Назначение кнопок панели инструментов редактора/отладчика следующее:

- New — создание нового m-файла;
- Open — вывод окна загрузки файла;
- Save — запись файла на диск;
- Print — печать содержимого текущего окна редактора;
- Cut — вырезание выделенного фрагмента и перенос его в буфер;
- Copy — копирование выделенного объекта в буфер;
- Paste — размещение фрагмента из буфера в позиции текстового курсора;
- Undo — отмена предшествующей операции;
- Redo — повтор отмененной операции;
- Find text — нахождение указанного текста;
- Show function — показ функции;
- Set/Clear Breakpoint — установка/сброс точки прерывания;
- Clear All Breakpoints — сброс всех точек прерывания;
- Step — выполнение шага трассировки;
- Step In — пошаговая трассировка с заходом в вызываемые m-файлы;
- Step Out — пошаговая трассировка без захода в вызываемые m-файлы;
- Save and Run — запись и сохранение;
- Exit Debug Mode — выход из режима отладки.

Основным приемом отладки *gn*-файлов является установка в их тексте точек прерывания (Breakpoints). Они устанавливаются (и сбрасываются) с помощью кнопки Set/Clear Breakpoint Сброс всех точек прерывания обеспечивается кнопкой Clear All Breakpoints. Для установки точки прерывания необходимо поместить текстовый курсор в нужное место (напротив указанного выражения) и щелкнуть на кнопке Set/Clear Breakpoint или щелкнуть справа от номера строки. Теперь при пуске программы командой Run она будет исполнена до точки прерывания, после чего текущие значения будут выведены в окне MATLAB. С помощью кнопки Step (Шаг) можно выполнить очередной шаг вычислений и т. д. Если отпала необходимость останова в точках прерывания, достаточно кнопкой Clear All Breakpoints удалить разом все точки прерывания. Желтая стрелка указывает, в каком месте программы произошла остановка.

Графическое окно MATLAB

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

Прежде всего в меню Edit окна графики наряду со стандартными операциями работы с буфером есть ряд новых команд:

- Copy Figure (Копировать рисунок) — копирование в буфер рисунка (графика);
- Copy Options (Копировать параметры) — копирование параметров графика;
- Figure Properties (Свойства рисунка) — вывод окна свойств графика;
- Axes Properties (Свойства осей) — вывод окна свойств осей графика;
- Current Object Properties (Свойства текущего объекта) - вывод окна свойств текущего объекта.

Для вывода свойств графиков, их осей и текущих объектов используется окно свойств графиков с соответствующими вкладками.

Отличительной особенностью окна графики в версии MATLAB 6.0 стало появление второй инструментальной панели со средствами форматирования трехмерной (3D) графики. Эта панель (она видна под основной панелью инструментов) выводится командой View > Camera Toolbar. Эта панель управляет некоторой воображаемой фотокамерой (или просто камерой), с помощью которой как бы наблюдается объект. Кнопки имеют наглядные изображения, поясняющие действия кнопок.

В связи с этим их подробное описание лишено смысла — проще опробовать их в действии.

Меню инструментов Tools

Действия кнопок панели инструментов камеры обзора продублированы в меню Tools (Инструменты). Состав команд указанного подменю в версии MATLAB 6.0 существенно изменен и обновлен. Теперь в нем имеются следующие команды:

- Edit Plot (Редактировать график) — редактирование графика;
- Zoom In (Увеличение) — увеличение масштаба графика;
- Zoom Out (Уменьшение) — уменьшение масштаба графика;
- Rotate 3D (Вращение 3D) — вращение в пространстве;
- Move Camera (Передвинуть камеру) — установка камеры обзора;
- Camera Motion (Передвижение камеры) — установка перемещения камеры обзора;
- Camera Axes (Оси камеры) — установка координатных осей при работе с камерой;
- Camera Reset (Установка начального состояния камеры) — сброс установок камеры;
- Basic Fitting — проведение аппроксимации и регрессии;
- Data Statistics — получение статистических данных для точек графика.

Две последние позиции этого меню дают весьма оригинальные возможности обработки точек графика — выполнение регрессии множеством методов с выводом (где это возможно) уравнения регрессии на график и вычисление статистических параметров для этих точек.

Хорошее впечатление оставляет возможность вращения графиков мышью — прием, введенный в целый ряд систем компьютерной математики (Mathcad, Maple 6 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. При этом вращается плоскость, в которой расположен график. Эта плоскость размещается в упомянутом параллелепипеде.

Мы уже рассматривали операции вставки с помощью основной панели инструментов. Эти возможности продублированы в позиции Insert (Вставка) меню графического окна. Это нанесение надписей по осям, титульной надписи, надписи внутри рисунка, стрелки, отрезка прямой, легенды и шкалы цветов.

Для общения с Интернетом служит команда web: О web спецификация — дает связь с Web-сервером. Примеры применения команды web:

- web <http://www.mathworks.com> — загружает Web-страницу MathWorks Web [*Команда support сразу открывает страницу технической поддержки MATLAB.*] в браузер помощи; [*При запуске с параметром —browser можно ввести URL в виде www.mathworks.com, но сайт будет открыт в браузере ОС по умолчанию*]

- web mailto:email_address — использует программу для отправки электронной почты, установленную по умолчанию в настройках операционной системы;

- все формы команды web могут использоваться в функциях. Например, функция `s = web('www.mathworks.com', '-browser')` запускает браузер Интернета операционной системы [*Для UNIX и Linux браузера Интернета можно задавать из MATLAB*] и выдает `s=0`, если браузер запущен, даже если браузер Интернета открывает страницу в автономном режиме (off-line) или не может ее найти, `s=1`, если браузер Интернет не был обнаружен, `s=2`, если браузер был обнаружен, но не был запущен.

РАБОТА С ОКНОМ «СПРАВКА»

Для получения справки по какому-либо конкретному объекту используются команды

» help имя

или

» doc имя,

где имя — имя объекта, для которого требуется вывод справочной информации.

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

lookfor 'Ключевые слова'

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 по примеру на каждый оператор или функцию. Наиболее поучительные примеры можно найти в разделе *demos*, исполнив команду

» `help demos`

MATLAB имеет ряд встроенных фигур, которые можно легко выводить на построение простым указанием их названия. Так, введя команду `knot`, можно задать построение сложной пространственной фигуры узла с функциональной окраской. При запуске этого примера нажатием кнопки `Spin` можно наблюдать изменение окраски, имитирующее как бы движение жидкости внутри замкнутой трубки, образующей данную фигуру. Таким образом, можно сделать вывод о том, что для имитации и моделирования математических и физических задач система MATLAB предоставляет значительные графические возможности: от простейших графиков функций в декартовой системе координат до сложных анимационных графиков с динамической цветной функциональной окраской. Среди множества примеров такой графики всегда можно подобрать наиболее подходящие для решения конкретных задач пользователя, в какой бы области науки, техники или образования он ни работал.

В меню `Help` имеется команда `Demos`, дающая доступ к галерее демонстрационных примеров применения системы MATLAB. При запуске этой команды появляется окно демонстрационных примеров MATLAB `Demos`. Это же окно можно вызвать выполнением команды `demo` в режиме диалога.

В этом окне имеются три панели:

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

Выбрав раздел примеров (щелчком мыши), следует выбрать нужный пример. После этого нажатием кнопки `Run` можно запустить `m`-файл с выбранным примером и наблюдать результат его работы.

Копирование демонстрационных примеров.

Вполне возможно, что вы захотите воспользоваться каким-либо примером для своих целей. Для этого можно использовать `m`-файл примера или перенести его текст в командное окно MATLAB, используя буфер обмена. Текст примера выделяется мышью и используется команда `Copy` (Копировать) меню `Edit` окна примера, в результате чего текст примера попадет в буфер обмена. После этого надо вернуться в командное окно MATLAB и, используя команду `Paste` (Вставить) меню `Edit`, перенести текст примера из буфера в текущую

строку ввода. Выполнив команду (как обычно, клавишей Enter), можно наблюдать исполнение примера. Трудно сказать почему, но создатели MATLAB в последних версиях отказались от возможности копирования выделенного в примерах окна Demos текста программ с помощью контекстного меню правой клавиши мыши. Зато появилась возможность исполнения выделенных в справке (как на заданную функцию в командном режиме, так и в HTML-браузере справки) примеров с помощью команды Evaluate Selection в контекстном меню правой клавиши мыши в тексте справки.

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

ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЙ

Полезно сразу усвоить некоторые команды управления окном командного режима:

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

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. Приемы работы:

- для указания ввода исходных данных используется символ »;
- данные вводятся с помощью простейшего строчного редактора;
- для блокировки вывода результата вычислений некоторого выражения после него надо установить знак ; (точка с запятой);
- если не указана переменная для значения результата вычислений, то MATLAB назначает такую переменную с именем `ans`;
- знаком присваивания является привычный математикам знак равенства;
- встроенные функции (например, `sin`) записываются строчными буквами и их аргументы указываются в *круглых скобках*.

```

Пример:
» 2+3
ans=
5
» sin(1)
ans=
0.8415
Примеры простых математических выражений:
2+3      2.301*sin(x)      4+exp(3)/5
sqrt(y)/2      sin(pi/2)
Примеры представления чисел:
0      2      -3
2.301  0.00001  123.456e-24      -234.456e10

```

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*. Основные системные переменные, применяемые в системе MATLAB, указаны ниже:

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

Вот примеры применения системных переменных:

```

» 2*pi
ans =
6.2832
» eps
ans =
2.2204e-016
» realmin

```

```
ans=
2.2251e-308
» realmax
ans=
1.7977e+308
» 1/0
Warning: Divide by zero, -предупреждение:деление на
ноль
ans=
Inf
» 0/0
Warning: Divide by zero, -предупреждение:деление на
ноль
ans =
NaN
```

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

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

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. Переменным можно задавать значения:

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

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

Оператор — это специальное обозначение для определенной операции над данными — *операндами*. Например, простейшими арифметическими операторами являются знаки суммы `+`, вычитания `-`, умножения `*`, деления `/`, возведение в степень `^`. Следует отметить, что большинство операторов относится к матричным операциям. Например, операторы умножения `*` и деления `/` вычисляют произведение и частное от деления двух многомерных массивов,

векторов или матриц. Есть ряд специальных операторов, например, оператор `\` означает деление *справа налево*, а операторы `.` `*` и `./` означают соответственно *поэлементное* умножение и *поэлементное* деление массивов. Полный список операторов можно получить, используя команду `» help ops`.

Следующие примеры поясняют сказанное на примере операций с векторами:

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

Применение оператора: (двоеточие)

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

Начальное_значение:Шаг:Конечное_значение

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

```
» 1:5
ans =
12345
» 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
```

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

Так, ввод

```
» M=[1 2 3: 4 5 6; 7 8 9];
задает квадратную матрицу, которую можно вывести:
» M
M =
1 2 3
4 5 6
7 8 9
```

СОХРАНЕНИЕ РАБОЧЕЙ ОБЛАСТИ СЕССИИ

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

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

Завершение работы с системой

Для завершения работы с системой можно использовать команды `exit`, `quit` (которые сохраняют содержимое рабочей области и выполняет другие действия в соответствии с файлом сценария `finish.m`) или комбинацию клавиш `Ctrl+Q`. Если необходимо сохранить значения всех переменных (векторов, матриц) системы, то перед вводом команды `exit` 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 С ЭЛЕМЕНТАРНЫМИ ФУНКЦИЯМИ

Арифметические операторы являются самыми распространенными и известными. В отличие от большинства языков программирования в системе MATLAB практически все операторы являются *матричными*, т. е. предназначены для выполнения операций над матрицами.

Если один из операндов — скаляр, происходит сравнение всех элементов второго операнда-массива со значением этого скаляра:

```
M=
-1 0
 1 2
» M>=0
ans =
0 1
1 1
```

В общем случае операторы отношения сравнивают два массива одного размера и выдают результат в виде массива того же размера:

```
» M>[0 1;.1 0]
ans =
0 0
0 1
```

Логические операторы и соответствующие им функции служат для реализации поэлементных логических операций над элементами одинаковых по размеру массивов.

В системе MATLAB определены следующие **математические** (алгебраические и арифметические) функции:

- `abs(X)` — возвращает абсолютную величину для каждого числового элемента вектора X . Если X содержит комплексные числа, `abs(X)` вычисляет модуль каждого числа;
- `exp(X)` — возвращает экспоненту для каждого элемента X . Для комплексного числа $z = x + i*y$ функция `exp(z)` вычисляет комплексную экспоненту $\exp(z)=\exp(x)*(\cos(y)+i*\sin(y))$;
- `factor(n)` — возвращает вектор-строку, содержащую простые множители числа n . Для массивов эта функция неприменима;
- `G=gcd(A, B)` — возвращает массив, содержащий наибольшие общие делители соответствующих элементов массивов целых чисел A и B . Функция `gcd(0,0)` возвращает значение 0, в остальных случаях возвращаемый массив G содержит положительные целые числа;

- $[G, C, D] = \text{gcd}(A, B)$ — возвращает массив наибольших общих делителей G и массивов C и D , которые удовлетворяют уравнению $A(i) \cdot C(i) + B(i) \cdot D(i) = G(i)$. Они полезны для выполнения элементарных эрмитовых преобразований;

- $\text{lcm}(A, B)$ — возвращает наименьшие общие кратные для соответствующих парных элементов массивов A и B . Массивы A и B должны содержать положительные целые числа и иметь одинаковую размерность (любой из них может быть скаляром). Пример:

```
» A=[1 354];
```

```
» B=[2 462];
```

```
» lcm(A, B)
```

```
ans =
```

```
2 12 30 4
```

- $\log(X)$ — возвращает натуральный логарифм элементов массива X ;

- $\log_2(X)$ — возвращает логарифм по основанию 2 элементов массива X ;

- $\log_{10}(X)$ — возвращает логарифм по основанию 10 для каждого элемента X . Область функции включает комплексные числа, что способно привести к непредвиденным результатам при некорректном использовании;

- $\text{mod}(X, Y)$ — взятие x по модулю y , возвращает остаток от деления X на Y (т. е., $X - Y \cdot \text{floor}(X./Y)$) для ненулевого Y , и X в противном случае. Если операнды X и Y имеют одинаковый знак, функция $\text{mod}(X, Y)$ возвращает тот же результат, что $\text{mod}(X, Y)$. Однако (для положительных X и Y) $\text{mod}(-x, y) = \text{rem}(-x, y) + y$.

Примеры:

```
» M = mod(5, 2)
```

```
M =
```

```
1
```

```
» mod(10, 4)
```

```
ans = 2
```

$\text{pow}_2(Y)$ — возвращает массив X , где каждый элемент есть Z^Y ;

- $\text{pow}_2(F, E)$ — вычисляет $X \cdot 2^E$ для соответствующих элементов F и E . Аргументы F и E — массивы действительных и целых чисел соответственно.

- $p = \text{nextpow}_2(A)$ — возвращает такой показатель степени p , что $2^p \geq \text{abs}(A)$. Эта функция эффективно применяется для выполнения быстрого преобразования Фурье. Если A не является скалярной величиной, то nextpow_2 возвращает значение $\text{nextpow}_2(\text{length}(A))$.

- Функция `primes(n)` возвращает вектор-строку простых чисел, меньших или равных n .
- `sqrt(A)` — возвращает квадратный корень каждого элемента массива X . Для отрицательных и комплексных элементов X функция `sqrt(X)` вычисляет комплексный результат.

Пример:

```
» A=[25 21.23 55.8 3]:  
» sqrt(A)  
ans =  
5.0 4.6076 7.4699 1.7321
```

Тригонометрические и обратные им функции

В системе MATLAB определены следующие тригонометрические и обратные тригонометрические функции. Функции вычисляются для каждого элемента массива. Входной массив допускает комплексные значения. Напоминаем, что все углы в функциях задаются в радианах.

- `acos(X)` — возвращает арккосинус для каждого элемента X . Для действительных значений X в области $[-1, 1]$ `acos(X)` возвращает действительное значение из диапазона $[0, \pi]$, для действительных значений X вне области $[-1, 1]$ `acos(X)` возвращает комплексное число.

Примеры:

```
» Y = acos(0.5)  
1.0472  
» acos([0.5 1 2])  
ans =  
1.0472 0 0 + 1.31701i
```

- `atan(X)` — возвращает арктангенс для каждого элемента X . Для действительных значений X `atan(X)` находится в области $[-\pi/2, \pi/2]$.

- `cos(X)` — возвращает косинус для каждого элемента X .
- `cot(X)` — возвращает котангенс для каждого элемента X .
- `csc(X)` — возвращает косеканс для каждого элемента X .
- `sec(X)` — возвращает массив той же размерности что и X , состоящий из секансов элементов X .

- `sin(X)` — возвращает синус для каждого элемента X .
- `tan(X)` — возвращает тангенс для каждого элемента X .

Пример:

```
» X=[0.08 0.06 1.09]
```

```
X=  
0.0800 0.0600 1.0900  
» tan(X)  
ans=  
0.802 0.0601 1.9171
```

Ряд особых функций служит для выполнения операций **округления числовых данных** и анализа их знака.

- `fix(A)` — возвращает массив `A` с элементами, округленными до ближайшего к нулю целого числа. Для комплексного `A` действительные и мнимые части округляются отдельно.

Примеры:

```
» A=[1/3 2/3; 4.99 5.01]  
A =  
0.3333 0.6667  
4.9900 5.0100  
» fix(A)  
ans =  
0 0  
4 5
```

- `floor(A)` — возвращает `A` с элементами, представляющими ближайшее меньшее или равное соответствующему элементу `A` целое число. Для комплексного `A` действительные и мнимые части преобразуются отдельно.

- `ceil(A)` — возвращает ближайшее большее или равное `A` целое число. Для комплексного `A` действительные и мнимые части округляются отдельно;

- `rem(X,Y)` — возвращает $X - \text{fix}(X/Y) \cdot Y$, где `fix(X/Y)` — целая часть от частного `X/Y`.

Если операнды `X` и `Y` имеют одинаковый знак, функция `rem(X, Y)` возвращает тот же результат, что `mod(X,Y)`. Однако (для положительных `X` и `Y`) `rem(-x,y) = mod(-x,y)-y`. Функция `rem` возвращает результат, находящийся между 0 и `sign(X)*abs(Y)`. Если `Y=0`, функция `rem` возвращает NaN. Аргументы `X` и `Y` должны быть целыми числами. Из-за неточного представления в компьютере чисел с плавающей запятой использование вещественных (или комплексных) входных аргументов может привести к непредвиденным результатам.

- `round(X)` — возвращает округленные до ближайшего целого элементы массива `X`. Для комплексного `X` действительные и мнимые части округляются отдельно;

- `sign(X)` — возвращает массив `Y` той же размерности, что и `X`, где каждый из элементов `Y` равен

- 1, если соответствующий элемент X больше 0;
- 0, если соответствующий элемент X равен 0;
- -1, если соответствующий элемент X меньше 0. Для ненулевых действительных и комплексных X — $\text{Sign}(X)=X./\text{abs}(X)$.

Функции обработки множеств

Множество — первичное понятие математики, не имеющее четкого определения. Под множеством подразумевается совокупность некоторых объектов, например книг в библиотеке, людей в зале или элементов вектора. В этом разделе приводятся некоторые функции для обработки множеств, представленных векторами. Они широко используются при анализе и обработке данных.

- `intersect(a,b)` — возвращает пересечение множеств для двух векторов a и b , т. е. общие элементы векторов a и b . Результирующий вектор отсортирован по возрастанию. Если входные массивы не являются векторами, то они рассматриваются как совокупность векторов-столбцов $a=a(:)$ или $b=b(:)$;
- `intersect(a,b, 'rows')` — возвращает строки, общие для a и b , когда a и b представляют собой матрицы с одинаковым числом столбцов;
- `[c,ia,ib] = intersect(a,b)` — также возвращает вектор-столбец индексов ia и ib , но так, что $c = a(ia)$ и $c = b(ib)$ (или $c = a(ia,:)$ и $c = b(ib,:)$);
- `ismember(a,S)` — возвращает вектор той же длины, что и исходный a , содержащий логические единицы на месте тех элементов вектора a , которые принадлежат множеству S , и логические нули на месте тех элементов вектора a , которые не принадлежат множеству S ;
- `ismember (A, S,' rows')` — возвращает вектор, содержащий логические единицы там, где строки матрицы A являются также строками матрицы S , и логические нули в остальных позициях. A и S должны быть матрицами с одним числом столбцов.
- `setdiff(a,b)` — возвращает разность множеств, т. е. те элементы вектора a , которые не содержатся в векторе b . Результирующий вектор сортируется по возрастанию;
- `setdiff(a,b, 'rows')` — возвращает те строки из матрицы a , которые не содержатся в матрице b . Матрицы a и b должны иметь одинаковое число столбцов;
- `[c,i] = setdiff(...)` — возвращает также вектор индексов i , такой что $c = a(i)$ или $c = a(i,:)$.

Если входной массив a является матрицей, то он расценивается как вектор-столбец $a(:)$.

- $\text{setxor}(a,b)$ — исключающее ИЛИ для векторов a и b . Результирующий вектор отсортирован;

- $\text{setxor}(a,b, \text{'rows'})$ — возвращает строки, которые не являются пересечениями матриц a и b . Матрицы a и b должны иметь одинаковое число столбцов;

- $[c,ia,ib] = \text{setxor}(\dots)$ — возвращает также векторы индексов ia и ib так, что c является отсортированной комбинацией элементов $c = a(ia) \text{ и } c = b(ib)$ или для комбинаций строк $c = a(ia,:) \text{ и } c = b(ib,:)$.

