

ГОСУДАРСТВЕННЫЙ ИНЖЕНЕРНЫЙ УНИВЕРСИТЕТ
АРМЕНИИ

MATLAB

УЧЕБНОЕ ПОСОБИЕ

ГАСПАРЯН Олег Николаевич
Д.т.н, с.н.с

2005

СОДЕРЖАНИЕ

Система математических расчетов MATLAB	4
Рабочий стол (desktop) системы MATLAB	5
Общие свойства и возможности рабочего стола MATLAB	5
Получение справок (Getting Help)	7
Рабочее пространство системы MATLAB	13
Просмотр и редактирование массивов данных при помощи редактора Array Editor	13
Пути доступа системы MATLAB	16
Операции с файлами	18
Дуальность (двойственность) команд и функций	20
Действия над матрицами в системе MATLAB	23
Двоеточие (Colon)	24
Решение систем линейных уравнений	28
Степени матриц и матричные экспоненты	30
Собственные значения и собственные векторы	35
Полиномы и интерполяция	37
Анализ данных и статистика	41
Многомерные Массивы	50
Создание Многомерных Массивов	65
Организация и хранение данных	66
Массивы структур	74
Массивы ячеек	75
Программирование на MATLAB-е	86
Типы данных	97
Команды управления данными (Flow Control)	108
<u>Приложение 1.</u> Тематические направления системы MATLAB	114
<u>Приложение 2.</u> Команды общего назначения (General purpose commands)	121
<u>Приложение 3.</u> Операторы и специальные символы	123
<u>Приложение 4.</u> Элементарные математические функции (Elementary math functions).	126
<u>Приложение 5.</u> Элементарные матрицы и операции над ними (Elementary matrices and matrix manipulation)	127
<u>Приложение 6.</u> - Матричные функции и линейная алгебра (Matrix functions - numerical linear algebra)	129
<u>Приложение 7.</u> Полиномы и интерполяция (Interpolation and polynomials)	132
<u>Приложение 8.</u> Анализ данных и преобразование Фурье (Data analysis and Fourier transforms)	133
<u>Приложение 9.</u> Функции обработки символьных строк (Character strings)	135

Система математических расчетов MATLAB

Система MATLAB (сокращение от MATrix LABoratory - МАТричная Лаборатория) разработана фирмой The MathWorks, Inc. (США, г.Нейтик, шт. Массачусетс) и является интерактивной системой для выполнения инженерных и научных расчетов, которая ориентирована на работу с массивами данных. Система использует математический сопроцессор и допускает обращения к программам, написанным на языках Fortran, C и C⁺⁺.

Наиболее известные области применения системы MATLAB:


- математика и вычисления;
- разработка алгоритмов;
- вычислительный эксперимент, имитационное моделирование;
- анализ данных, исследование и визуализация результатов;
- научная и инженерная графика;
- разработка приложений, включая графический интерфейс пользователя.

MATLAB – это интерактивная система, основным объектом которой является массив, для которого не требуется указывать размерность явно. Это позволяет решать многие вычислительные задачи, связанные с векторно-матричными формулировками, существенно сокращая время, необходимое для программирования на скалярных языках типа Fortran или C. Будучи ориентированной на работу с реальными данными, эта система выполняет все вычисления в арифметике с плавающей точкой, в отличие от систем компьютерной алгебры REDUCE, MACSYMA, DERIVE, Maple, Mathematica, Theorist, где преобладает целочисленное представление и символьная обработка данных.

Система MATLAB – это одновременно и операционная среда и язык программирования. Одна из наиболее сильных сторон системы состоит в том, что на языке MATLAB могут быть написаны программы для многократного использования. Пользователь может сам написать специализированные функции и программы, которые оформляются в виде М-файлов. По мере увеличения количества созданных программ возникают проблемы их классификации и тогда можно попытаться собрать родственные функции в специальные папки. Это приводит к концепции пакетов прикладных программ (Application Toolboxes или просто