Если массив a является матрицей, то он расценивается как вектор-столбец $a(:)$;

- $\text{union}(a,b)$ — возвращает вектор объединенных значений из a и b без повторяющихся элементов. Результирующий вектор сортируется в порядке возрастания;

- $\text{union}(a,b, \text{'rows'})$ — возвращает объединенные строки из a и b , не содержащие повторений (a и b — это матрицы с одинаковым числом столбцов);

- $[c,ia,ib] = \text{union}(\dots)$ — дополнительно возвращает ia и ib — векторы индексов, такие, что $c = a(ia)$ и $c = b(ib)$, или, для объединенных строк, $c = a(ia,:) \text{ и } c = b(ib,:)$. Невекторный массив a расценивается как вектор-столбец $a(:)$;

- $\text{unique}(a)$ — возвращает значения элементов из a , не содержащие повторений. Результирующий вектор сортируется в порядке возрастания. Невекторный массив расценивается как вектор-столбец $a=a(:)$;

- $\text{unique}(a, \text{'rows'})$ — возвращает уникальные строки a ;

- $[b,i,j] = \text{unique}(\dots)$ — дополнительно возвращает i и j — векторы индексов, такие что $b = a(i)$ (или $b = a(1,:) \text{ и } b = a(j,:)$).

Функции над строками

К функциям над строками обычно относят поиск вхождений одних строк в другие, замену регистров символов, объединение строк и т. д. Следующие функции осуществляют операции над строками:

- $\text{findstr}(str1, str2)$ — обеспечивает поиск начальных индексов более короткой строки внутри более длинной и возвращает вектор этих индексов. Индексы указывают положение первого символа более короткой строки в более длинной строке.

Пример:

```

» strb'Example of the function Is the findstr
function';
» str2='the';
» k = findstr(str1,str2)
k =
12 28

```

- `lower('str')` — возвращает строку символов `str`, в которой символы верхнего регистра переводятся в нижний регистр, а все остальные символы остаются без изменений.

Пример:

```

» str='Example Of The Function';
» t=lower(str)
t =
example of the function

```

- `upper('str')` — возвращает строку символов `str`, в которой все символы нижнего регистра переводятся в верхний регистр, а все остальные символы остаются без изменений.

Пример:

```

» str='danger!';
» t = upper(str)
t =
DANGER!

```

- `strcat(s1,s2,s3,...)` — выполняет горизонтальное объединение соответствующих рядов массивов символов `s1`, `s2`, `s3` и т. д., причем пробелы в конце каждого ряда отбрасываются, и возвращает объединенную строку (ряд) результирующего массива символов, пробелы добавляются заново после анализа строк в полученном массиве. Все входные массивы должны иметь одинаковое число строк (в частном случае должны быть представлены в виде одной строки символов), но если один из входных аргументов — не массив символов, а строковый массив ячеек, то любой из других входных аргументов может быть скаляром или любым массивом той же размерности и того же размера. Если входной массив состоит только из символов, то выходной массив также будет являться массивом символов. Если любой из входных массивов является строковым массивом ячеек, то функция `strcat` возвращает строковый массив ячеек, сформированный из объединенных соответствующих элементов массивов `s1`, `s2`, `s3`, при этом любой из элементов может быть скаляром и т. д.

Примеры:

```

» s1{1.1}=' Home '

```

```

» s1{1,2}='book':
» s1 s1 =
'Home' 'book'
» s2{ 1,1}= 'home':
» s2{ 1,2}= 'reading';
» s2 s2 =
'home' 'reading'
» t = strcat(s1.s2)
t =
'Homehome' 'book reading'

```

• `strvcat(t1,t2,t3,...)` — выполняет вертикальное объединение строк `t1`, `t2`, `t3`,... в массив символов `S` аналогично `char(t1,t2,t3,...)`. Пример:

```

» t1=['string'];
» t2=['concatenation']:
» S = strvcat(t1,t2)
S =
string concatenation

```

• `strcmp('str1','str2')` — возвращает логическую единицу, если две сравниваемые строки `str1` и `str2` идентичны, и логический ноль в противном случае;

• `TF = strcmp(S,T)` — возвращает строковый массив ячеек `TF`, содержащий единицы для идентичных элементов массивов `S` и `T` и нули для всех остальных, причем если один из массивов — не массив символов, а строковый массив ячеек, то перед сравнением из сравниваемых копий рядов массива символов удаляются пробелы в конце строк. Массивы `S` и `T` должны иметь одинаковый размер, или один из них может быть скалярной ячейкой.

Примеры:

```

» str1='computer';
» str2='computer';
» k = strcmp(str1,str2)
k =
1
» S{1,1}='first';
» S{1,2}='second':
» S S =
'first' 'second'
» T{1,1}='third';
» TF = strcmp(S,T)

```

```

TF =
0 0
» T{1.1}='second';
» TF = strcmp(S,T)
TF =
0 1

```

• `strncmp( 'str1', 'str2', n)` — возвращает логическую единицу, если две сравниваемые строки `str1` и `str2` содержат `n` первых идентичных символов, и логический ноль в противном случае. Аргументы `str1` и `str2` могут быть также строковыми массивами ячеек.

• `TF = strncmp(S,T,n)` — возвращает строковый массив ячеек `TF`, содержащий единицы для идентичных (до `n` символов) элементов массивов `S` и `T` и нули для всех остальных.

Примеры:

```

» str1='computer'
str1 =
computer
» str1='computer for me'
str1 =
computer for me
» k = strncmp(str1,str2,3)
k =
1
» k = strncmp(str1,str2,12)
L =
0

```

• `strmatch( 'str', STRS, 'exact')` — возвращает только индексы строк символов массива `STRS`, точно совпадающих со строкой символов `str`;

• `strjust(S)` — возвращает выровненный вправо массив символов (т. е. перемещает пробелы в конце рядов массива символов, если они есть, в начало тех же рядов) [*Функция `strjust(S, 'left')` возвращает массив символов, где все строки выровнены влево, а `strjust(S, 'center')` — где все строки выровнены по центру.*— Примеч.ред.];

• `strmatch( 'str', STRS)` — просматривает массив символов или строковый массив ячеек `STRS` по строкам, находит строки символов, начинающиеся со строки `str`, и возвращает соответствующие индексы строк;

Пример:

```
» STRS{1.1}='character':
» STRS{1.2}='array';
» STRS{2.1}='character array':
» STRS{2.2}='string':
» STRS
STRS =
'character' 'array'
'character array' 'string'
» i = strmatchCcharac'.STRS)
i =
1
2
»i = strmatch('character',STRS,'exact')
i =
1
```

- `strrep(str1,str2,str3)` — заменяет все подстроки `str2`, найденные внутри строки символов `str1` на строку `str3`;

- `strrep(str1,str2,str3)` — возвращает строковый массив ячеек, полученный в результате выполнения функции `strrep` над соответствующими рядами входных массивов символов или ячеек, если один из аргументов `str1`, `str2` или `str3` — строковый массив ячеек. В этом случае любой из аргументов может быть также скалярной ячейкой.

Пример:

```
» str1='This is a good example for me.':
» str2='good';
» str3='best';
» str = strrep(str1.str2.str3)
str =
This is a best example for me.
```

- `strtokCstr'.delimiter)` — возвращает часть текстовой строки `str`, ограниченную с ее конца разделителем `del i miter`. Символы-разделители в начале строки игнорируются. Вектор `delimiter` содержит возможные символы-разделители;

- `strtokCstr')` — использует символ-разделитель по умолчанию («белое пространство»). Реальными символами-разделителями при этом являются символ табуляции (ASCII-код 9), символ возврата каретки (ASCII-код 13) и пробел (ASCII-код 32);

- `[token,rem]=strtok(...)` — возвращает остаток `rem` исходной строки.

Примеры:

```
» str='This is a good example for me.':
» token = strtok(str)
token =
This
» token = strtok(str,'f')
token =
This is a good example
» [token,rem] = strtok(str)
token =
This
rem =
is a good example for me.
```

РАБОТА С ВЕКТОРОМ, МАТРИЦЕЙ И МНОГОМЕРНЫМ МАССИВОМ

Особенности задания векторов и матриц

MATLAB — система, специально предназначенная для проведения сложных вычислений с векторами, матрицами и массивами. При этом она по умолчанию предполагает, что каждая заданная переменная — это вектор, матрица или массив. Все определяется конкретным значением переменной. Например, если задано $X=1$, то это значит, что X — это вектор с единственным элементом, имеющим значение 1. Если надо задать вектор из трех элементов, то их значения следует перечислить в квадратных скобках, разделяя пробелами. Так, например, присваивание

```
» V=[1 2 3]
```

V=

123

задает вектор V, имеющий три элемента со значениями 1, 2 и 3. После ввода вектора система выводит его на экран дисплея. Задание матрицы требует указания нескольких строк. Для разграничения строк используется знак ; (точка с запятой). Этот же знак в конце ввода предотвращает вывод матрицы или вектора (и вообще результата любой операции) на экран дисплея. Так, ввод

```
» M=[1 2 3: 4 5 6; 7 8 9];
```

задает квадратную матрицу, которую можно вывести:

```
» M
```

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

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

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

Для указания отдельного элемента вектора или матрицы используются выражения вида V(1) или M(i,j). Например, если задать >>M(2,2)

```
ans =
5
```

то результат будет равен 5. Если нужно присвоить элементу M(i, j) (в тексте программ MATLAB лучше не использовать i и j как индексы, так как i и j — обозначение квадратного корня из -1, но можно использовать I и 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=
123
456
7 8 9
>>M(2)
ans=
4
>> M(8)
ans =
6
>> M(9)
```

```

ans =
9
» M(5)=100;
    » M
    M=
123
41006
789

```

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

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

```

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

```

Уже сама по себе возможность создания такой матрицы с помощью простой функции `magic` заинтересует любителей математики. Но векторных и матричных функций в системе множество, и мы их де-

тально рассмотрим в дальнейшем. Напомним, что для стирания переменных из рабочей области памяти служит команда `clear`.

Объединение малых матриц в большую

Описанный способ задания матриц позволяет выполнить операцию *конкатенации* — объединения малых матриц в большую. Например, создадим вначале магическую матрицу размера 3x3:

» `A=magic(3)`

A=

8	1	6
3	5	7
4	9	2

Теперь можно построить матрицу, содержащую четыре матрицы:

» `B=[A A+16:A+32 A+16]`

B =

8	1	6	24	17	22
3	5	7	19	21	23
4	9	2	20	25	18
40	33	38	24	17	22
35	37	39	19	21	23
36	41	34	20	25	18

Полученная матрица имеет уже размер 6x6. Вычислим сумму ее столбцов:

» `sum(B)`

ans=

126 126 126 126 126 126

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

» `sum(B, '')`

ans =

78 78 78 174 174 174

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

» `D=magic(6)`

D=

35	1	6	26	19	24
3	32	7	21	23	25
31	9	2	22	27	20
8	28	33	17	10	15
30	5	34	12	14	16
4	36	29	13	18	11

```

» sum(D)
ans=
111 111 111 111 111 111
» sum(D. ')
ans=
111 111 111 111 111 111

```

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

Удаление столбцов и строк матриц

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

```
» M=[1 2 3; 4 5 6; 7 8 9]
```

```
M =
```

```

1      2      3
4      5      6
7      8      9

```

Удалим второй столбец используя оператор : (двоеточие):

```
» M(:,2)=[ ]
```

```

1 3
4 6
7 9

```

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

```
» M(2,:)= [ ]
```

```
M =
```

```

1 3
7 9

```

ОПЕРАЦИИ С ВЕКТОРАМИ И МАТРИЦАМИ

Матрицы представляют собой самые распространенные объекты системы MATLAB. По обилию матричных операторов и функций MATLAB является лидером среди массовых систем компьютерной математики.

Создание матриц с заданными свойствами

Создание единичной матрицы

Для создания единичной матрицы (она обычно обозначается как E) служит функция `eye`:

- `eye(n)` — возвращает единичную матрицу размера $n \times n$;
- `eye(m,n)` или `eye([m n])` — возвращают матрицу размера $m \times n$ с единицами по диагонали и нулями в остальных ячейках;
- `eye(size(A))` — возвращает единичную матрицу того же размера, что и A .

Единичная матрица не определена для многомерных массивов. Так, функция `y = eye([2,3,4])` при попытке ее вычисления приведет к ошибке.

Пример использования функции `eye`:

```
> t=eye(4,5)
```

```
t =
```

```
1     0     0     0     0
0     1     0     0     0
0     0     1     0     0
0     0     0     1     0
```

Для создания матриц, все элементы которых — единицы, используется функция `ones`:

- `ones(n)` — возвращает матрицу размера $n \times n$, все элементы которой — единицы. Если n — не скаляр, то появится сообщение об ошибке;
- `ones(m,n)` или `ones([m n])` — возвращают матрицу размера $m \times n$, состоящую из единиц;
- `ones(d1,d2,d3,...)` или `ones([d1 d2 d3...])` — возвращает массив из единиц с размером $d1 \times d2 \times d3 \times \dots$;
- `ones(size(A))` — возвращает массив единиц той же размерности и размера, что и A . Матрица с единичными элементами в

отличие от единичной матрицы в MATLAB определена и для многомерных массивов.

Иногда нужны матрицы, все элементы которых — нули. Следующая функция обеспечивает создание таких матриц:

- `zeros(n)` — возвращает матрицу размера $n \times n$, содержащую нули. Если n — не скаляр, то появится сообщение об ошибке;
- `zeros(m,n)` или `zeros([m n])` — возвращают матрицу размера $m \times n$, состоящую из нулей;
- `zeros(d1,d2,d3,...)` или `zeros([d1.d2.d3...])` — возвращают массив из нулей размера $d1 \times d2 \times d3 \times \dots$;
- `zeros(size(A))` — возвращает массив нулей того же размера и размерности, что и A .

Функция `linspace` формирует линейный массив равноотстоящих узлов. Это подобно оператору `:`, но дает прямой контроль над *числом* точек. Применяется в следующих формах:

- `linspace(a,b)` — возвращает линейный массив из 100 точек, равномерно распределенных между a и b ;
- `linspace(a,b,n)` — генерирует n точек, равномерно распределенных в интервале от a до b .

Функция `logspace` генерирует вектор равноотстоящих в логарифмическом масштабе точек. Она особенно эффективна при создании вектора частот. Это логарифмический эквивалент оператора `:` и функции `linspace`:

- `logspace(a,b)` — возвращает вектор-строку из 50 равноотстоящих в логарифмическом масштабе точек между декадами 10^0 и 10^b ;
- `logspace(a,b,n)` — возвращает n точек между декадами 10^a и 10^b ;
- `logspace(a,pi)` — возвращает точки в интервале между 10^a и π . Эта функция очень полезна в цифровой обработке сигналов.

Все аргументы функции `logspace` должны быть скалярными величинами.

Создание массивов со случайными элементами

- `r = randperm(n)` — возвращает случайные перестановки целых чисел $1:n$ в векторе-строке. Пример:

```
» randperm(6)
ans =
243651
```

Функция `rand` генерирует массивы случайных чисел, значения элементов которых *равномерно* распределены в промежутке $(0, 1)$:

- `rand(n)` — возвращает матрицу размера $n \times n$. Если n — не скаляр, то появится сообщение об ошибке;
- `rand(m,n)` или `rand([m n])` — возвращают матрицу размера $m \times n$;
- `rand(m,n,p,...)` или `rand([m n p...])` — возвращает многомерный массив;
- `rand(size(A))` — возвращает массив того же размера и размерности, что и A , с элементами, распределенными по равномерному закону;
- `rand` (без аргументов) — возвращает одно случайное число, которое изменяется при каждом последующем вызове и имеет равномерный закон распределения;
- `rand('state')` — возвращает вектор с 35 элементами, содержащий текущее состояние генератора случайных чисел с равномерным распределением. Для изменения состояния генератора можно применять следующие формы этой функции:
 - `rand('state'.s)` — устанавливает состояние в s ;
 - `rand('state',0)` — сбрасывает генератор в начальное состояние;
 - `rand('state'.j)` — для целых j , устанавливает генератор в j -е состояние;
 - `rand('state',sum(100*clock))` — каждый раз сбрасывает генератор в состояние, зависящее от времени.

Функция `randn` генерирует массив со случайными элементами, распределенными по нормальному закону с нулевым математическим ожиданием и среднеквадратическим отклонением, равным 1:

- `randn(n)` — возвращает матрицу размера $n \times n$. Если n — не скаляр, то появится сообщение об ошибке;
- `randn(m,n)` или `randn([m n])` — возвращают матрицу размера $m \times n$;
- `randn(m,n,p,...)` или `randn([m n p...])` — возвращает массив с элементами, значения которых распределены по нормальному закону;
- `randn(size(A))` — возвращает массив того же размера, что и A , с элементами, распределенными по нормальному закону;
- `randn` (без аргументов) — возвращает одно случайное число, которое изменяется при каждом последующем вызове и имеет нормальное распределение;
- `randn('state')` — возвращает двухэлементный вектор, включающий текущее состояние нормального генератора. Для

изменения состояния генератора можно применять следующие формы этой функции:

- `randn('state',s)` — устанавливает состояние в s ;
- `randn('state', 0)` — сбрасывает генератор в начальное состояние;
- `randn('state', j)` — для целых j устанавливает генератор в J -е состояние;
- `randn('state', sum( 100*clock))` — каждый раз сбрасывает генератор в состояние, зависящее от времени.

Конкатенацией называют объединение массивов, которое реализует следующая функция.

- `C = cat(dim, A, B)` — объединяет массивы A и B в соответствии со спецификацией размерности dim и возвращает объединенный массив; $dim = 1$ — горизонтальная конкатенация, $dim = 2$ — вертикальная, $dim = 3$ — многомерный массив размерности 3 и т. д.;
- `C = cat(dim,A1,A2,A3,A4,...)` объединяет все входные массивы ($A1, A2, A3, A4$ и т. д.) в соответствии со спецификацией размерности dim и возвращает объединенный массив;
- `cat(2,A,B)` — это то же самое, что и `[A,B]`, `acat(A,B)` — то же самое, что и `[A; B]`. При записи `cat(dim, C (:))` или `cat(dim, C. field)` эта функция применима к массивам ячеек или структур, содержащим численные матрицы.

Свойства матриц сильно зависят от их диагональных элементов. Следующая функция MATLAB позволяет создавать специальные типы матриц с заданными диагональными элементами:

- `X = diag(v,k)` — для вектора v , состоящего из n компонентов, возвращает квадратную матрицу X порядка $n+abs(k)$ с элементами v на k -й диагонали, при $k=0$ — это главная диагональ (из левого верхнего угла матрицы в правый нижний угол), при $k>0$ — одна из диагоналей (диагональ в терминологии MATLAB — это линия, параллельная главной диагонали) выше главной диагонали, при $k<0$ — одна из нижних диагоналей. Остальные элементы матрицы — нули;
- `X = diag(v)` — помещает вектор v на главную диагональ (то же, что и в предыдущем случае при $k=0$);
- `v = diag(X,k)` — для матрицы X возвращает вектор-столбец, состоящий из элементов n -й диагонали матрицы X ;
- `v = diag(X)` — возвращает главную диагональ матрицы X (то же, что и в предыдущем случае при $k=0$);

Для перестановок элементов матриц служат следующие функции:

- $B = \text{flipr}(A)$ — зеркально переставляет столбцы матрицы A относительно вертикальной оси;
- $B = \text{flipud}(A)$ — зеркально переставляет строки матрицы A относительно горизонтальной оси;
- $\text{perms}(v)$ — возвращает матрицу P , которая содержит все возможные перестановки элементов вектора v . Каждая перестановка в отдельной строке. Матрица P содержит $n!$ строк и n столбцов.

Несколько простых функций служат для перемножения элементов массивов;

- $\text{prod}(A)$ — возвращает произведение элементов массива, если A — вектор, или вектор-строку, содержащую произведения элементов каждого столбца, если A — матрица;
- $\text{prod}(A, \text{dim})$ — возвращает матрицу (массив размерности два) с произведением элементов массива A по столбцам ($\text{dim}=1$), по строкам ($\text{dim}=2$), по иным размерностям в зависимости от значения скаляра dim ;
- $\text{cumprod}(A)$ — возвращает произведение с накоплением. Если A — вектор, $\text{cumprod}(A)$ возвращает вектор, содержащий произведения с накоплением элементов вектора A . Если A — матрица, $\text{cumprod}(A)$ возвращает матрицу того же размера, что и A , содержащую произведения с накоплением для каждого столбца матрицы A (Первая строка без изменений, во второй строке произведение первых двух элементов каждого столбца, в третьей строке элементы второй строки матрицы-результата умножаются на элементы третьей строки матрицы входного аргумента по столбцам и т. д.);
- $\text{cumprod}(A, \text{dim})$ — возвращает произведение с накоплением элементов по строкам или столбцам матрицы в зависимости от значения скаляра dim ;
- $\text{cross}(U, V)$ — возвращает векторное произведение векторов U и V в трехмерном пространстве, т. е. $W=U \times V$. U и V — обязательно векторы с тремя элементами;
- $\text{cross}(U, V, \text{dim})$ — возвращает векторное произведение U и V по размерности, определенной скаляром dim . U и V — многомерные массивы, которые должны иметь одну и ту же размерность, причем размер векторов в каждой размерности $\text{size}(U, \text{dim})$ и $\text{size}(V, \text{dim})$ должен быть равен 3.

Определены следующие функции суммирования элементов массивов:

- `sum(A)` — возвращает сумму элементов массива, если `A` — вектор, или вектор-строку, содержащую сумму элементов каждого столбца, если `A` — матрица;
- `sum(A,dim)` — возвращает сумму элементов массива по столбцам (`dim=1`), строкам (`dim=2`) или иным размерностям в зависимости от значения скаляра `dim`.

Пример:

```
» A=magic(4)
```

```
A =
```

```
16     2         3         13
 5    11        10         8
 9     7         6        12
 4    14        15         1
```

```
» B=sum(A)
```

```
B =
```

```
34 34 34 34
```

• `cumsum(A)` — выполняет суммирование с накоплением. Если `A` — вектор, `cumsum(A)` возвращает вектор, содержащий результаты суммирования с накоплением элементов вектора `A`. Если `A` — матрица, `cumsum(A)` возвращает матрицу того же размера, что и `A`, содержащую суммирование с накоплением для каждого столбца матрицы `A`;

• `cumsum(A,dim)` — выполняет суммирование с накоплением элементов по размерности, определенной скаляром `dim`. Например, `cumsum(A,1)` выполняет суммирование по столбцам.

Для создания матриц, состоящих из других матриц, используются следующие функции:

• `repmat(A, m, n)` — возвращает матрицу `B`, состоящую из `m`х`n` копий матрицы `A` (т. е. в матрице `m`х`n` каждый элемент заменяется на копию матрицы `A`);

• `repmat(A,n)` — формирует матрицу, состоящую из `n`х`n` копий матрицы `A`;

• `repmat(A,[m n])` — дает тот же результат, что и `repmat(A,m,n)`;

• `repmat(A,[m n p...])` — возвращает многомерный массив (`m`х`n`х`p`х...), состоящий из копий многомерного массива или матрицы `A`;

• `repmat(A, m, n)` — когда `A` — скаляр, возвращает матрицу размера `m`х`n` со значениями элементов, заданных `A`. Это делается намного быстрее, чем `A*ones(m,n)`;

• `reshape(A,m,n)` — возвращает матрицу `B` размерностью `m`х`n`, сформированную из `A` путем последовательной выборки по столбцам.

Если число элементов A не равно $m \times n$, то выдается сообщение об ошибке;

- `reshape(A,m,n,p,...)` или `B = reshape(A,[m n p...])` — возвращает N -мерный массив с элементами из A , но имеющий размер $m \times n \times p \times \dots$. Произведение $m \times n \times p \times \dots$ должно быть равно значению `prod(size(A))`.

- `reshape(A, slz)` — возвращает N -мерный массив с элементами из A , но перестроенный к размеру, заданному с помощью вектора `slz`.

Следующая функция обеспечивает поворот матрицы (по расположению элементов): `rot90(A)` — осуществляет поворот матрицы A на 90° против часовой стрелки;

- `rot90(A,k)` — осуществляет поворот матрицы A на величину $90^\circ \times k$ градусов, где k — целое число.

Пример:

```
» M=[3.2, 7; 3.3.2:1.1.1]
```

```
M=
```

3	2	7
3	3	2
1	1	1

```
» rot90(M)
```

```
ans
```

=		
7	2	1
2	3	1
3	3	1

При выполнении ряда матричных вычислений возникает необходимость в выделении треугольных частей матриц. Следующие функции обеспечивают такое выделение:

- `tril(X)` — возвращает матрицу, все элементы которой выше главной диагонали X заменены нулями, неизменными остаются лишь элементы нижней треугольной части, включая элементы главной диагонали;

- `tril(X,k)` — возвращает неизменной нижнюю треугольную часть матрицы X начиная с k -й диагонали. При $k=0$ это главная диагональ, при $k>0$ — одна из верхних диагоналей, при $k<0$ — одна из нижних диагоналей;

- `triu(X)` — возвращает неизменной верхнюю треугольную часть матрицы X включая элементы главной диагонали, и заменяет нулями остальные элементы;
- `triu(X,k)` — возвращает неизменной верхнюю треугольную часть матрицы X начиная с k -й диагонали. При $k=0$ — это главная диагональ, при $k>0$ — одна из верхних диагоналей, при $k<0$ — одна из нижних диагоналей.

Имеются функции, относящиеся к различным *специальным матрицам*: сопровождающая матрица, тестовая матрица, матрица Адамара, матрица Ганкеля, матрица Гильберта, матрица Паскаля и др, которые довольно широко используются при решении достаточно серьезных задач матричного исчисления.

СТАТИСТИЧЕСКАЯ ОБРАБОТКА

Самый простой анализ данных, содержащихся в некотором массиве, заключается в поиске его элементов с максимальным и минимальным значениями. В системе MATLAB определены следующие быстрые функции для нахождения минимальных и максимальных элементов массива:

- `max(A)` — возвращает наибольший элемент, если A — вектор; или возвращает вектор-строку, содержащую максимальные элементы каждого столбца, если A — матрица, в многомерных массивах работает с первой не единичной размерности;
- `max(A,B)` — возвращает массив того же размера, что A и B , каждый элемент которого есть максимальный из соответствующих элементов этих массивов;
- `max(A,[ ],dim)` — возвращает наибольшие элементы по столбцам или по строкам матрицы в зависимости от значения скаляра dim . Например, `max(A,[ ],1)` возвращает максимальные элементы каждого столбца матрицы A ;
- `[C,I]=max(A)` — кроме максимальных значений возвращает вектор индексов I этих элементов.

Примеры:

```
» A=magic(7)
```

30	39	48	1	10	19	28
38	47	7	9	18	27	29
46	6	8	17	26	35	37
5	14	16	25	34	36	45
13	15	24	33	42	44	4

21	23	32	41	43	3	12
22	31	40	49	2	11	20

» $C = \max(A)$

$C = 46 \ 47 \ 48 \ 49 \ 43 \ 44 \ 45$

» $C = \max(A, [], 1)$

$C =$

$46 \ 47 \ 48 \ 49 \ 43 \ 44 \ 45$

» $C = \max(A, [], 2)$

$C =$

48

47

46

45

44

43

49

Для быстрого нахождения элемента массива с минимальным значением служит следующая функция:

- $\min(A)$ — возвращает минимальный элемент, если A — вектор; или возвращает вектор-строку, содержащую минимальные элементы каждого столбца, если A — матрица;

- $\min(A,B)$ — возвращает массив того же размера, что A и B , каждый элемент которого есть минимальный из соответствующих элементов этих массивов;

- $\min(A, [], \dim)$ — возвращает наименьший элемент по столбцам или по строкам матрицы в зависимости от значения скаляра $\dim$. Например, $\min(A, [], 1)$ возвращает минимальные элементы каждого столбца матрицы A ;

- $[C,I] = \min(A)$ — кроме минимальных значений возвращает вектор индексов этих элементов.

Элементарная статистическая обработка данных в массиве обычно сводится к нахождению их среднего значения, медианы (срединного значения) и стандартного отклонения. Для этого в системе MATLAB определены следующие функции:

- $\text{mean}(A)$ — возвращает арифметическое среднее значение элементов массива, если A — вектор; или возвращает вектор-строку, содержащую средние значения элементов каждого столбца, если A —

матрица. Арифметическое среднее значение есть сумма элементов массива, деленная на их число;

- `mean(A,dim)` — возвращает среднее значение элементов по столбцам или по строкам матрицы в зависимости от значения скаляра `dim` (`dim=1` по столбцам и `dim=2` по строкам соответственно).

Примеры:

`A =`

1	2	6	4	8
6	7	13	5	4
7	9	0	8	12
6	6	7	1	2

`»mean(A)`

`ans =`

5.0000	6.0000	6.5000	4.5000	6.5000
--------	--------	--------	--------	--------

`»mean(A, 2)`

`ans =`

4.2000
7.0000
7.2000
4.4000

- `median(A)` — возвращает медиану, если `A` — вектор; или вектор-строку медиан для каждого столбца, если `A` — матрица;

- `median(A,dim)` — возвращает значения медиан для столбцов или строк матрицы в зависимости от значения скаляра `dim`;

- `std(X)` — возвращает стандартное отклонение элементов массива, вычисляемое по формуле, если `X` — вектор. Если `X` — матрица, то `std(X)` возвращает вектор-строку, содержащую стандартное отклонение элементов каждого столбца (обратите внимание, что оно отличается от среднеквадратического отклонения);

- `std(X,flag)` — возвращает то же значение, что и `std(X)`, если `flag=0`; если `flag=1`, функция `std(X,1)` возвращает среднеквадратическое отклонение (квадратный корень из несмещенной дисперсии);

- `std(X,flag,dim)` — возвращает стандартное или среднеквадратическое отклонения по рядам (`dim=2`) или по столбцам(`dim=1`) матрицы `X` в зависимости от значения переменной `dim`.

Примеры:

```
» X = linspace(0,3*pi,10)
X = Columns 1 through 7
0 1.0472 2.0944 3.1416 4.1888 5.2360 6.2832
Columns 8 through 10
7.3304 8.3776 9.4248
» s = std(X)
s =
3.1705
```

ФУНКЦИИ СОРТИРОВКИ ЭЛЕМЕНТОВ МАССИВА

Многие операции статистической обработки данных выполняются быстрее и надежнее, если данные предварительно отсортированы. Кроме того, нередко представление данных в отсортированном виде более наглядно и ценно. Ряд функций служит для выполнения сортировки элементов массива. Они представлены ниже.

- `sort(A)` — в случае одномерного массива `A` сортирует и возвращает элементы по возрастанию их значений; в случае двумерного массива происходит сортировка и возврат элементов каждого столбца. Допустимы вещественные, комплексные и строковые элементы. Если `A` принимает комплексные значения, то элементы сначала сортируются по абсолютному значению, а затем, если абсолютные значения равны, по аргументу. Если `A` включает NaN-элементы, `sort` помещает их в конец;

- `[B, INDEX] = sort(A)` — наряду с отсортированным массивом возвращает массив индексов `INDEX`. Он имеет размер `size(A)`, с помощью этого массива можно восстановить структуру исходного массива;

- `sort(A,dim)` — для матриц сортирует элементы по столбцам (`dim=1`) или по рядам в зависимости от значения переменной `dim`.

Примеры:

```
» A=magic(5)
A =
17      24      1      8      15
23      5      7      14     16
```

4	6	13	20	22	
10	12	19	21	3	
11	18	25	2	9	
» [B. INDEX] sort (A)					
B=					
4	5	1	2	3	
10	6	7	8	9	
11	12	13	14	15	
17	18	19	20	16	
23	24	25	21	22	
index=					
	3	2	1	5	4
	4	3	2	1	5
	5	4	3	2	1
	1	5	4	3	2
	2	1	5	4	3

- `sortrows(A)` — выполняет сортировку рядов массива `A` по возрастанию и возвращает отсортированный массив, который может быть или матрицей, или вектором-столбцом;

- `sortrows(A,column)` — возвращает матрицу, отсортированную по столбцам, точно указанным в векторе `column`. Например, `sortrows(A,[2 3])` сортирует строки матрицы `A` сначала по второму столбцу и затем, если его элементы равны, по третьему;

- `[B, index] = sort rows (A)` — также возвращает вектор индексов `index`. Если `A` — вектор-столбец, то `B=A(index)`. Если `A` — матрица размера $m \times n$, то `B=A(index,:)`;

- `sortrows(A)` — выполняет сортировку рядов массива `A` по возрастанию и возвращает отсортированный массив, который может быть или матрицей, или вектором-столбцом;

- `sortrows(A,column)` — возвращает матрицу, отсортированную по столбцам, точно указанным в векторе `column`. Например, `sortrows(A,[2 3])` сортирует строки матрицы `A` сначала по второму столбцу, и затем, если его элементы равны, по третьему;

- `[B, index] = sortrows (A)` — также возвращает вектор индексов `index`. Если `A` — вектор-столбец, то `B=A(index)`. Если `A` — матрица размера $m \times n$, то `B=A(index,:)`.

Примеры:

```
» A=[2 35689: 5 7 1 2 3 5:1 3 2 1 5 1:5 0 8 8 4 3]
A =
```

	2	3	5	6	8	9
	5	7	1	2	3	5
	1	3	2	1	5	1
	5	0	8	8	4	3

```
» B= sortrows(A)
B =
```

1	3	2	1	5	1
2	3	5	6	3	9
5	0	8	8	4	3
5	7	1	2	3	5

ВЫЧИСЛЕНИЕ КОЭФФИЦИЕНТОВ КОРРЕЛЯЦИИ

Под *корреляцией* понимается взаимосвязь некоторых величин, представленных данными — векторами или матрицами. Общепринятой мерой линейной корреляции является *коэффициент корреляции*. Его близость к единице указывает на высокую степень линейной зависимости. Данный раздел посвящен описанию функции для вычисления коэффициентов корреляции и определения ковариационной матрицы элементов массива. Приведенная ниже функция позволяет вычислить коэффициенты корреляции для входного массива данных.

- `corrcoef(X)` — возвращает матрицу коэффициентов корреляции для входной матрицы, строки которой рассматриваются как наблюдения, а столбцы — как переменные. Матрица $S = \text{corrcoef}(X)$ связана с матрицей ковариаций $C = \text{cov}(X)$ следующим соотношением: $S(i,j) = C(i,j) / \sqrt{C(i,i)C(j,j)}$;

- Функция $S = \text{corrcoef}(x,y)$, где x и y — векторы-столбцы, аналогична функции `corrcoef([x y])`.

Пример:

```
» M=magic(5)
```

```
M =
```

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22

```

10      12      19      21      3
11      18      25      2      9
» S=corrcoef(M)
S =
1.0000 0.0856 -0.5455 -0.3210 -0.0238
0.0856 1.0000 -0.0981 -0.6731 -0.3210
-0.5455 -0.0981 1.0000 -0.0981 -0.5455
-0.3210 -0.6731 -0.0981 1.0000 0.0856
-0.0238 -0.3210 -0.5455 0.0856 1.0000

```

В целом, корреляция данных довольно низкая. В данных, расположенных по диагонали — здесь коэффициенты корреляции равны 1, — вычисляется линейная корреляция переменной со своей копией.

Вычисление матрицы ковариации

Приведенная далее функция позволяет вычислить матрицу ковариации для массива данных.

- `cov(x)` — возвращает смещенную дисперсию элементов вектора x . Для матрицы, где каждая строка рассматривается как наблюдение, а каждый столбец — как переменная, `cov(x)` возвращает матрицу ковариации. `diag(cov(x))` — вектор смещенных дисперсий для каждого столбца и `sqrt(diag(cov(x)))` — вектор стандартных отклонений.

- Функция `C = cov(x,y)`, где x и y — векторы-столбцы одинаковой длины, равносильна функции `cov([x y])`.

Пример:

```

» D=[2 -3 6:3 6 -1:9 8 5]:C=cov(D)
C = 14.3333 16.3333 3.6667
16.3333 34.3333 -10.3333
3.6667 -10.3333 14.3333
» diag(cov(D))
ans =
14.3333
34.3333
14.3333
» sqrt(diag(cov(D)))
ans =
3.7859
5.8595
3.7859
» std(D)
ans =

```

РАБОТА С ГРАФИКАМИ

С понятием графики связано представление о *графических объектах*, имеющих определенные свойства. Большинство команд высокоуровневой графики, ориентированной на конечного пользователя, автоматически устанавливают свойства графических объектов и обеспечивают воспроизведение графики в нужных системе координат, палитре цветов, масштабе и т. д.

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

Построение графиков

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

`plot(X, Y)` — строит график функции $y(x)$, координаты точек (x, y) которой берутся из векторов одинакового размера Y и X . Если X или Y — матрица, то строится семейство графиков по данным, содержащимся в колонках матрицы.

Приведенный ниже пример иллюстрирует построение графиков двух функций — $\sin(x)$ и $\cos(x)$, значения функции которых содержатся в матрице Y , а значения аргумента x хранятся в векторе X :

```
» x=[0 12345];  
» Y=[sin(x):cos(x)];  
» plot(x,Y)
```

`plot(Y)` — строит график $y(i)$, где значения y берутся из вектора Y , а i представляет собой индекс соответствующего элемента. Если Y содержит комплексные элементы, то выполняется команда `plot(real`

(Y). $\text{imag}(Y)$). Во всех других случаях мнимая часть данных игнорируется. Пример:

» $x=2*\pi:0.02*\pi:2*\pi$;

» $y=\sin(x)+i*\cos(3*x)$;

» $\text{plot}(y)$

$\text{plot}(X,Y,S)$ — аналогична команде $\text{plot}(X,Y)$, но тип линии графика можно задавать с помощью строковой константы S.

Значениями константы S могут быть следующие символы.

Цвет линии

Y	Желтый
M	Фиолетовый
C	Голубой
R	Красный
G	Зеленый
B	Синий
W	Белый
K	Черный

Тип точки

.	Точка
o	Окружность
X	Крест
+	Плюс
*	Звездочка
S	Квадрат
D	Ромб
V	Треугольник (вниз)
A	Треугольник (вверх)
<	Треугольник (влево)
>	Треугольник (вправо)
P	Пятиугольник
H	Шестиугольник

Тип линии

-	Сплошная
;	Двойной пунктир
-.	Штрих-пунктир
--	Штриховая

Таким образом, с помощью строковой константы S можно изменять цвет линии, представлять узловые точки различными

отметками (точка, окружность, крест, треугольник с разной ориентацией вершины и т. д.) и менять тип линии графика.

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

`loglog(...)` — синтаксис команды аналогичен ранее рассмотренному для функции `plot(...)`. Логарифмический масштаб используется для координатных осей X и Y . Ниже дан пример применения данной команды:

```
» x=logspace(-1.3);  
» loglog(x.exp(x)./x)  
» grid on
```

Для построения графиков функций в полулогарифмическом масштабе используются следующие команды:

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

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

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

`bar(x, Y)` — строит столбцовый график элементов вектора Y (или группы столбцов для матрицы Y) со спецификацией положения столбцов, заданной значениями элементов вектора x , которые должны идти в монотонно возрастающем порядке;

`bar(Y)` — строит график значений элементов матрицы Y так же, как указано выше, но фактически для построения графика используется вектор `x=1:m`;

`bar(Y,WIDTH)` или `BAR(Y,WIDTH)` — команда аналогична ранее рассмотренным, но со спецификацией ширины столбцов (при `WIDTH > 1` столбцы в одной и той же позиции перекрываются). По умолчанию задано `WIDTH = 0.8`.

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

`N=hist(Y)` — возвращает вектор чисел попаданий для 10 интервалов, выбираемых автоматически. Если Y — матрица, то

выдается массив данных о числе попаданий для каждого из ее столбцов;

$N = \text{hist}(Y, M)$ — аналогична вышерассмотренной, но используется M интервалов (M — скаляр);

$N = \text{hist}(Y, X)$ — возвращает числа попаданий элементов вектора Y в интервалы, центры которых заданы элементами вектора X ;

$[N, X] = \text{HIST}(\dots)$ — возвращает числа попаданий в интервалы и данные о центрах интервалов.

Команда $\text{hist}(\dots)$ с синтаксисом, аналогичным приведенному выше, строит график гистограммы. В следующем примере строится гистограмма для 1000 случайных чисел и выводится вектор с данными о числе их попаданий в интервалы, заданные вектором x :

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

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

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

$\text{stairs}(Y)$ — строит лестничный график по данным вектора Y ;

$\text{stairs}(X, Y)$ — строит лестничный график по данным вектора Y с координатами x переходов от ступеньки к ступеньке, заданными значениями элементов вектора X ;

$\text{stairs}(\dots, S)$ — аналогична по действию вышеописанным командам, но строит график линиями, стиль которых задается строками S .

Следующий пример иллюстрирует построение лестничного графика:

```
» x=0:0.25:10;
```

» stairs(x,x^2).

Существуют и такие типы графиков: графики с зонами погрешности, график дискретных отсчетов функции, графики в полярной системе координат, угловые гистограммы, графики векторов, график проекций векторов на плоскость, контурные графики и др.

Создание массивов данных и построение трехмерной графики

Трехмерные поверхности обычно описываются функцией двух переменных $z(x, y)$. Специфика построения трехмерных графиков требует не просто задания ряда значений x и y , то есть векторов x и y . Она требует определения для X и Y двумерных массивов — матриц. Для создания таких массивов служит функция `meshgrid`. В основном она используется совместно с функциями построения графиков трехмерных поверхностей. Функция `meshgrid` записывается в следующих формах:

$[X,Y] = \text{meshgrid}(x)$ — аналогична $[X,Y] = \text{meshgrid}(x,x)$;

$[X,Y,Z] = \text{meshgrid}(x,y,z)$ — возвращает трехмерные массивы, используемые для вычисления функций трех переменных и построения трехмерных графиков;

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

Пример:

» $[X,Y] = \text{meshgnd}(1:4,13:17)$

$X =$

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

$Y =$

13	13	13	13
14	14	14	14
15	15	15	15
16	16	16	16
17	17	17	17

Приведем еще один пример применения функции `meshgrid`:

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

Такой вызов функции позволяет задать опорную плоскость для построения трехмерной поверхности при изменении дигг/от-2до2с шагом 0.2. Дополнительные примеры применения функции `meshgrid` будут приведены далее при описании соответствующих команд. Рекомендуется ознакомиться также с командами `surf` и `slice` (ломтик).

Функция `ndgrid` является многомерным аналогом функции `meshgrid`:

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

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

Пример применения функции `ndgrid` представлен ниже:

```
»[X1.'X2] = ndgrid(-2:.2:2, -2:.2:2);
```

```
»Z = X1.* exp(-X1.^2 - X2.^2);
```

```
»mesh(Z)
```

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

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

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

Ниже дан пример построения трехмерной поверхности, описываемой функцией

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

```
» [X,Y]=meshgrid([-3:0.15:3]);
```

```
» Z=X.A2+Y.^2;
```

```
» plot3(X,Y,Z)
```

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

`mesh(X,Y,Z,C)` — выводит в графическое окно сетчатую поверхность $Z(X,Y)$ с цветами узлов поверхности, заданными массивом C ;

`mesh(X,Y,Z)` — аналог предшествующей команды при $C=Z$. В данном случае используется функциональная окраска, при которой цвет задается высотой поверхности.

Возможны также формы команды `mesh(x,y,Z)`, `mesh(x,y,Z,C)`, `mesh(Z)` и `mesh(Z,C)`.

Функция `mesh` возвращает дескриптор для объекта класса `surface`. Ниже приводится пример применения команды `mesh`:

```
» [X,Y]=meshgrid([-3:0,15:3]);  
» Z=X.^2+Y.^2;  
» mesh(X,Y,Z)
```

Закрашенные секторы часто используются для построения круговых диаграмм. Для этого в MATLAB служит команда `pie`:

`pie(X)` — строит круговую диаграмму по данным нормализованного вектора X / $SUM(X)$. $SUM(X)$ —сумма элементов вектора. Если $SUM(X) \leq 1.0$, то значения в X непосредственно определяют площадь секторов;

`pie(X,EXPLODE)` — строит круговую диаграмму, у которой отрыв секторов от центра задается вектором `EXPLODE`, который должен иметь тот же размер, что и вектор данных X .

Следующий пример строит цветную круговую диаграмму с пятью секторами, причем последний сектор отделен от остальных:

```
» X=[1 2 3 4 5];  
» pie(X,[0 0 0 0 2])
```

Для закрашки многоугольников, определенных в пространстве, служит команда `fill3`. Ниже представлены основные ее формы:

`fill3(X,Y,Z,C)` — строит закрашенный многоугольник в пространстве с данными вершин, хранящимися в векторах X , Y и Z , и цветом, заданным палитрой C . При построении нескольких закрашенных многоугольников параметры команды должны быть матрицами;

`fill3(X1,Y1,Z1,C1,X2,Y2,Z2,C2,...)` — другой вариант построения нескольких закрашенных многоугольников в пространстве;

`fill 3` — функция, возвращающая вектор-столбец дескрипторов объектов класса `patch`.

Иногда используются объемные круговые диаграммы. Для их построения служит команда `pie3`:

`pie3(...)` — аналогична команде `pie(...)`, но дает построение объемных секторов.

В приведенном ниже примере строится объемная диаграмма с отделением двух секторов, показанная на рис. 6.51:

» `X=[1 2345];`

» `pie3(X,[0 0 1 01])`

Следующий пример показывает действие команды `fill3`:

» `fill3(rand(5,4),rand(5,4).rand(5.4).rand(5.4))`

Для построения цилиндра в виде трехмерной фигуры применяется функция

`cylinder`:

`[X,Y,Z]=cylinder(R,N)` — создает массивы X , Y и Z , описывающие цилиндрическую поверхность с радиусом R и числом узловых точек N для последующего построения с помощью функции `surf(X,Y,Z)`;

`[X,Y,Z]=cylinder(R)` и `[X,Y,Z]=cylinder` — подобны предшествующей функции для $N=20$ и $R=[1\ 1]$.

Пример построения объемного цилиндра:

» `[X,Y,Z]=cylinder(10.30);`

» `surf(X,Y,Z,X)`

Для расчета массивов X , Y и Z координат точек сферы как трехмерной фигуры используется функция `sphere`:

`[X,Y,Z]=sphere(N)` — генерирует матрицы X , Y и Z размера $(N+1) \times (N+1)$ для последующего построения сферы с помощью команд `surf(X,Y,Z)` или `surf(X,Y,Z)`;

`[X,Y,Z]=sphere` — аналогична предшествующей функции при $N=20$.

Пример применения этой функции:

» `[X,Y,Z]=sphere(30);`

» `surf(X,Y,Z)`

К числу специальных видов графики относится построение объемных фигур с помощью плоских треугольников. Для построения таких фигур в виде каркаса (без окраски и отображения плоскостей) используется команда `trimesh`:

`trimesh(TRI,X,Y,Z,C)` — построение объемной каркасной фигуры с треугольниками, специфицированными матрицей поверхности `TRI`, каждая строка которой содержит три элемента и задает одну треугольную грань путем указания индексов, по которым координаты выбираются из векторов X , Y , Z . Цвета ребер задаются вектором C ;

`trimesh(TRI,X,Y,Z)` — построение, аналогичное предшествующему при $C=Z$, т. е. с цветом ребер, зависящим от значений высоты;

`H=trimesh(...)` — строит график и возвращает дескрипторы графических объектов;

`trimesh(..., 'param'. 'value'. 'param', 'value'...)` — добавляет значения 'value' для параметров 'param'.

Следующий пример иллюстрирует применение команды `trimesh` для построения случайной объемной фигуры, параметры которой задаются с помощью генератора случайных чисел:

```
» x = rand(1.40);  
» y = rand(1.40);  
» z = sin(x.*y);  
» tri = delaunay(x,y);  
» tmesh(tri .x.y.z)
```

В результате вычислений в системе MatLab обычно получается большой массив данных, который трудно анализировать без наглядной визуализации.

Функция `PLOT` строит графики в двухмерном пространстве.

Пример:

```
>> x=0:0.01:2;  
>> y=sin(x);  
>> z=cos(x);  
>> plot(x,y,x,z)
```

На рисунке 1 представлено вычисление данной функции.

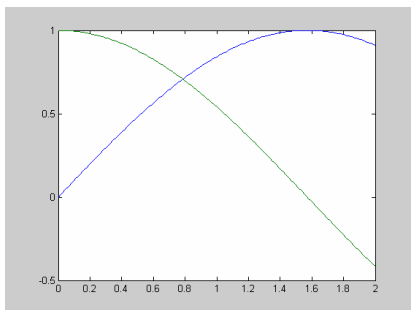


Рис. 1. Построение нескольких графиков в одних и тех же осях координат

Ещё одним примером является построение графиков в полярных координатах.

Функция `POLAR` предназначена специально для построения графиков в полярных координатах.

Пример:

```
>> phi=0:0.01:2*pi;  
>> r=tan(5*phi);  
>> polar(phi,r)
```

На рис.2. представлен результат выполнения данной функции.

Диаграммы

В столбцовых диаграммах каждому элементу массива соответствует один столбец. Для наглядности столбцы, относящиеся к одной строке, группируются.

Функция BAR – порождает наглядную столбцовую диаграмму.

Пример:

```
>> M=[5 2 4; 1 3 5; 2 1 4];
```

```
>> Bar(M)
```

Данная функция порождает изображение, представленное на рис. 3.

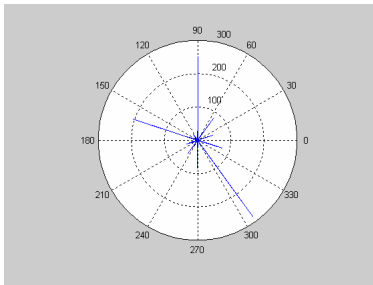


Рис. 2. Построение графика в полярных координатах

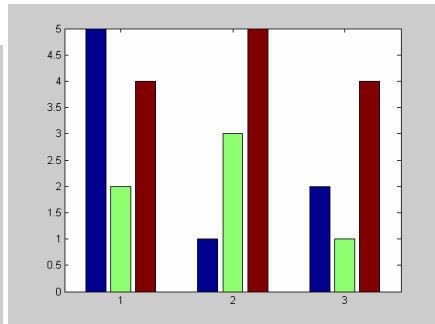


Рис. 3. Столбцовая диаграмма

Наглядность показа элементов матрицы только усилится, если перейти к трехмерным изображениям.

Функция BAR3 строит столбцовые диаграммы в трехмерном пространстве.

Пример:

```
>> K=[5 2 3; 7 1 5; 1 5 3];
```

```
>> bar3(K)
```

Вызов данной функции порождает картину, представленную на рис. 4.

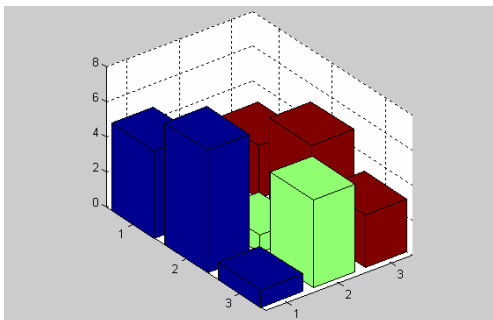


Рис.4 . Столбцовая диаграмма в трехмерном пространстве.

Помимо функций построения столбцовых диаграмм, в системе MatLab имеются также функции построения круговых диаграмм.

Функция `PIE(x)` строит круговые диаграммы.

Пример:

```
>> X=[3, 7, 1,2];
```

```
>> pie(X)
```

На рис. 5 можно увидеть получившуюся диаграмму.

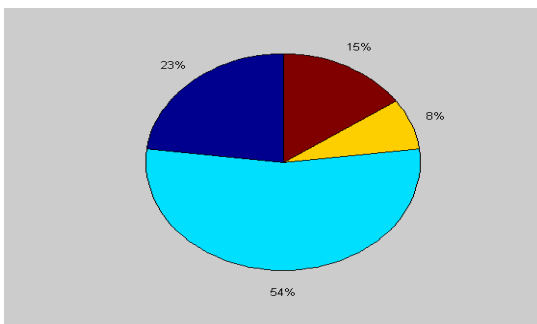


Рис. 5. Круговая диаграмма

Помимо изображения элементов массивов в виде столбцов или «кусков пирога», система MatLab предоставляет еще способ визуализации, когда каждый элемент массива изображается отрезком вертикальной линии с маленьким кружочком на конце. Такие диаграммы строит функция `STEM(x,y)`-они называются черенковые.

Наиболее интересным применением такого рода диаграмм является случай, когда такие «черенки» позиционируются вдоль горизонтальной оси не напротив номеров элементов массива, а напротив значений другого одномерного массива.

Пример:

Черенковая диаграмма, для построения которой вызывается функция `STEM`.

```
>> x=-1:0.1:1;
>> y=asin(x);
>> stem(x,y)
```

На рис.6 наглядно отображена данная диаграмма.

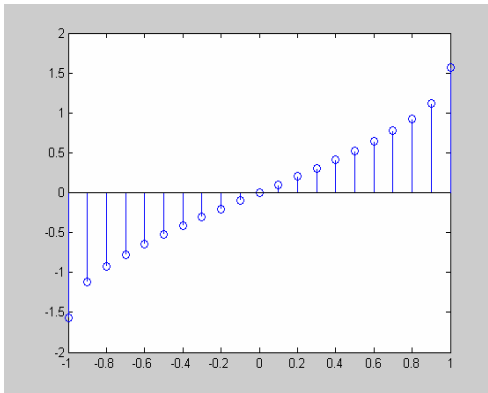


Рис. 6. Черенковая диаграмма

Проследить за изменением величин и одновременно узнать вклад значений в общую сумму позволяет функция `AREA`, выводящая диаграмму с областями.

Пример:

```
>> A=[5 3 1; 10 3 7; 4 7 3; 2 5 1];
>> area(A)
```

В результате получается диаграмма с областями, изображённая на рис.7.

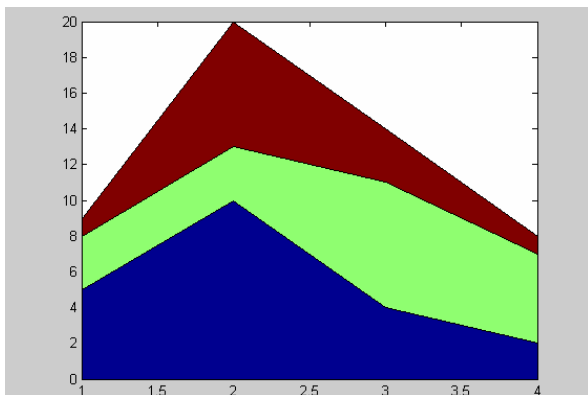


Рис.7 . Диаграмма с областями

Создание закрашенных многоугольников

Для создания закрашенного пятна в виде многоугольника может использоваться команда `PATCH` .

Пример:

```
>> x=[0 2 5 6 4];  
>> y=[1 2 0 5 1];  
>> patch ( x, y, [1 0.5 0])
```

```
>> x=[2 5 8 1 0 9 15];  
>> y=[5 0.5 0 4 10 20 1 ];  
>> patch (x,y,[1 0 1])
```

На рис. 8 вы можете наблюдать исполнение этой команды.

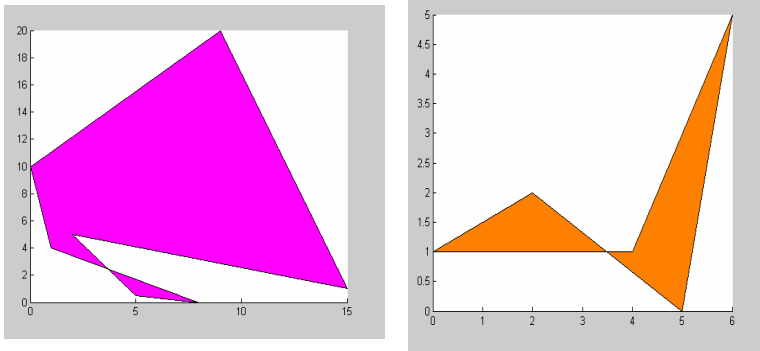


Рис. 8. Использование функции PATCH

Для построения массива произвольных многоугольников используется функция FILL3.

Пример:

```
>> fill3 (rand(3,4), rand(3,4), rand(3,4), rand(3,4))
```

На рис.9 представлен результат выполнения данной функции.

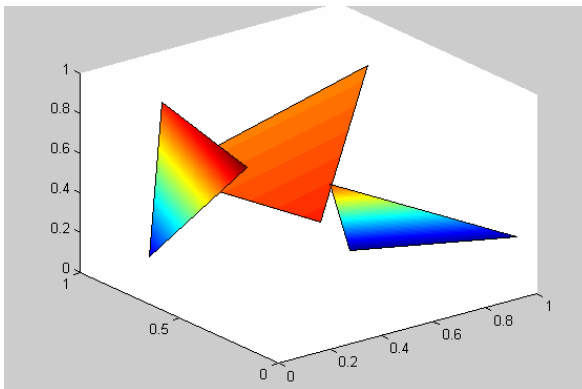


Рис.9. Произвольные многоугольники

Трехмерная графика

Построение цилиндра

Для построения цилиндров используется функция CYLINDER.

Пример

```
>> [X,Y,Z]=cylinder(1,6);
```

```
>> surf(X,Y,Z,X)
```

На рис.10 показан результат построения цилиндра для данного примера.

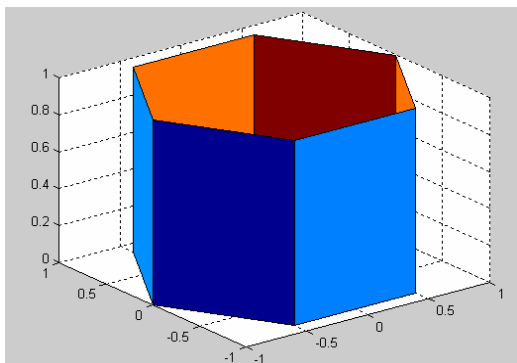


Рис. 10. Построение цилиндра

Построение сферы

Для построения сфер используют функцию SPHERE.

Пример

```
>> [X,Y,Z]=sphere(55);
```

```
>> surf(X,Y,Z,[0 0 1])
```

На рис.11 показан результат выполнения данной функции.

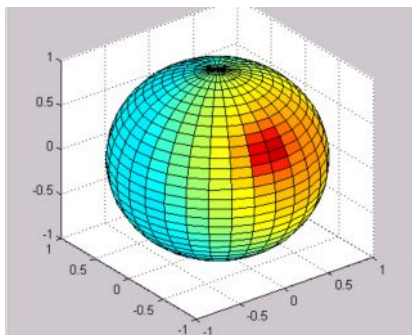


Рис.11. Построение сферы

Построение поверхности столбцами

Еще один тип представления поверхности, когда она строится из многочисленных столбцов. Для этого используется функция MESHZ.

Пример:

```
>> [X,Y]=meshgrid([-3:0.15:3]);  
Z=X.*2+Y.*2;  
meshz(X,Y,Z)
```

На рис.12 представлен результат выполнения данной функции.

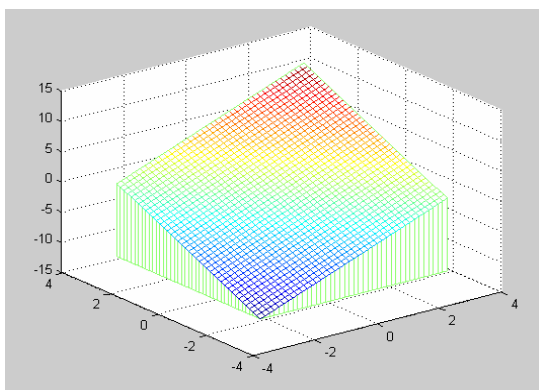


Рис. 12. Построение поверхности столбцами

В MatLab просто обеспечивается построение графиков сложных поверхностей. Надо только знать, какой командой реализуется тот или иной график.

Основы форматирования двумерных графиков

Форматирование графиков

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

В версии MATLAB 6.5 для изменения свойств графиков (их форматирования) используются принципы визуального контроля за стилем (видом) всех объектов графиков. Это позволяет легко придать графикам должный вид, перед записью их в виде файлов на диск. .

Форматирование линий графиков

MATLAB имеет возможность легко настраивать и корректировать свойства графиков с помощью специальных средств. В версии MATLAB 6.5 они существенно изменены. Так, в предшествующей версии для настройки (форматирования) графиков использовался специальный редактор свойств — Graphics Properties Editor (Редактор свойств графики). Его можно было вызвать из меню File окна командного режима MATLAB с помощью команды Show Graphics Properties Editor (Показать редактор свойств графики).

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

При построении графиков появляется графическое окно. Иногда оно бывает скрыто ранее имеющимися окнами как системы MATLAB, так и других работающих в среде Windows 95/98/Me/2000/NT4 приложений. Если вы не увидели графика, заданного для построения, то поищите его в списке открытых окон (приложений), нажимая клавиши Alt + Tab, и выберите из списка нужное окно. Окна графики имеют изображение логотипа системы MATLAB. Щелкнув на кнопке

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

Например, указав в режиме редактирования мышью на линию графика (и дважды щелкнув левой клавишей), можно увидеть окно форматирования линий графика, показанное на рис. 13 слева. Часть окна графики с выделенным графиком видна справа. Обратите внимание на появление на линии графика ряда черных квадратиков: они используются для указания курсором мыши именно на линию графика, а не на другие объекты.

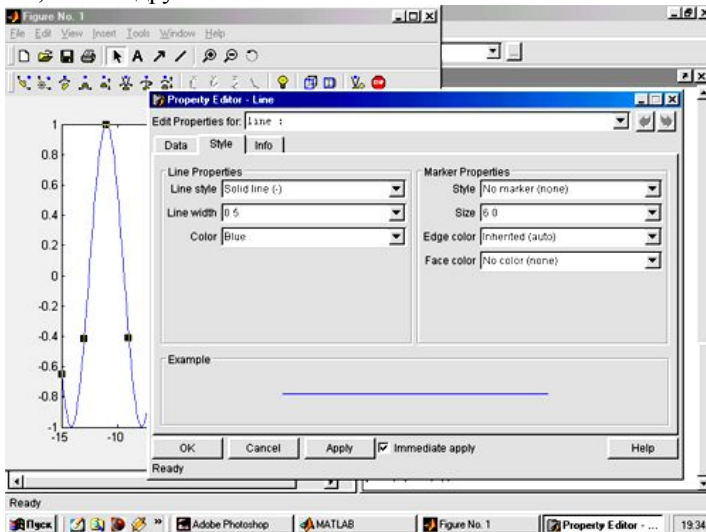


Рис. 13 . Окно графика (справа) и окно форматирования линий (слева)

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

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

Форматирование маркеров опорных точек

В нашем случае опорные точки задаются ранжированной переменной x , имеющей ряд значений от -15 до +15 с шагом 0.1. Эти точки появляются на графике, если в поле свойств маркера Marker Properties (Свойства маркера) из меню Style выбрать стиль маркера. На рис.14, к примеру, показано построение графика с маркерами опорных точек в виде окружностей.

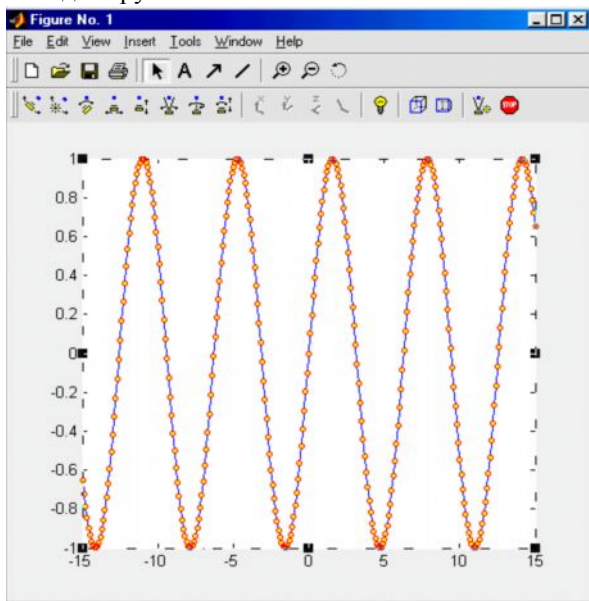


Рис. 14. Пример задания параметров маркеров

Можно задавать размеры маркеров, цвет их заливки и цвет окантовки. Так, на рисунке 14 при его просмотре на экране цветного дисплея маркеры имеют вид окружностей с условным размером 4, цветом окантовки красным и цветом заливки желтым. Маркеры можно задавать в виде окружностей, прямоугольников, крестиков, ромбиков. Применение маркеров делает графики более наглядными.

Форматирование линий и маркеров

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

```
»x=-6:1:6;
```

```
» plot(x.sin(x).x.sin(x).^3.x.sin(x).^5):
```

Рис. 15 показывает пример такого форматирования для графика, полученного исполнением этих команд.

Кстати, обратите внимание на то, как заданы степени синуса. Записать эти выражения в виде $\sin(x)^2$ и $\cos(x)^2$ будет грубейшей ошибкой, поскольку x здесь вектор.

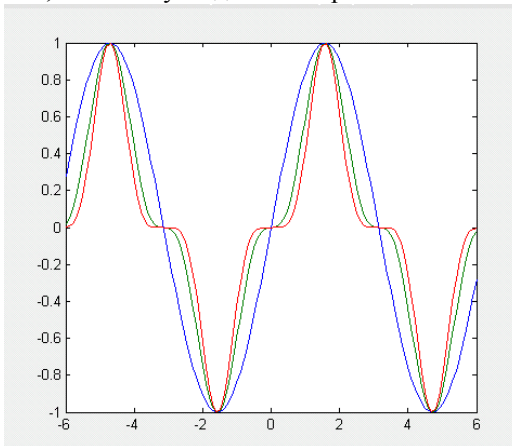


Рисунок 15 - Пример форматирования для графика трех функций

Форматирование осей графиков

Аналогично описанным выше правилам выполняется форматирование и других объектов графиков. Например, указав курсором мыши на оси графиков (на них тоже есть метки в виде черных квадратиков) и дважды щелкнув левой клавишей мыши, можно увидеть окно форматирования объектов дескрипторной графики Property Editor (Редактор свойств), изображенное на рис. 16, настроенного на форматирование осей.

Окно графического редактора свойств дескрипторной графики имеет множество вкладок, настройки которых довольно очевидны, и ничто не мешает читателю поэкспериментировать с ними несколько

минут. Это позволит понять простоту и одновременно высокую эффективность средств форматирования объектов графики. Например, вы можете задать линейный или логарифмический масштаб осей (вкладка Scale), открытая на рисунке 16, нормальное или инверсное направление осей (X, Y, а в случае трехмерных графиков и Z), показ сетки (параметр Grid Show), изменить стиль осей и цвета фона (вкладка Style), нанести у осей надписи (вкладка Label (Ярлык)) и пр.

Рис.17 показывает график синусоиды после некоторых операций по форматированию осей. Здесь, как и на рис.16, задано построение сетки Grid по осям X и Y, построение надписей (просто буквы X и Y) по координатным осям и построение титульной надписи. Так же, на рис. 16, показано, в открытом виде, меню расширенных инструментальных средств графического окна. Некоторые из возможностей форматирования объектов графики мы рассмотрим позже, по мере описания типов графиков.

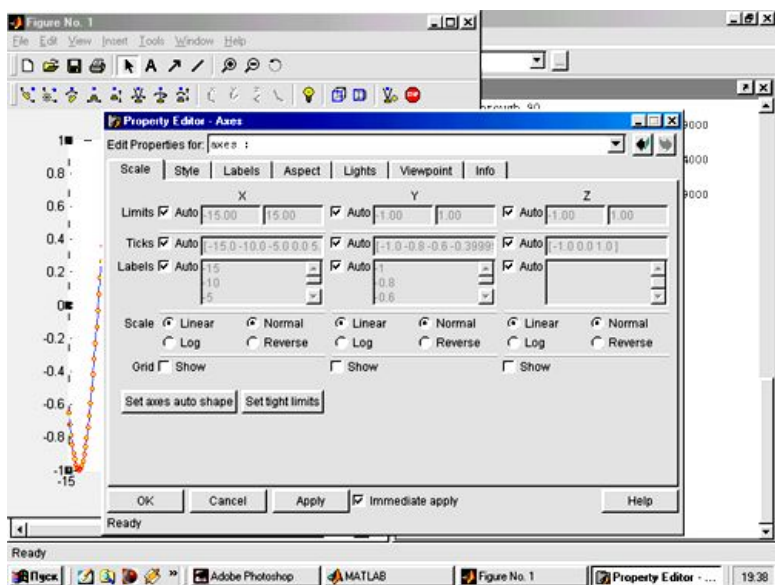


Рис.16. Пример форматирования осей графика

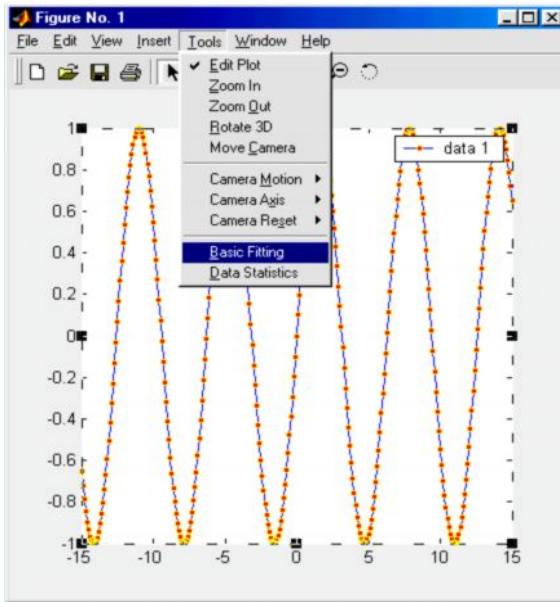


Рис.17. Пример построения графика синусоиды

Нанесение надписей и стрелок на график

Дополнительно на график можно нанести надписи с помощью кнопки панели инструментов с буквой А. Место надписи фиксируется щелчком мыши. На рис. 18 показан отформатированный график с текстовым блоком, созданным таким образом в левой верхней части поля графика.

Здесь показано контекстное меню правой клавиши мыши, поясняющее выбор размера символов надписи (и другие возможности этого меню). Напоминаем, что это меню появляется при щелчке правой кнопки мыши на заданном объекте. В этом меню имеются все команды, доступные для данного объекта в данной ситуации.

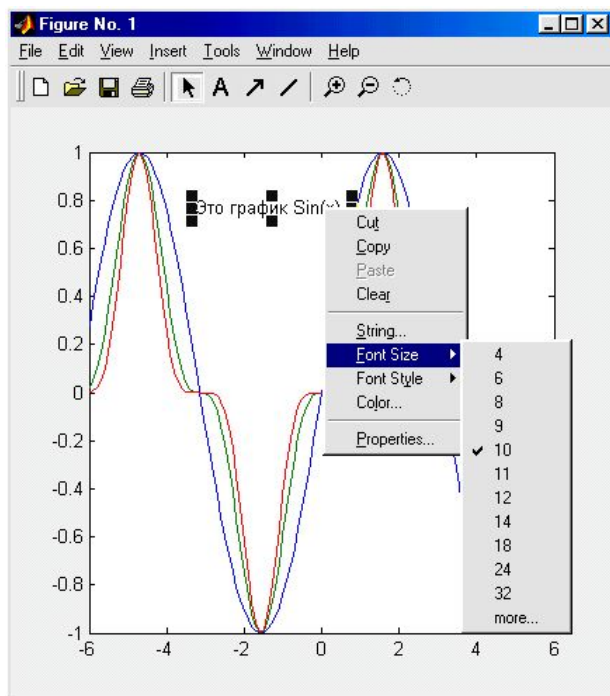


Рис.18 . Нанесение надписи на отформатированный график

Полученную таким образом надпись можно выделить и перенести мышью в любое другое место. Рис. 19 показывает процесс создания еще двух надписей с переносом их текстового блока в нужное место. Надписи сделаны с разным размером символов и разным стилем. Особенно приятно, что при задании на надписи возведения в степень знаком $^$ надпись на экране отображается в естественном математическом виде (степень в виде верхнего индекса).

На рис. 19, в частности, показано задание надписей разным стилем, а также задание стрелки с помощью соответствующей кнопки панели инструментов. Это стрелку в режиме редактирования графика можно перемещать и вращать мышью, а также менять ее длину. Можно также наносить на график и обычные линии (без стрелки).

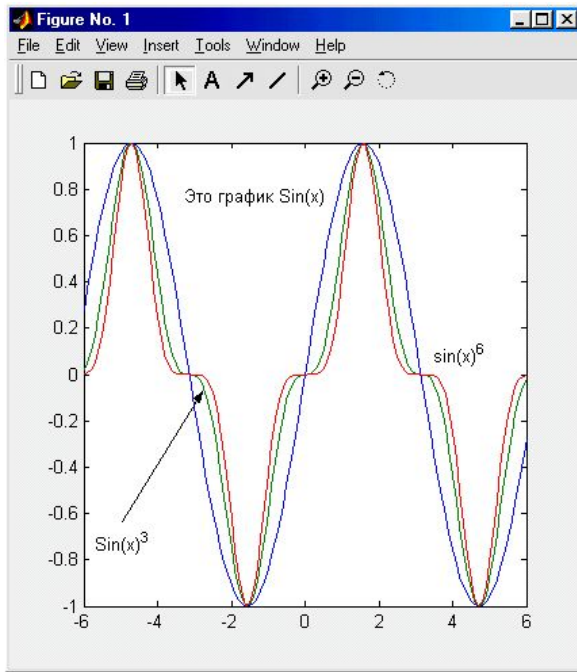


Рис. 19 . Окончательно отформатированный график трех функций

Построение легенды и шкалы цветов на графике

Дополнительно можно изменить размеры графика (см. меню Tools (Инструменты) и его команды Zoom In (Увеличить) и Zoom Out (Уменьшить)), начать поворот графика мышью (команда Rotate 3D), добавить отрезок прямой или иной графический примитив (подменю Add) и подключить к графику легенду — пояснение в виде отрезков линий со справочными надписями, размещаемое внутри графика или около него. Поскольку наш график содержит три кривые, то легенда представляет собой обозначение этих трех линий в правом верхнем углу рис. 20. Каждая линия имеет тот же цвет, что и на графике (и тот же стиль).

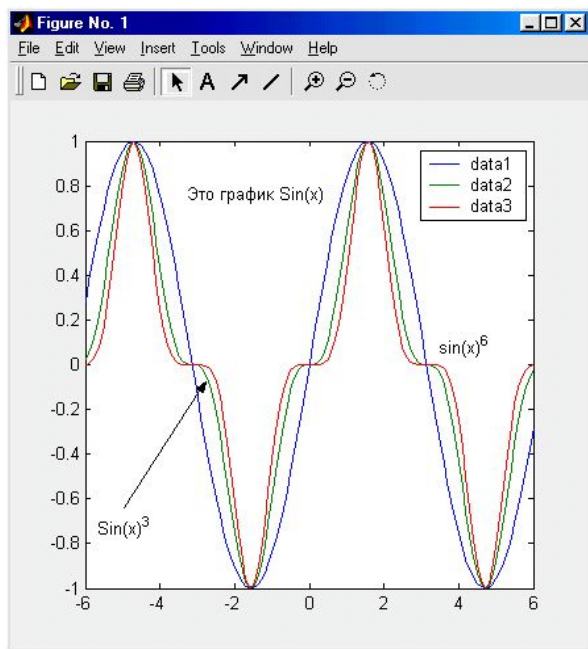


Рис.20. Окончательно сформированный график

Так, для установки над графиком титульной надписи используется следующая команда:

`title('string')` — установка на двумерных и трехмерных графиках титульной надписи, заданной строковой константой 'string'.

Для установки надписей возле осей x , y и z используются следующие команды:

```
xlabel('String')
ylabel('String')
zlabel('String')
```

Соответствующая надпись задается символьной константой или переменной 'String'. Пример установки титульной надписи и надписей по осям графиков приводится ниже:

```
» [X,Y]=meshgrid([-3:0.1:3]);
» Z=sin(X)./(X.^2+Y.^2+0.3);
» surf(X,Y,Z)
» colorbar
» colormap(gray)
```

- » shading interp
- » xlabel('Axis X')
- » ylabel('Axis Y')
- » zlabel('Axis Z')
- » title('Surface graphic')

Часто возникает необходимость добавления текста в определенное место графика, например для обозначения той или иной кривой графика. Для этого используется команда `text`:

`text(X,Y, 'string')` — добавляет в двумерный график текст, заданный строковой константой 'string', так что начало текста расположено в точке с координатами (X, Y). Если X и Y заданы как одномерные массивы, то надпись помещается во все позиции $[x(i), y(i)]$;

`text(X,Y,Z, 'string')` — добавляет в трехмерный график текст, заданный строковой константой 'string', так что начало текста расположено в позиции, заданной координатами X, Y и Z.

В приведенном ниже примере надпись «График функции $\sin(x^3)$ » размещается под кривой графика в позиции (-4, 0.7):

- » `x=10:0.1:10;`
- » `plot(x,sin(x).^3)`
- » `text(-4,0.7,'Graphic sin(x)^3')`

Очень удобный способ ввода текста предоставляет команда `gtext`: `gtext('string')` — задает выводимый на график текст в виде строковой константы 'string' и выводит на график перемещаемый мышью маркер в виде крестика. Установив маркер в нужное место, достаточно щелкнуть любой кнопкой мыши для вывода текста;

`gtext(C)` — позволяет аналогичным образом разместить многострочную надпись из массива строковых переменных C.

Пример применения команды `gtext`:

- » `x=-15:0.1:15;`
- » `plot(x, sin(x).^3)`
- » `gtext('Function sin(x)*3')`

Пояснение в виде отрезков линий со справочными надписями, размещаемое внутри графика или около него, называется легендой. Для создания легенды используются различные варианты команды `legend`:

`legend(string1,string2, strings,...)` — добавляет к текущему графику легенду в виде строк, указанных в списке параметров;

`legend (H,string1,string2, strings,...)` — помещает легенду на график, содержащий объекты с дескрипторами H, используя заданные строки как метки для соответствующих дескрипторов;

`legend( AX , ...)` — помещает легенду в осях (объект класса `axes`) с дескриптором `AX`;

`legend( M)` — размещает легенду, используя данные из строковой матрицы `M`;

`legend OFF` — устраняет ранее выведенную легенду;

`legend` — перерисовывает текущую легенду, если таковая имеется;

`legend(legendhandle)` — перерисовывает легенду, указанную дескриптором `legendhandle`;

`legend( ....Pos)` — помещает легенду в точно определенное место, специфицированное параметром `Pos`:

`Pos=0` — лучшее место, выбираемое автоматически;

`Pos=1` — верхний правый угол;

`Pos=2` — верхний левый угол;

`Pos=3` — нижний левый угол;

`Pos=4` — нижний правый угол;

`Pos=-1` — справа от графика.

Чтобы перенести легенду, установите на нее курсор, нажмите левую кнопку мыши и перетащите легенду в необходимую позицию.

`[legh,objh]=legend(...)` — эта функция возвращает дескриптор объекта для легенды (`legh`) и матрицу `objh`, содержащую дескрипторы объектов, из которых легенда состоит.

Команда `legend` может использоваться с двумерной и трехмерной графикой и со специальной графикой — столбцовыми и круговыми диаграммами и т. д. Двойным щелчком можно вывести легенду на редактирование.

Обычно графики выводятся в режиме автоматического масштабирования. Следующие команды класса `axis` меняют эту ситуацию:

`axis([XMIN XMAX YMIN YMAX])` — установка диапазонов координат по осям `x` и `y` для текущего двумерного графика;

`axis([XMIN XMAX YMIN YMAX ZMIN ZMAX])` - установка диапазонов координат по осям `x`, `y` и `z` текущего трехмерного графика;

`axis auto` — установка параметров осей по умолчанию;

`axis manual` — «замораживает» масштабирование в текущем состоянии, чтобы при использовании команды `hold on` следующие графики использовали те же параметры осей;

`axis tight` — устанавливает диапазоны координат по осям в соответствии с диапазонами изменения данных;

`axis ij` — задает «матричную» прямоугольную систему координат с началом координат в левом верхнем углу, ось `i` — вертикальная,

размечаемая сверху вниз, ось j — горизонтальная и размечается слева направо;

`axis xy` — устанавливает декартову систему координат с горизонтальной осью x , размечаемой слева направо, и вертикальной осью y , размечаемой снизу вверх. Начало координат размещается в нижнем левом углу;

`axis equal` — включает масштаб с одинаковым расстоянием между метками по осям x , y и z ;

`axis image` — устанавливает масштаб, при котором пиксели изображения становятся квадратами;

`axis square` — устанавливает текущие оси в виде квадрата (или куба в трехмерном случае) с одинаковым расстоянием между метками и одинаковой длиной осей;

`axis normal` — восстанавливает масштаб, отменяя установки `axis equal` и `axis square`;

`axis vis3d` — «замораживает» пропорции осей для возможности поворота трехмерных объектов;

`axis off` — убирает с осей их обозначения и маркеры; `0 axis on` — восстанавливает ранее введенные обозначения осей и маркеры;

`V=axis` — возвращает вектор-строку, содержащую коэффициенты масштабирования для текущего графика. Если текущий график двумерный, то вектор имеет 4 компонента, если трехмерный — 6 компонентов.

В математической, физической и иной литературе при построении графиков в дополнение к разметке осей часто используют масштабную сетку. Команды `grid` позволяют задавать построение сетки или отменять это построение:

- `grid on` — добавляет сетку к текущему графику;
- `grid off` — отключает сетку;
- `grid` — последовательно производит включение и отключение сетки.

Для изменения масштаба двумерных графиков используются команды класса `zoom`:

- `zoom` — переключает состояние режима интерактивного изменения масштаба для текущего графика;
- `zoom (FACTOR)` устанавливает масштаб в соответствии с коэффициентом `FACTOR`;
- `zoom on` — включает режим интерактивного изменения масштаба для текущего графика;

- `zoom off` — выключает режим интерактивного изменения масштаба для текущего графика;
- `zoom out` — обеспечивает полный просмотр, т. е. устанавливает стандартный масштаб графика;
- `zoom хon` или `zoom уon` — включает режим изменения масштаба только по оси *x* или по оси *y*;
- `zoom reset` — запоминает текущий масштаб в качестве масштаба по умолчанию для данного графика;
- `zoom(FIG,OPTION)` — применяется к графику, заданному дескриптором `FIG`, при этом `OPTION` может быть любым из перечисленных выше аргументов.

Поскольку графика `MATLAB` обеспечивает получение цветных изображений, в ней есть ряд команд для управления цветом и различными световыми эффектами. Среди них важное место занимает установка палитры цветов. Палитра цветов `RGB` задается матрицей `MAP` из трех столбцов, определяющих значения интенсивности красного (`red`), зеленого (`green`) и синего (`blue`) цветов. Их интенсивность задается в относительных единицах от 0.0 до 1.0. Например, `[0 0 0]` задает черный цвет, `[1 1 1]` — белый цвет, `[0 0 1]` — синий цвет. При изменении интенсивности цветов в указанных пределах возможно задание любого цвета. Таким образом, цвет соответствует общепринятому формату `RGB`.

Для установки **палитры цветов** служит команда `colormap`, записываемая в следующих формах:

- `colormap('default')` — устанавливает палитру по умолчанию, при которой распределение цветов соответствует радуге;
- `colormap(MAP)` — устанавливает палитру `RGB`, заданную матрицей `MAP`;
- `C=colormap` — функция возвращает матрицу текущей палитры цветов `C`. `m`-файл с именем `colormap` устанавливает свойства цветов для текущего графика.

Команда `help graphSd` наряду с прочим выводит полный список характерных палитр, используемых графической системой `MATLAB`:

- `hsv` — цвета радуги;
- `hot` — чередование черного, красного, желтого и белого цветов;
- `gray` — линейная палитра в оттенках серого цвета;
- `bone` — серые цвета с оттенком синего;
- `copper` — линейная палитра с оттенками меди;
- `pink` — розовые цвета с оттенками пастели;

- white — палитра белого цвета;
- flag — чередование красного, белого, синего и черного цветов;
- lines — палитра с чередованием цветов линий;
- colorcube — расширенная палитра RGB;
- jet — разновидность палитры HSV;
- prism — призматическая палитра цветов;
- cool — оттенки голубого и фиолетового цветов;
- autumn — оттенки красного и желтого цветов;
- spring — оттенки желтого и фиолетового цветов;
- winter — оттенки синего и зеленого цветов;
- summer — оттенки зеленого и желтого цветов.

Все эти палитры могут служить параметрами команды `colormap`, например `colormap(hsv)` фактически устанавливает то же, что и команда `colormap('default')`. Примеры применения команды `colormap` будут приведены в следующих разделах.

Для окраски поверхностей используется команда `shading`, которая управляет объектами `surface` (поверхность) и `patch` (заплата), создаваемыми командами и функциями `surf`, `mesh`, `pcolor`, `fill` и `fill 3`. Команда `shading` (затенение) работает с параметрами и имеет следующий вид:

- `shading flat` — задает окраску ячеек или граней в зависимости от текущих данных;
- `shading interp` — задает окраску с билинейной интерполяцией цветов;
- `shading faceted` — равномерная раскраска ячеек поверхности (принята по умолчанию).

Эти команды устанавливают свойства `EdgeColor` и `FaceColor` для графических объектов `surface` и `patch` в зависимости от того, какая из команд — `mesh` (сетчатая поверхность) или `surf` (затененная поверхность) — используется. Примеры применения команд `shading` уже приводились.

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

- » `Z=peaks(40)` ;
- » `mesh(Z)` ;

Здесь первая команда создает массив точек поверхности с помощью одного из ряда встроенных в ядро системы MATLAB готовых описаний таких поверхностей. Вторая команда просто строит эту поверхность по опорным точкам с использованием интерполяции для

промежуточных точек. Таким образом создается цветная каркасная поверхность, как бы сотканная из разноцветных проволок. Несмотря на множество кнопок, пользование панелью инструментов 3D-графики Camera (Камера) достаточно просто, если представить себе, что вы смотрите на предмет через объектив фотокамеры. Наглядные рисунки на кнопках поясняют смысл их действия — это перемещение и вращение 3D-рисунков относительно тех или иных координатных осей, включение отображения перспективы, изменение цветовой схемы и др. Приемы форматирования двумерной графики можно использовать при работе с трехмерной графикой — вывод надписи на график, вывод легенды (кстати, теперь объемной) и шкалы цветов.

ЧИСЛЕННЫЕ МЕТОДЫ

Элементарные средства решения СЛУ

Решение систем линейных уравнений (СЛУ) относится к самой массовой области применения матричных методов. В этом разделе вы найдете ответы на вопросы, каким образом применяются указанные методы и какие дополнительные функции имеет система MATLAB для решения систем линейных уравнений.

Как известно, обычная СЛУ имеет вид

$$a_{11}X_1, a_{12}, X_2, \dots, a_{1n}X_n = b_1.$$

Здесь $a_{11}, a_{12}, \dots, a_{1n}$ — коэффициенты, образующие матрицу A , которые могут иметь действительные или комплексные значения, $x_1, x_2, \dots, x_n$ — неизвестные, образующие вектор X , и $b_1, b_2, \dots, b_n$ — свободные члены (действительные или комплексные), образующие вектор B . Эта система может быть представлена в матричном виде как $AX=B$, где A — матрица коэффициентов уравнений; X — искомый вектор неизвестных; B — вектор свободных членов. В зависимости от вида матрицы A и ее характерных особенностей MATLAB позволяет реализовать различные методы решения.

Для реализации различных алгоритмов решения СЛУ и связанных с ними матричных операций применяются следующие операторы: $+, -, *, /, \setminus, ^*, '.$ Как отмечалось ранее, MATLAB имеет два различных типа арифметических операций — поэлементные и для массивов (векторов и матриц) в целом. Матричные арифметические операции определяются *правилами линейной алгебры*.

Арифметические операции сложения и вычитания над массивами выполняются поэлементно. Знак точки «.» отличает операции над элементами массивов от матричных операций. Однако, поскольку операции сложения и вычитания одинаковы для матрицы и элементов мас-

сива, знаки «.+» и «.-» не используются. Рассмотрим другие операторы и выполняемые ими операции:

- * — матричное умножение;
- $C = A*B$ — линейное алгебраическое произведение матриц A

и B :

$$C(i, j) = \sum_{k=1}^n A(i, k) \cdot B(k, j)$$

Для случая нескаларных A и B число столбцов матрицы A должно равняться числу строк матрицы B . Скаляр может умножаться на матрицу любого размера.

- / — *правое* деление. Выражение $X=B/A$ дает решение ряда систем линейных уравнений $AX=B$, где A — матрица размера $m \times n$; B — матрица размера $n \times k$;

- \ — *левое* деление. Выражение $X=B \backslash A$ дает решение ряда систем линейных уравнений $XA=B$, где A — матрица размера $m \times n$ и B — матрица размера $n \times k$. Если A — квадратная матрица, то $A \backslash B$ — примерно то же самое, что и $\text{inv}(A)*B$, в остальных случаях возможны варианты, отмеченные ниже.

Если A — матрица размера $n \times n$, а B — вектор-столбец с n компонентами или матрица с несколькими подобными столбцами, тогда $X=A \backslash B$ — решение уравнения $AX=B$, которое находится хорошо известным методом исключения Гаусса.

Если A — матрица размера $m \times n$ и $m < n$, а B представляет собой вектор-столбец с m компонентами или матрицу с несколькими такими столбцами, тогда система оказывается недоопределенной или переопределенной и решается на основе минимизации второй нормы невязок.

Ранг k матрицы A находится на основе QR-разложения с выбором ведущего элемента. Полученное решение X будет иметь не больше чем k ненулевых компонентов на столбец. Если $k < n$, то решение, как правило, не будет совпадать с $\text{pinv}(A)*B$, которое имеет наименьшую норму невязок $\|X\|$.

- ^ — возведение матрицы в степень. $X^{\wedge p}$ — это X в степени p , если p — скаляр. Если p — целое число, то степень матрицы вычисляется путем умножения X на себя p раз. Если p — целое отрицательное число, то X сначала инвертируется. Для других значений p вычисляются собственные значения и собственные векторы, так что если $[V,D]=\text{eig}(X)$, то $X^{\wedge p}=V*D.^{\wedge p}/V$. Если X — скаляр и P — матрица, то $X^{\wedge P}$ — это скаляр X , возведенный в матричную степень P . Если X и P — матрицы, то $X^{\wedge P}$ становится некорректной операцией и система

выдает сообщение об ошибке. Возможный вариант решения матричного уравнения $AX=B$ с применением оператора $\wedge$ можно представить как $X=B*A\wedge-1$.

- $'$ — транспонирование матрицы, то есть замена строк столбцами и наоборот. Например, A' — транспонированная матрица A . Для комплексных матриц транспонирование дополняется комплексным сопряжением. Транспонирование при решении СЛУ полезно, если в матрице A переставлены местами столбцы и строки.

При записи СЛУ в матричной форме необходимо следить за правильностью записи матрицы A и вектора B .

Пример (в виде m-файла):

```
A=[2    1          0          1:
   1    -3          2          4;
  -5     0         -1         -7:
   1    -6          2          6]:
B=[8    9          -5          0]:

X1=B/A
X2=B*A^-1
X3=B*inv(A)
```

Эта программа выдает результаты решения тремя способами:

```
X1 =
3.0000 -4.0000-1.0000i 1.0000
X2 =
3.0000 -4.0000-1.0000i 1.0000
X3 =
3.0000 -4.0000-1.0000i 1.0000
```

Как и следовало ожидать, результаты оказываются одинаковыми для всех трех методов. При решении систем линейных уравнений, особенно с разреженной матрицей коэффициентов, полезно применение функций `colmmd` (`colamd`), `symmmd` (`symamd`).

Функции для решения систем линейных уравнений с ограничениями

Теперь рассмотрим функции, введенные для решения систем линейных уравнений с ограничениями методом наименьших квадратов:

- $X=\text{lscov}(A,B,V)$ —возвращает вектор X решения СЛУ вида $A*X=B+e$, где e — вектор шумов, который имеет ковариационную матрицу V . Реализуется метод наименьших квадратов в присутствии

шумов с известной ковариацией. Прямоугольная матрица A должна быть размера $m \times n$, где $m > n$. При решении минимизируется следующее выражение: $(AX-b)' \text{inv}(V) * (AX-b)$. Решение имеет вид $X = \text{inv}(A' * \text{inv}(V) * A) * A' * \text{inv}(V) * B$. Алгоритм решения, однако, построен так, что операция инвертирования матрицы V не проводится;

- $[X, dX] = \text{lscov}(A, B, V)$ — возвращает также стандартную погрешность X , помещая ее в переменную dX . Статистическая формула для стандартной погрешности вычисления коэффициентов имеет следующий вид:

$$\begin{aligned} \text{mse} &= B' * (\text{inv}(V) - \text{inv}(V) * A * \text{inv}(A' * \text{inv}(V) * A) * A' * \text{inv}(V)) * B. / (m - n) \\ dX &= \text{sqrt}(\text{diag}(\text{inv}(A' * \text{inv}(V) * A) * \text{mse})) \end{aligned}$$

- $X = \text{lsqnonneg}(A, B)$ — решение СЛЮ $AX=B$ методом наименьших квадратов с неотрицательными ограничениями. A — действительная прямоугольная матрица, B — действительный вектор. Вектор X содержит неотрицательные элементы $X_j \geq 0$, где $j = 1, 2, \dots, n$. Критерий: минимизация второй нормы вектора $B=AX$;

- $X = \text{lsqnonneg}(A, B, X0)$ — решение СЛЮ с явно заданным неотрицательным начальным значением X для итераций;

- $[X, W] = \text{lsqnonneg}(\dots)$ — вместе с решением возвращает вторую норму вектора остатков в квадрате;

- $[X, W, W1] = \text{lsqnonneg}(\dots)$ — вместе с решением возвращает вторую норму вектора остатков в квадрате и вектор остатков $W1$;

- $[X, W, W1, \text{exitflag}] = \text{lsqnonneg}(\dots)$ — вместе с решением возвращает вторую норму вектора остатков в квадрате, вектор остатков $W1$ и флаг exitflag , равный 1, если решение сходится после заданного по умолчанию числа итераций, и 0 — в противном случае;

- $[X, W, W1, \text{exitflag}, \text{output}] = \text{lsqnonneg}(\dots)$ — дополнительно возвращает число итераций в output ;

- $[X, W, W1, \text{exitflag}, \text{output}, \text{lambda}] = \text{lsqnonneg}(\dots)$ — дополнительно возвращает вектор lambda , минимизирующий произведение $\text{lambda} * W1$ на последнем шаге итераций решения, $\text{lambda}(i)$ близко к нулю, когда $X(i)$ положительно, $\text{lambda}(i)$ отрицательно, когда $X(i)$ равно 0;

- $[X, W, W1, \text{exitflag}, \text{output}, \text{lambda}] = \text{lsqnonneg}(A, B, X0, \text{options})$ — обычно используется, если при предыдущей попытке решения системы $\text{exitflag} \neq 1$ или если необходимо изменить заданный по умолчанию порог отбора по X - $\text{tol} X$, равный $10 * \max(\text{size}(A)) * \text{norm}(A, 1) * \text{eps}$ (произведению первой нормы матрицы, большего из измерений матрицы, машинной точности и 10). Также используется такая форма за-

писи, как $X = \text{lsqnonneg}(A, B, X0, \text{options})$. Параметры options должны быть предварительно заданы при помощи функции optimset. Функция lsqnonneg использует только поля 'display' и 'tolX' структуры options. Поэтому наиболее часто используемая вместе с lsqnonneg форма записи функции — $\text{options} = \text{optimset}('C \text{ tol } X', \text{tol value})$, где tolvalue — новое значение порога отбора по X.

Применение ограничений позволяет избежать получения отрицательных корней, хотя и ведет к появлению несколько больших погрешностей решения, чем в случае решений без ограничений. Пример:

```
» C = [ 4 3 : 12 5 : 3
12 ] ; b = [ 1.45, 41 ] ; D = lsqnonneg ( C . b ' )
D =
2.2242
2.6954
```

Решение СЛУ с разреженными матрицами

Решение СЛУ с разреженными матрицами — хотя и не единственная, но несомненно одна из основных сфер применения аппарата разреженных матриц, описанного в уроке 12. Ниже приведены функции, относящиеся к этой области применения разреженных матриц. Большинство описанных методов относится к итерационным, т. е. к тем, решение которых получается в ходе ряда шагов — итераций, постепенно ведущих к получению результата с заданной погрешностью или с максимальным правдоподобием. Описанные ниже функции MATLAB могут и должны применяться и при решении обычных СЛУ — без разреженных матриц. *[Использование всех этих функций, кроме lsqr, как правило, ограничено системами уравнений, в которых A — нормальная квадратная матрица, т. е. $A^* A = A A^*$, где A^* — сопряженная (эрмитова) матрица A. — Примеч. ред.]*

Точное решение, метод наименьших квадратов и сопряженных градиентов

- $\text{lsqr}(A, B)$ — возвращает точное решение X СЛУ $A^* X = B$, если матрица последовательная, в противном случае — возвращает решение, полученное итерационным методом наименьших квадратов. Матрица коэффициентов A должна быть прямоугольной размера $t \times n$, а вектор-столбец правых частей уравнений B должен иметь размер t. Условие $m > n$ может быть и необязательным. Функция lsqr начинает итерации от начальной оценки, по умолчанию представляющей собой вектор размером n, состоящий из нулей. Итерации производятся или до сходимости к решению, или до появления ошибки, или до достижения максимального числа итераций (по умолчанию равного $\min(20, m,$

n) — либо 20, либо числу уравнений, либо числу неизвестных). Сходимость достигается, когда отношение вторых норм векторов $\text{norm}(B-Ax)/\text{norm}(B)$ меньше или равно погрешности метода tol (по умолчанию $1e-6$);

- $\text{lsqr}(A,B,\text{tol})$ — возвращает решение с заданной погрешностью (порогом отбора) tol ;

- $\text{lsqr}(A,b,\text{tol}.\text{maxit})$ — возвращает решение при заданном максимальном числе итераций maxit вместо, возможно, чересчур малого числа, заданного по умолчанию;

- $\text{lsqr}(A,b,\text{tol}.\text{maxit},M)$ и $\text{lsqr}(A,b,\text{tol}.\text{maxit},M1,M2)$ — при решении используются матрица предусловий M или $M=M1*M2$, так что производится решение системы $\text{inv}(M)*A*x=\text{inv}(M)*b$ относительно x . Если $M1$ или $M2$ — пустые матрицы, то они рассматриваются как единичные матрицы, что эквивалентно отсутствию входных условий вообще;

- $\text{lsqr}(A,B,\text{tol}.\text{maxit},M1,M2,X0)$ — точно задается начальное приближение $X0$. Если $X0$ — пустая матрица, то по умолчанию используется вектор, состоящий из нулей;

- $X = \text{lsqr}(A,B,\text{tol}.\text{maxit},M1,M2,X0)$ — при наличии единственного выходного параметра возвращает решение X . Если метод lsqr сходится, выводится соответствующее сообщение. Если метод не сходится после максимального числа итераций или по другой причине, на экран выдается относительный остаток $\text{popp}(B-A*X)/\text{norm}(B)$ и номер итерации, на которой метод остановлен;

- $[X.\text{flag}] = \text{lsqr}(A,X,\text{tol}.\text{maxit},M1,M2,X0)$ — возвращает решение X и флаг flag , описывающий сходимость метода;

- $[X.\text{flag}.\text{relres}] = \text{lsqr}(A,X,\text{tol}.\text{maxit},M1,M2,X0)$ — также возвращает относительную вторую норму вектора остатков $\text{rel res}=\text{norm}(B-A*X)/\text{norm}(B)$. Если флаг flag равен 0, то $\text{relres}<\text{tol}$;

- $[X.\text{flag}.\text{relres}.\text{iter}] = \text{bicg}(A,B,\text{tol}.\text{maxit},M1,M2,X0)$ — также возвращает номер итерации, на которой был вычислен X . Значение iter всегда удовлетворяет условию $0<\text{iter}<\text{maxit}$;

- $[X.\text{flag}.\text{relres}.\text{iter}.\text{resvec}] = \text{lsqr}(A,B,\text{tol}.\text{maxit},M1,M2,X0)$ — также возвращает вектор вторых норм остатков resvec для каждой итерации начиная с $\text{res-vec}(1)=\text{norm}(B-A*X0)$. Если флаг flag равен 0, то resvec имеет длину $\text{iter}+1$ и $\text{resvec}(\text{end})<\text{tol}*\text{norm}(B)$. Возможны значения flag , равные 0, 1, 2, 3 и 4. Значения flag предоставляют следующие данные о сходимости решения:

- $\text{flag}=0$ - решение сходится при заданной точности tol и числе итераций не более заданного maxit ;

- flag=1 - число итераций равно заданному maxit, но сходимость не достигнута;
- flag=2 - матрица предусловий М плохо обусловлена;
- flag=3 - процедура решения остановлена, поскольку две последовательные оценки решения оказались одинаковыми;
- flag=4 - одна из величин в процессе решения вышла за пределы допустимых величин чисел (разрядной сетки компьютера).

Если значение flag больше нуля, то возвращается не последнее решение, а то решение, которое имеет минимальное значение отношения вторых норм векторов $\text{norm}(B-A*x)/\text{norm}(B)$.

Пример:

» A=[0 012; 1300; 0101; 1010];

» B=[11; 7; 6; 4];

Введенные в этом примере матрица A и вектор B будут использованы и в других примерах данного раздела. В примере процесс итераций сходится на пятом шаге с относительным остатком (отношением вторых норм векторов невязки и свободных членов) $1.9 \cdot 10^{-13}$.

Пример:

» lsqr(A,B,1e-6.5)

Isqr converged at iteration 5 to a solution
with relative residual

1.9e-013

ans =

1.0000

2.0000

3.0000

4.0000

Вычисление нулей функции одной переменной

Ряд функций системы MATLAB предназначен для работы с функциями. По аналогии с дескрипторами графических объектов могут использоваться объекты класса дескрипторов функций, задаваемых с помощью символа @, например: » fe=@exp.

Подфункциями понимаются как встроенные функции, например $\sin(x)$ или $\exp(x)$, так и функции пользователя, например $f(x)$, задаваемые как m-файлы-функции.

Численные значения таких функций, заданных дескрипторами, вычисляются с помощью функции feval:

» feval(fe,1.0)

ans =

2.7183

Для совместимости с прежними версиями можно записывать функции в символьном виде в апострофах, использование функции `eval` для их вычисления может быть более наглядно, не нужно создавать *m*-файл, но в учебном курсе мы будем стараться использовать новую нотацию, с использованием дескрипторов функций и `feval`, так как при этом программирование становится «более объектно-ориентированным», повышается скорость, точность и надежность численных методов. Поэтому, хотя везде в нижеследующем тексте вместо `@fun` можно подставить и символьное значение функции в апострофах, мы будем использовать нотацию `@fun` в дидактических целях. Все же иногда в интерактивном режиме можно использовать старую запись, чтобы не создавать *m*-файл функции.

Довольно часто возникает задача решения нелинейного уравнения вида $f(x) = 0$ или $(\gamma) = \frac{1}{2}(\text{дг})$. Последнее, однако, можно свести к виду $f(x) = f_1(x) - f_2(x) = 0$. Таким образом, данная задача сводится к нахождению значений аргумента x функции $f(x)$ одной переменной, при котором значение функции равно нулю. Соответствующая функция MATLAB, решающая данную задачу, приведена ниже:

- `fzero(@fun,x)` — возвращает уточненное значение x , при котором достигается нуль функции `fun`, представленной в символьном виде, при начальном значении аргумента x . Возвращенное значение близко к точке, где функция меняет знак, или равно NaN, если такая точка не найдена;

- `fzero(@fun,[x1 x2])` — возвращает значение x , при котором $\text{fun}(x)=0$ с заданием интервала поиска с помощью вектора $x=[x1\ x2]$, такого, что знак $\text{fun}(x(1))$ отличается от знака $\text{fun}(x(2))$. Если это не так, выдается сообщение об ошибке. Вызов функции `fzero` с интервалом гарантирует, что `fzero` возвратит значение, близкое к точке, где `fun` изменяет знак;

- `fzero(@fun,x.tol)` — возвращает результат с заданной погрешностью `tol`;

- `fzero(@fun,x.tol .trace)` — выдает на экран информацию о каждой итерации;

- `fzero(@fun,x.tol .trace,P1,P2,...)` — предусматривает дополнительные аргументы, передаваемые в функцию `fun(x,P1,P2,...)`. При задании пустой матрицы для `tol` или `trace` используются значения по умолчанию. Пример:

```
fzero(fun,x.[ ].[ ].P1).
```

Для функции `fzero` ноль рассматривается как точка, где график функции `fun` *пересекает* ось x , а не *касается* ее. В зависимости от формы задания функции `fzero` реализуются следующие хорошо из-

вестные численные методы поиска нуля функции: деления отрезка пополам, секущей и обратной квадратичной интерполяции. Приведенный ниже пример показывает приближенное вычисление $\pi/2$ из решения уравнения $\cos(x)=0$ с представлением косинуса дескриптором:

```
» x= fzero(@cos,[1 3])
x =
1.5708
```

В более сложных случаях настоятельно рекомендуется строить график функции $f(x)$ для приближенного определения корней и интервалов, в пределах которых они находятся. Ниже дан пример такого рода (следующий листинг представляет собой содержимое m-файла `fun1.m`):

```
%Функция, корни которой ищутся
function f=fun1(x)
f=0.25*x+sin(x)-1;
» x=0:0.1:10;
» plot(x,fun1(x));grid on;
```

Из рисунка нетрудно заметить, что значения корней заключены в интервалах $[0.5\ 1]$, $[2\ 3]$ и $[5\ 6]$. Найдем их, используя функцию `fzero`:

```
» x1=fzero(@fun1,[0.5 1])
x1 =
0.8905
» x2=fzero(@fun1,[2 3])
x2 =
2.8500
» x3=fzero(@fun1,[5.6])
x3 =
5.8128
» x3=fzero(@fun1,5.0.001)
x3 =
5.8111
```

Обратите внимание на то, что корень x_3 найден двумя способами и что его значения в третьем знаке после десятичной точки отличаются в пределах заданной погрешности $\text{tol}=0.001$. К сожалению, сразу найти все корни функции `fzero` не в состоянии. Решим эту же систему при помощи функции `f sol ve` из пакета Optimization Toolbox, которая решает систему нелинейных уравнений вида $f(x)=0$ методом наименьших квадратов, ищет не только точки пересечения, но и точки касания, `f solve` имеет почти те же параметры (дополнительный параметр — задание якобиана) и почти ту же запись, что и функция `lsqnonneg`, подробно рассмотренная ранее. Пример:

```

»fsolve(@fun1,0:10 )
ans =
Columns 1 through 7
0.8905    0.8905    2.8500    2.8500    2.8500    5.8128
5.8128
Columns 8 through 11
5.8128    2.8500    2.8500    10.7429

```

Для решения систем нелинейных уравнений следует также использовать функцию `solve` из пакета Symbolic Math Toolbox. Эта функция способна выдавать результат в символьной форме, а если такого нет, то она позволяет получить решение в численном виде. Пример:

```

» solve('0.25*x + sin(x) -1)
ans =
.89048708074438001001103173059554

```

Минимизация функции одной переменной

Еще одна важная задача численных методов — поиск *минимума* функции $f(x)$ в некотором интервале изменения x — от x_1 до x_2 . Если нужно найти *максимум* такой функции, то достаточно поставить знак «минус» перед функцией. Для решения этой задачи используется следующая функция:

```

[X.fval.exitflag,output] =
fminbnd(@fun,x1,x2,options, p1,p2,...)

```

- `fminbnd(@fun,x1,x2)` — возвращает значение x , которое является локальным минимумом функции $\text{fun}(x)$ на интервале $x_1 < x < x_2$;

- `fminbnd(@fun,x1,x2,options)` — сходна с описанной выше формой функции, но использует параметры `tolX`, `maxfuneval`, `maxiter`, `display` из вектора `options`, предварительно установленные при помощи команды `optimset` (смотрите описание `lsqnonneg`);

- `fminbnd(@fun,x1,x2,options.P1.P2...)` — сходна с описанной выше, но передает в целевую функцию дополнительные аргументы: `P1`, `P2`.... Если требуется использовать параметры вычислений по умолчанию, то вместо `options` перед `P1`, `P2` необходимо ввести `[]` (пустой массив);

- `[x.fval] = fminbnd(...)` — дополнительно возвращает значение целевой функции `fval` в точке минимума;

- `[x.fval .exitflag] = fminbnd(L...)` — дополнительно возвращает параметр `exitflag`, равный 1, если функция сошлась с использованием `options.tolX`, и 0, если достигнуто максимальное число итераций `options.maxiter`.

В этих представлениях используются следующие обозначения: $x1$, $x2$ — интервал, на котором ищется минимум функции; $P1, P2, \dots$ — дополнительные, помимо x , передаваемые в функцию аргументы; fun — строка, содержащая название функции, которая будет минимизирована; $options$ — вектор параметров вычислений. В зависимости от формы задания функции `fminbnd` вычисление минимума выполняется известными методами золотого сечения или параболической интерполяции. Пример:

```
» options=optimset('tolX',1.e-10):
[x]=fminbnd(@cos.3,4,options)
x =
3.1416
```

Решение ОДУ

Анализ поведения многих систем и устройств в динамике, а также решение многих задач в теории колебаний и в поведении упругих: оболочек обычно базируется на решении систем *обыкновенных дифференциальных уравнений* (ОДУ). Их, как правило, представляют в виде системы из дифференциальных уравнений первого порядка в форме Коши:

$$\frac{dy}{dt} = y'$$

$$y' = f(y, t)$$

с граничными условиями $y(t_0, t_{\text{end}}, p) = b$, где t_{end}, t_0 — начальные и конечные точки интервалов. Параметр t не обязательно означает время, хотя чаще всего решение дифференциальных уравнений ищется во временной области. Вектор b задает начальные и конечные условия.

Ниже коротко описаны численные методы решения обыкновенных дифференциальных уравнений (ОДУ) и некоторые вспомогательные функции, полезные для решения систем ОДУ. Дается представление о пакете расширения, решающем дифференциальные уравнения в частных производных.

Решатели ОДУ

Для решения систем ОДУ в MATLAB реализованы различные методы. Их реализации названы *решателями* ОДУ.

В этом разделе обобщенное название *so ver* (решатель) означает один из возможных численных методов решения ОДУ: *ode45*, *ode23*, *ode113*, *ode15s*, *ode23s*, *ode23t*, *ode23tb*, *bvp4c* или *pdepe*.

Решатели реализуют следующие методы решения систем дифференциальных уравнений, причем для решения жестких систем уравнений рекомендуется использовать только специальные решатели *ode15s*, *ode23s*, *ode23t*, *ode23tb*:

- *ode45* — одношаговые явные методы Рунге-Кутты 4-го и 5-го порядка. Это классический метод, рекомендуемый для начальной пробы решения. Во многих случаях он дает хорошие результаты;
- *ode23* — одношаговые явные методы Рунге-Кутты 2-го и 4-го порядка. При умеренной жесткости системы ОДУ и низких требованиях к точности этот метод может дать выигрыш в скорости решения;
- *ode113* — многошаговый метод Адамса-Башворта-Мултона переменного порядка. Это адаптивный метод, который может обеспечить высокую точность решения
- *ode23tb* — неявный метод Рунге-Кутты в начале решения и метод, использующий формулы обратного дифференцирования 2-го порядка в последующем

Несмотря на сравнительно низкую точность, этот метод может оказаться более эффективным, чем *ode15s*;

- *ode15s* — многошаговый метод переменного порядка (от 1 до 5, по умолчанию 5), использующий формулы численного дифференцирования. Это адаптивный метод, его стоит применять, если решатель *ode45* не обеспечивает решения;
- *ode23s* — одношаговый метод, использующий модифицированную формулу Розенброка 2-го порядка. Может обеспечить высокую скорость вычислений при низкой точности решения жесткой системы дифференциальных уравнений;
- *ode23t* — метод трапеций с интерполяцией. Этот метод дает хорошие результаты при решении задач, описывающих колебательные системы с почти гармоническим выходным сигналом;
- *bvp4c* служит для проблемы граничных значений систем дифференциальных уравнений вида $y'' = f(t, y)$, $F(y(a), y(b), p) = 0$ (краевая задача);
- *pdepe* нужен для решения систем параболических и эллиптических дифференциальных уравнений в частных производных, введен в ядро системы для поддержки новых графических функций Open GL, пакет расширения Partial Differential Equations Toolbox содержит более мощные средства.

Все решатели могут решать системы уравнений явного вида $y' = F(t, y)$. Решатели `ode15s` и `ode23t` способны найти корни дифференциально-алгебраических уравнений $M(t)y' = F(t, y)$, где M называется матрицей массы. Решатели `ode15s`, `ode23s`, `ode23t` и `ode23tb` могут решать уравнения неявного вида $M(t, y)y' = F(t, y)$. И наконец, все решатели, за исключением `ode23s`, который требует постоянства матрицы массы, и `bvp4c`, могут находить корни матричного уравнения вида $M(t, y)y' - F(t, y)$. `ode23tb`, `ode23s` служат для решения жестких дифференциальных уравнений. `ode15s` - жестких дифференциальных и дифференциально-алгебраических уравнений, `ode23t` - умеренно жестких дифференциальных и дифференциально-алгебраических уравнений.

Использование решателей систем ОДУ

В описанных далее функциях для решения систем дифференциальных уравнений приняты следующие обозначения и правила:

- `options` — аргумент, создаваемый функцией `odeset` (еще одна функция — `odeget` или `bvpget` (только для `bvp4c`) — позволяет вывести параметры, установленные по умолчанию или с помощью функции `odeset/bvpset`);
- `tspan` — вектор, определяющий интервал интегрирования $[t_0 \text{ } tfinal]$. Для получения решений в конкретные моменты времени $t_0, t_1, \dots, tfinal$ (расположенные в порядке уменьшения или увеличения) нужно использовать `tspan = [t0 t1 ... tfinal]`;
- `y0` — вектор начальных условий;
- `p1, p2, \dots` — произвольные параметры, передаваемые в функцию F ;
- T, Y — матрица решений Y , где каждая строка соответствует времени, возвращенном в векторе-столбце T .

Перейдем к описанию функций для решения систем дифференциальных уравнений:

- $[T, Y] = \text{solver}(@F, \text{tspan}, y_0)$ — где вместо `solver` подставляем имя конкретного решателя — интегрирует систему дифференциальных уравнений вида $y' = F(t, y)$ на интервале `tspan` с начальными условиями `y0`. `@F` — дескриптор ODE-функции. Каждая строка в массиве решений Y соответствует значению времени, возвращаемому в векторе-столбце T ;
- $[T, Y] = \text{solver}(@F, \text{tspan}, y_0, \text{options})$ — дает решение, подобное описанному выше, но с параметрами, определяемыми значениями аргумента `options`, созданного функцией `odeset`. Обычно используемые

параметры включают допустимое значение относительной погрешности RelTol (по умолчанию $1e-3$) и вектор допустимых значений абсолютной погрешности AbsTol (все компоненты по умолчанию равны $1e-6$);

- $[T, Y] = \text{solver}(@F, \text{tspan}, y0, \text{options}, p1, p2, \dots)$ — дает решение, подобное описанному выше, передавая дополнительные параметры $p1, p2, \dots$ в m -файл F всякий раз, когда он вызывается. Используйте $\text{options}=[]$, если никакие параметры не задаются;

- $[T, Y, TE, YE, IE] = \text{solver}(@F, \text{tspan}, y0, \text{options})$ — в дополнение к описанному решению содержит свойства Events, установленные в структуре options ссылкой на функции событий. Когда эти функции событий от (t, y) , равны нулю, производятся действия в зависимости от значения трех векторов value, isterminal, direction (их величины можно установить в m -файлах функций событий). Для i -й функции событий value(i) — значение функции, isterminal(i) — прекратить интеграцию при достижении функцией нулевого значения, direction(i) = 0, если все нули функции событий нужно вычислять (по умолчанию), +1 — только те нули, где функция событий увеличивается, -1 — только те нули, где функция событий уменьшается. Выходной аргумент TE — вектор-столбец времен, в которые происходят события (events), строки YE являются соответствующими решениями, а индексы в векторе IE определяют, какая из i функций событий (event) равна нулю в момент времени, определенный TE. Когда происходит вызов функции без выходных аргументов, по умолчанию вызывается выходная функция odeplot для построения вычисленного решения. В качестве альтернативы можно, например, установить свойство OutputFcn в значение 'odephas2' или 'odephas3' для построения двумерных или трехмерных фазовых плоскостей.

- $[T, X, Y] = \text{sim}(@\text{model}, \text{tspan}, -y0, \text{options}, ut, p1, p2, \dots)$ — использует модель SIMULINK, вызывая соответствующий решатель из нее. Пример:

```
[T, X, Y] = sim(@model....)
```

Параметры интегрирования (options) могут быть определены и в m -файле, и в командной строке с помощью команды odeset. Если параметр определен в обоих местах, определение в командной строке имеет приоритет.

Решатели используют в списке параметров различные параметры:

- NormControl — управление ошибкой в зависимости от нормы вектора решения $[on | \{off\}]$. Установите 'on', чтобы $\text{norm}(e) \leq \max(\text{RelTol} * \text{norm}(y), \text{AbsTol})$. По умолчанию все решатели используют

более жесткое управление по каждой из составляющих вектора решения;

- RelTol — относительный порог отбора [положительный скаляр]. По умолчанию $1e-3$ (0.1% точность) во всех решателях; оценка ошибки на каждом шаге интеграции $e(i) \leq \max(\text{RelTol} * \text{abs}(y(i)), \text{AbsTol}(i))$;

- AbsTol — абсолютная точность [положительный скаляр или вектор $\{1e-6\}$]. Скаляр вводится для всех составляющих вектора решения, а вектор указывает на компоненты вектора решения. AbsTol по умолчанию $1e-6$ во всех решателях;

- Refine - фактор уточнения вывода [положительное целое число] — умножает число точек вывода на этот множитель. По умолчанию всегда равен 1, кроме ODE45, где он 4. Не применяется, если $tspan > 2$;

- OutputFcn — дескриптор функция вывода [function] — имеет значение в том случае, если решатель вызывается без явного указания функции вывода, OutputFcn по умолчанию вызывает функцию odeplot. Эту установку можно поменять именно здесь;

- OutputSel — индексы отбора [вектор целых чисел] Установите компоненты, которые поступают в OutputFcn. OutputSel по умолчанию выводит все компоненты;

- Stats — установите статистику стоимости вычислений [on | off];

- Jacobian — функция матрицы Якоби [function constant matrix]. Установите это свойство на дескриптор функции FJac (если FJac(t, y) возвращает dF/dy) или на имя постоянной матрицы dF/dy ;

- Jpattern — график разреженности матрицы Якоби [имя разреженной матрицы]. Матрица S с $S(i,j) = 1$, если составляющая i F(t, y) зависит от составляющей j величины y, и 0 в противоположном случае;

- Vectorized — векторизованная ODE-функция [on | off]. Устанавливается на 'on', если ODE-функция F F(t,[y1 y2...]) возвращает вектор [F(t, y1) F(t, y2) ...];

- Events — [function] — введите дескрипторы функций событий, содержащих собственно функцию в векторе value, и векторы isterminal и direction (см выше);

- Mass — матрица массы [constant matrix function]. Для задач $M * y' = f(t, y)$ установите имя постоянной матрицы. Для задач с переменной M введите дескриптор функции, описывающей матрицу массы;

- **MstateDependence** — зависимость матрицы массы от y [`none` | `{weak}` | `{strong}`] — установите 'none' для уравнений $M(t)*y' = F(t, y)$. И слабая ('weak'), и сильная ('strong') зависимости означают $M(t, y)$, но 'weak' приводит к неявным алгоритмам решения, использующим аппроксимации при решении алгебраических уравнений;

- **MassSingular** — матрица массы M сингулярная [`yes` | `no` | `{maybe}`] (да/нет/может быть);

- **MvPattern** — разреженность (dMv/dy), график разреженности (см функцию `spy`) — введите имя разреженной матрицы S с $S(i,j) = 1$ для любого k , где (i, k) элемент матрицы массы $M(t, y)$ зависит от проекции] переменной y , и 0 в противном случае;

- **Initial Step** — предлагаемый начальный размер шага, по умолчанию каждый решатель определяет его автоматически по своему алгоритму;

- **Initial SI ore** — вектор начального уклона y_0 $y_0' = F(t_0, y_0)/M(t_0, y_0)$;

- **MaxStep** — максимальный шаг, по умолчанию во всех решателях равен одной десятой интервала `tspan`;

- **BDF (Backward Differentiation Formulas)** [`on` | `{off}`] — указывает, нужно ли использовать формулы обратного дифференцирования (методы `Gear`) вместо формул численного дифференцирования, используемых в `ode15s` по умолчанию;

- **MaxOrder** - максимальный порядок `ODE15S` [`1` | `2` | `3` | `4` | `{5}`].

Решатель `bvp4c` имеет очень небольшое число параметров, но можно вводить не только матрицу Якоби интегрируемой функции, но и матрицу Якоби, содержащую частные производные функции граничных условий по границам интервала и по неизвестным параметрам.

Покажем применение решателя ОДУ на ставшем классическом примере — решении уравнения Ван-дер-Поля, записанного в виде системы из двух дифференциальных уравнений:

$$\begin{aligned} y_1' &= y_2; \\ y_2' &= 100*(1-y_1)^2*y_2 - y_1 \\ &\text{при начальных условиях} \\ y_1(0) &= 0; \\ y_2(0) &= 1. \end{aligned}$$

Перед решением нужно записать систему дифференциальных уравнений в виде `ode-функции`. Для этого в главном меню выберем `File > New > M-File` и введем

```
function dydt = vdp100(t,y)
dydt = zeros(2,1); % a column vector
```

```
dydt(1) = y(2);
dydtC2 = 100*(1 - y(1)^2)*y(2) - y(1);
```

Сохраним m-файл-функцию. Тогда решение решателем ode15s и сопровождающий его график можно получить, используя следующие команды:

```
» [T,Y]=ode15s(@vdp100,[0 30],[2 0]);
» plot(T,Y)
» hold on;gtext('y1').gtext('y2')
```

Последние команды позволяют с помощью мыши нанести на графики решений $y_1 = y(1)$ и $y_2 = y(2)$, помечающие их надписи.

Описание системы ОДУ

Можно использовать m-файл типа odefunction (или m-file типа odefile для совместимости с прежними версиями, но последний случай мы рассматривать не будем, чтобы определить систему дифференциальных уравнений в одной из явных (первая формула) или неявных форм:

$y' = F(t, y)$, $My' = F(t, y)$ или $M(t)y' = Y(t, y)$,

где t — независимая переменная (скаляр), которая обычно представляет время; y — вектор зависимых переменных; F — функция от t и y , возвращающая вектор-столбец такой же длины как и y ; M и $M(t)$ — матрицы, которые не должны быть вырожденными. M может быть и константой.

Подготовим m-файл ode-функции vdp.m:

```
function [out1,out2,out3] = vdp(t,y,flag)
if nargin < 3 | isempty(flag)
out1 = [2.*y(2).*(1-y(2).^2)-y(1); y(1)];
else
switch(flag)
case 'inlt' % Return tspan, y0 and options
out1 = [0 20];
out2 = [2; 0];
out3 = [ ];
otherwise
error([' Unknown request '' flag ''.']);
end
end
```

Тогда решение системы с помощью решателя ode23 реализуется следующими командами:

```
» [T,Y] = ode23(@vdp,[0 20],[2 0]);
```

Еще проще работать с решателями нового поколения. Рассмотрим систему уравнений $y' + \text{abs}(y) = 0$; $y(0) = 0$; $y(4) = -2$.

Для решения в пределах отрезка $[0; 4]$ с помощью `bvp4c` достаточно привести эту систему к виду: $y' = -\text{abs}(y)$, $y(0) = 0$; $y(4) + 2 = 0$. Затем создаем две ode-функции: `twoode` и `twobc` в разных m-файлах:

```
function dydx = twoode(x,y)
dydx = [ y(2)
        -abs(yd) ] ;
function res = twobc(ya,yb) res = [ ya(1)
yb(1) + 2];
```

Теперь наберите в командной строке `type twobvp` и посмотрите само решение уравнения, которое содержится в уже имеющемся в системе файле `twobvp`. А исполнив команду `twodvp`, можно наблюдать результат решения в виде графиков. В решении вы найдете структуру узлов начальной сетки решения, которая поясняется ниже.

- `solinit` — это структура узлов начальной сетки решения (в любой шкале), но такая, что `solinit.x=a`, `solinit.x - b`. И функция y , и функция y' должны быть непрерывны на участке $[a, b]$. Для начальной итерации `solinit=bvp4c-mt(x,yinit.params)` в `bvp4c` отличается тем, что ваше начальное представление о функции y `yinit` можно вводить не только в виде вектора, но и как символьную функцию.

Рекомендуется просмотреть также пример `mat4bvp` и дополнительные примеры решения систем дифференциальных уравнений, приведенные в файле `odedemo`. Во многих случаях решение задач, сводящихся к решению систем дифференциальных уравнений, удобнее осуществлять с помощью пакета расширения `Simulink`.

Дескрипторная поддержка параметров решателя

При помощи перечисленных ниже функций можно получить и создать или изменить параметры решателя:

- `o=odeget(options, 'name')` — извлекает значение свойства, определенного строкой `'name'`, из структуры параметров `options`; возвращает пустую матрицу, если значение данного свойства в структуре `options` не определено. Можно ввести только первые буквы, которые однозначно определяют имя свойства. Пустая матрица `[ ]` — допустимый аргумент `options`;

Пример:

```
>> options = odesetCRelTol' , [1e-6 1e-
7] . 'AbsTol' , 6e-3);
>> odeget(options.'Rel')
ans =
```

```
1.0e-006*
1.0000 0.1000
» odegetOptions.'Abs')
ans =
0.0060
```

- `options=odeset('name1'.value1, 'name2'.value2,...)` — создает структуру параметров, в которой указанные свойства по имени 'name...' принимают следующие за ними значения. Вместо 'name...' можно ввести только первые буквы, которые однозначно определяют имя свойства (`abs` — `AbsTol`, `inaxit` — `maxiter` и т. д.);

- `options=odeset(ol dopts, newopts)` — изменяет существующую структуру параметров `oldopts` путем объединения ее с новой структурой `newopts`. Все новые параметры, не равные пустой матрице, заменяют соответствующие параметры в структуре `oldopts`;

- `options=odeset(ol dopts, 'name1'.value1,...)` — изменяет в существующей структуре параметров соответствующие значения. Пример:

```
oldopts
F 1 [ ] 4 'S' 'S' [ ] [ ] [ ]
newopts
T 3 F [ ] 'S' [ ] [ ] [ ]
odeset(oldopts.newopts)
T 3 F . 4 ' ' 'S' [ ] [ ] [ ]
```

Функция `odeset` без параметров возвращает все имена свойств и их допустимые значения.

Пример:

```
» odeset
AbsTol: [ positive scalar or vector {1e-6} ]
RelTol: [ positive scalar {1e-3} ]
NormControl: [ on | {off} ]
OutputFcn: [ function ]
OutputSel: [ vector of integers ]
Refine: [ positive integer ]
Stats: [ on | {off} ]
InitialStep: [ positive scalar ]
MaxStep: [ positive scalar ]
BDF: [ on | {off} ]
MaxOrder: [ 1 | 2 | 3 | 4 | {5} ]
Jacobian: [ matrix | function ]
JPattern: [ sparse matrix ]
Vectorized: [ on | {off} ]
```

```

Mass: [ matrix | function ] MStateOpendence: [
none | weak | strong ]
MvPattern: [ sparse matrix ]
MassSingular: [ yes | no | {maybe} ]
InitialSlope: [ vector ] Events: [ function ]

```

ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

Численное интегрирование традиционно является одной из важнейших сфер применения математического аппарата. В данном разделе приводятся функции для численного интегрирования различными методами. Численное интегрирование заключается в приближенном вычислении определенного интеграла вида

$$\int_a^b y(x) dx$$

Одним из многочисленных численных методов, для вычисления неопределенного интеграла по рекурсивному алгоритму усредняются значения $b=a+dn$, где d — предельно малая величина.

Метод трапеций

Приведенные ниже функции выполняют численное интегрирование методом *трапеций* и методом *трапеций с накоплением*.

- `trapz(Y)` — возвращает определенный интеграл, используя интегрирование методом трапеций с единичным шагом между отсчетами. Если Y — вектор, то `trapz(Y)` возвращает интеграл элементов вектора Y , если Y — матрица, то `trapz(Y)` возвращает вектор-строку, содержащую интегралы каждого столбца этой матрицы;
- `trapz(X,Y)` — возвращает интеграл от функции Y по переменной X , используя метод трапеций (пределы интегрирования в этом случае задаются начальным и конечным элементами вектора X);
- `trapz(...,dim)` — возвращает интеграл по строкам или по столбцам для входной матрицы в зависимости от значения переменной `dim`.

Примеры:

```

»Y=[1.2.3.4]
Y =
1 2 3 4
» trapz(y)
ans =

```

```
7.5000
» X=0:pi/70:pi/2;
» Y=cos(X);
» Z = trapz(Y)
Z =
22.2780
```

• `cumtrapz(Y)` — возвращает численное значение определенного интеграла для функции, заданной ординатами в векторе или матрице `Y` с шагом интегрирования, равным единице (интегрирование методом трапеций с накоплением). В случае когда шаг отличен от единицы, но постоянен, вычисленный интеграл достаточно умножить на величину шага. Для векторов эта функция возвращает вектор, содержащий результат интегрирования с накоплением элементов вектора `Y`. Для матриц — возвращает матрицу того же размера, что и `Y`, содержащую результаты интегрирования с накоплением для каждого столбца матрицы `Y`;

• `cumtrapz(X, Y)` — выполняет интегрирование с накоплением от `Y` по переменной `X`, используя метод трапеций. `X` и `Y` должны быть векторами одной и той же длины или `X` должен быть вектором-столбцом, а `Y` — матрицей;

• `cumtrapz(...,dim)` — выполняет интегрирование с накоплением элементов по размерности, точно определенной скаляром `dim`. Длина вектора `X` должна быть равна `size(Y,dim)`.

Примеры:

```
» cumtrapz(y)
ans=
0 1.5000 4.0000 7.5000
» Y=magic(4)
Y =
162 3 13
5 11 10 8
97 6 12
4 14 15 1
» Z= cumtrapz(Y,1)
Z =
0 0 0 0
10.5000 6.5000 6.5000 10.5000
17.5000 15.5000 14.5000 20.5000
24.0000 26.0000 25.0000 27.0000
```

Численное интегрирование методом квадратур

Приведенные ниже функции осуществляют интегрирование и двойное интегрирование, используя квадратурную формулу Симпсона или метод Гаусса-Лобатто. Квадратура — численный метод нахождения площади под графиком функции/(т), т. е. вычисление определенного интеграла вида

$$\int_a^b y(x) dx$$

В приведенных ниже формулах подынтегральное выражение `fun` обычно задается в форме дескриптора функции, поэтому с дидактическими целями используем нотацию `@fun`.

Функции `quad` и `quadl` используют два различных алгоритма квадратуры для вычисления определенного интеграла. Функция `quad` выполняет интегрирование по методу низкого порядка, используя рекурсивное правило Симпсона [4, 40]. Но она может быть более эффективной при негладких подынтегральных функциях или при низкой требуемой точности вычислений. Алгоритм `quad` в MATLAB 6 изменен по сравнению с предшествовавшими версиями, точность по умолчанию по сравнению с версиями 5.3х повышена в 1000 раз (с 10^{-3} до 10^{-6}). Новая функция `quadl` (квадратура Лобатто) использует адаптивное правило квадратуры Гаусса—Лобатто очень высокого порядка. Устаревшая функция `quads` выполняла интегрирование, используя квадратурные формулы Ньютона—Котеса 8-го порядка [40]. Достигаемая точность интегрирования гладких функций в MATLAB 6 поэтому также значительно выше, чем в предшествующих версиях.

- `quad(@fun,a,b)` — возвращает численное значение определенного интеграла от заданной функции `@fun` на отрезке `[a b]`. Используется значительно усовершенствованный в MATLAB 6 адаптивный метод Симпсона;
- `quad(@fun,a,b,tol)` — возвращает численное значение определенного интеграла с заданной относительной погрешностью `tol`. По умолчанию `tol=1.e-6`. Можно также использовать вектор, состоящий из двух элементов `tol=[rel_tol abs_tol]`, чтобы точно определить комбинацию относительной и абсолютной погрешности;
- `quad(@fun,a,b,tol,trace)` — возвращает численное значение определенного интеграла и при значении `trace`, не равном нулю, строит график, показывающий ход вычисления интеграла;
- `quad(@fun,a,b,tol,trace,P1,P2,...)` — возвращает численное значение определенного интеграла по хот подынтегральной функции `fun`,

использует дополнительные аргументы P1, P2,..., которые напрямую передаются в подынтегральную функцию: $G=\text{fun}(X,P1,P2,...)$.

Примеры:

```
» quad(' (exp(x)-1) ' .0.1.1e-5)
ans =
0.7183
» q = quad(@exp.0.2.1e-4)
q =
6.3891
» q = quad(@sin.0,pi,1e-3)
q = 2.0000
```

• `dblquad(@fun,inmin,inmax,outmin,outmax)` — вычисляет и возвращает значение двойного интеграла для подынтегральной функции `fun` (`Inner`, `outer`), по умолчанию используя квадратурную функцию `quad`. `Inner` — внутренняя переменная, изменяющаяся на закрытом интервале от `inmin` до `inmax`, а `outer` — внешняя переменная, изменяющаяся на закрытом интервале от `outmin` до `outmax`. Первый аргумент `@fun` — строка, описывающая подынтегральную функцию. Это может быть либо дескриптор функции, либо объект `inline` (в последнем случае символ «@» в ее записи отсутствует). Обычная запись в апострофах теперь недопустима. Эта функция должна быть функцией двух переменных вида `fout=fun(inner,outer)`. Функция должна брать вектор `inner` и скаляр `outer` возвращать вектор `fout`, который является функцией, вычисленной в `outer` и каждом значении `inner` [Функция `inline('expr','arg1',...'argn')` так же создает объект, но без дескриптора, 'expr' — выражение Строки 'argx' — входные аргументы. При их отсутствии по умолчанию подставляется x. Если вместо 'arg' — скаляр, то он означает количество дополнительных переменных P. Примеры запис; `g = inline(exp); g - inline('t^2');`; `g = inline('sin(2*pi*f + theta)');`; `g = inline('sin(2*pi*f + theta)', : 'theta');`; `g - inline('x^P1+P2', 2).` - Примеч. ред.];

• `dblquad(@fun,inmin,inmax,outmin,oiitmax,tol .trace)` — передает в функцию `dblquad` параметры `tol` и `trace`. Смотрите справку по функции `quad` для получения информации о параметрах `tol` и `trace`;

• `dblquad(@fun,inmin,inmax,outmin,outmax.tol .trace,order)` — передает параметры `tol` и `trace` для функции `quad` или `quadl` в зависимости от значения строки `order`. Допустимые значения для параметра `order` — `@quad`, `@quadl` или имя любого определенного пользователем квадратурного метода с таким же вызовом и такими же возвращаемыми параметрами, как у функций `quad` и `quadl`. (Например, при проверке старых программ можно использовать `@quad8` для большей совместимо-

сти с прежними версиями MATLAB). По умолчанию (без параметра `order`) вызывается `@quad`, поскольку подинтегральные функции могут быть негладкими.

Пример: пусть `m`-файл `integr.m` описывает функцию $2*y*\sin(x)+x/2*\cos(y)$, тогда вычислить двойной интеграл от той функции можно следующим образом:

```
» result = dblquad(@integr,pi,2*pi,0.2*pi)
result =
78.9574
```

РАБОТА С ПОЛИНОМАМИ

Полиномы (у нас их принято называть также *степенными многочленами*) — широко известный объект математических вычислений и обработки данных. Обычно полином записывается в виде

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0,$$

и так обычно принято в MATLAB для n , отрицательных по умолчанию, хотя и возможны иные формы записи, например

$$p(x) = a_1 x^n + a_2 x^{n-1} + \dots + a_n x + a_{n+1}$$

Последняя запись обычно принята в MATLAB для n , по умолчанию положительных.

Широкое применение полиномов отчасти обусловлено большими возможностями полиномов в представлении данных, а также их простотой и единообразием вычислений. В этом разделе описаны основные функции для работы с полиномами. При этом полиномы обычно задаются векторами их коэффициентов.

Умножение и деление полиномов

Ниже приведены функции, осуществляющие умножение и деление полиномов или, что то же самое, свертку двух входных векторов, в которых находятся коэффициенты полиномов, и операцию, обратную свертке.

- `w = conv(u,v)` — возвращает свертку векторов u и v . Алгебраически свертка — то же самое, что и произведение полиномов, чьи коэффициенты — элементы векторов u и v . Если длина вектора u равна m , а длина вектора v — n , то вектор w имеет длину $m+n-1$, а его k -й элемент вычисляется по следующей формуле :

$$w(k) = \sum_j u(j) \cdot v(k+1-j)$$

Пример:

```
» f=[2.3.5.6]; d=[7,8,3]; r=conv(f,d)
```

```
r =
```

```
14 37 65 91 63 18
```

• `[q,r]=deconv(v,u)` —возвращает результат деления полинома v на полином u . Вектор q представляет собой частное от деления, а r — остаток от деления, так что выполняется соотношение $v=\text{conv}(u,q)+r$.

Пример:

```
»
```

```
t=[14,37.65.91,63,18]; r=[7.8.3]; [w,e]=deconv(t,r)
```

```
w =
```

```
2.0000 3.0000 5.0000 6.0000
```

```
e =
```

```
1.0e-013
```

```
0 0 0.1421 -0.1421-0.2132-0.1066
```

Вычисление полиномов

В этом разделе приведены функции вычисления коэффициентов характеристического полинома, значения полинома в точке и матричного полинома:

• `poly(A)` — для квадратной матрицы A размера $n \times n$ возвращает вектор-строку размером $n+1$, элементы которой являются коэффициентами характеристического полинома $\det(A-sI)$, где I — единичная матрица, as — оператор Лапласа. Коэффициенты упорядочены по убыванию степеней. Если вектор состоит из $n+1$ компонентов, то ему соответствует полином вида $c_1 s^n + \dots + c_n s + c_{n+1}$;

• `poly(r)` — для вектора r возвращает вектор-строку p с элементами, представляющими собой коэффициенты полинома, корнями которого являются элементы вектора r . Функция `roots(p)` является обратной, ее результаты, умноженные на целое число, дают `poly(r)`.

```
A =
```

```
2 3 6
```

```
3 8 6
```

```
1 7 4
```

```
» d=poly(A)
```

```
d =
```

```
1.0000 -14.0000 -1.0000-40.0000
```

```
» A=[3,6.8:12.23.5:11.12.32]
```

```
A =
```

```
3 6 8
```

```
1223 5
```

```
1112 32
```

```
» poly(A)
```

```
ans =
```

```
1.0000 -58.0000 681.0000 818.0000
```

Приведенная ниже функция вычисляет корни (в том числе комплексные) для полинома вида

$$C_1 S^n + \dots + C_n S + C_{n+1}$$

- `roots(c)` — возвращает вектор-столбец, чьи элементы являются корнями полинома `c`.

Вектор-строка `c` содержит коэффициенты полинома, упорядоченные по убыванию степеней. Если `c` имеет $n+1$ компонентов, то полином, представленный этим вектором, имеет вид

$$C_1 S^n + \dots + C_n S + C_{n+1}$$

Пример:

```
» x=[7.45.12.23];d=roots(x)
```

```
d =
```

```
-6.2382
```

```
-0.0952+0.7195i
```

```
-0.0952 -0.7195i
```

```
A=[-6.2382 -0.0952+0.7195i -0.0952 -0.7195i]:
```

```
B=Poly(A)
```

```
B=[1.0000 6.4286 1.7145 3.2859]
```

```
B*7
```

```
ans =
```

```
7.0000 45.000212.001523.0013
```

С погрешностью округления получили тот же вектор.

- `polyval(p,x)` — возвращает значения полинома `p`, вычисленные в точках, заданных в массиве `x`. Полином `p` — вектор, элементы которого являются коэффициентами полинома в порядке уменьшения степеней, `x` может быть матрицей или вектором. В любом случае

функция `polyval` вычисляет значения полинома `p` для каждого элемента `x`;

- `[y,delta] = polyval (p, x.S)` или `[y,delta] = polyval (p,x.S.mu)` — использует структуру `S`, возвращенную функцией `polyfit`, и данные о среднем значении (`mu(1)`) и стандартном отклонении (`mu(2)`) генеральной совокупности для оценки пр-грешности аппроксимации (`y+delta`).

Пример:

```
» p=[3,0.4.3]; d=polyval(p,[2,6])
d =
35 675
```

- `polyvalm(p,X)` — вычисляет значения полинома для матрицы. Это эквивалентно подстановке матрицы `X` в полином `p`. Полином `p` — вектор, чьи элементы являются коэффициентами полинома в порядке уменьшения степеней, а `X` — квадратная матрица.

Пример:

```
» D=pascal(5)
D =
1 1          1          1          1
1 2          3          4          5
1 3          6          10         15
1 4          10         20         35
1 5          15         35         70

f=poly(d)
f =
1.0000 -99.0000 626.0000 -626.0000 99.0000-
1.0000
» polyvalm(f,D)
ans =
1.0e-006*
-0.0003 -0.0011-0.0038-0.0059-0.0162
-0.0012 -0.0048-0.0163-0.0253-0.0692
-0.0034 -0.0131 -0.0447 -0.0696 -0.1897
-0.0076 -0.0288-0.0983-0.1529-0.4169
-0.0145-0.0551-0.1883-0.2929-0.7984
```

Данный пример иллюстрирует также погрешности численных методов, поскольку точное решение дает нулевую матрицу.

Вычисление производной полинома

Ниже приведена функция, возвращающая производную полинома p :

- `polyder(p)` — возвращает производную полинома p ;
- `polyder(a,b)` — возвращает производную от произведения полиномов a и b ;
- `[q,d]=polyder(b,a)` — возвращает числитель q и знаменатель d производной от отношения полиномов b/a .

Примеры:

```
» a=[3.5.8]:b=[5,3.8]:dp=polyder(a)
dp =
6 5
» dt=polyder(a,b)
dt =
60102 158 64
» [q,p]=polyder(b,a)
q =
1632 -16
p =
9 30 73 80 64
```

Решение полиномиальных матричных уравнений.

Приведенная ниже функция вычисляет собственные значения матричного полинома.

- `[X,e] = polyeig(A0,A1,...,Ap)` — решает задачу собственных значений для матричного полинома степени p вида:

$$(A_0 \cdot \lambda + A_1 \cdot \lambda^2 + \dots + A_p \cdot \lambda^{p+1}) \cdot x = 0$$

где степень полинома p — целое неотрицательное число, а $A_0, A_1, \dots, A_p$ — входные матрицы порядка n . Выходная матрица X размера $n \times n$ содержит собственные векторы в столбцах. Вектор e размером n содержит собственные значения.

Пример:

```
» A=[1:4:5:8:9:12:13:16]
A =

1 2          3 4
5 6          7 8
9 10         11 12
13 14        15 16
» B=[4:7 ;2:5;10:13;23:26]
```

```

3 -
4 5          6 7
2 3          4 5
1011         12 13
2324         25 26
» [F,a]=      polyeig(A,B)
F =
0.4373      0.0689      -0.5426      -0.7594
-0.3372     -0.4969      0.6675      -0.1314
-0.6375     0.7870      0.2927      -0.1314
0.5374     -0.3591      - 0.4176      0.3771
a =
4.4048
0.4425
-0.3229
-1.0000

```

Разложение на простые дроби

Для отношения полиномов b и a используются следующие функции:

- $[r,p,k] = \text{res I due } (b, a)$ — возвращает вычеты, полюса и многочлен целой части отношения двух полиномов $b(s)$ и $a(s)$ в виде

$$\frac{b(s)}{a(s)} := \frac{b_1 + \dots + b_m \cdot s - m}{a_1 + \dots + a_{n+1} \cdot s^{-n}}$$

- $[b,a] = \text{residue}(r,p,k)$ — выполняет обратную свертку суммы простых дробей (см. более подробное описание в справочной системе) в пару полиномов с коэффициентами в векторах b и a .

Пример:

```

» b=[4.3.1]:a=[1.3.7.1]:[r,p,k]=residue(b,a)
r=
1.9484 + 0.8064i
1.9484 - 0.8064i
0.1033
P =
-1.4239 + 2.1305i
-1.4239 - 2.1305i
-0.1523

```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Самарский А. А. Численные методы: Учеб. пособие для вузов. – М.: Наука, 1989.
2. Турчак Л.И. Основы численных методов: Учеб. пособие для вузов. – М.: Наука, 1987.
3. Яблочкин Л.Б. и др. Основы численных методов: Учеб. пособие. – Тула: ТГУ, 2000.
4. Раkitин В. И., Первушин В.Е. Практическое руководство по методам вычислений с приложением программ для персональных компьютеров: Учеб. пособие. – М.: Высшая школа, 1998.
5. Васильков Ю.И., Василькова Н.Н. Компьютерные технологии вычислений в математическом моделировании. Учеб. пособие. – М.: Финансы и статистика, 1999.
6. Дьяконов В. MATLAB: учебный курс. – СПб.: Питер, 2001.
7. Мартынов Н.Н., Иванов А.П. MATLAB 5X. Вычисления, визуализация, программирование. – М.: КУДИЦ-ОБРАЗ, 2000.
8. Гуляев А. Визуальное моделирование в среде MATLAB: учебный курс – СПб.: Питер, 2000.
9. Потемкин В.Г. Инструментальные средства MATLAB 5X. – М.: ДИАЛОГ-МИФИ, 2000.
10. Потемкин В.Г., Рудаков П.И. Система MATLAB 5 для студентов. Учебно-справ. изд. - М.: ДИАЛОГ-МИФИ, 1999.
11. Соболев М.И. Метод Монте-Карло. – М.: Наука, 1972.
12. Гутер Р.С. и др. Задачник-практикум по программированию и вычислительной математике. – М.: Наука, 1973.
13. Мостеллер Ф. Пятьдесят занимательных вероятностных задач с решениями. – М.: Наука, 1971.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС МАТЛАВ.....	5
РАБОТА С ОКНОМ СПРАВКА.....	12
ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЙ.....	14
СОХРАНЕНИЕ РАБОЧЕЙ ОБЛАСТИ СЕССИИ.....	18
РАБОТА С ЭЛЕМЕНТАРНЫМИ ФУНКЦИЯМИ.....	19
РАБОТА С ВЕКТОРОМ, МАТРИЦЕЙ И МНОГОМЕРНЫМ МАССИВОМ	29
ОПЕРАЦИИ С ВЕКТОРАМИ И МАТРИЦАМИ.....	34
СТАТИСТИЧЕСКАЯ ОБРАБОТКА.....	41
ФУНКЦИИ СОРТИРОВКИ ЭЛЕМЕНТОВ МАССИВА.....	44
ВЫЧИСЛЕНИЕ КОЭФФИЦИЕНТОВ КОРРЕЛЯЦИИ.....	46
РАБОТА С ГРАФИКАМИ.....	48
ЧИСЛЕННЫЕ МЕТОДЫ.....	78
ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ.....	97
РАБОТА С ПОЛИНОМАМИ.....	101
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	107

Учебное издание

Киселева Ольга Владимировна

КОМПЬЮТЕРНЫЕ ВЫЧИСЛЕНИЯ

Учебное пособие

Зав. РО

З.А. Губайдулина

Редактор

С.Н. Яниева

Техн.редактор

К.Ж. Кошербаева

Компьютерная верстка

Д. Ш. Тажиева

Подписано в печать

17.03.2005 г.

Тираж 500 экз. Формат 60 x 84 1/16. Бумага типографская.

Объем 6,2 уч.-изд.л. Заказ № Цена договорная

Издание Казахского национального технического университета
им. К.И. Сатпаева

Издательский центр КазНТУ

Алматы, ул.Ладыгина, 32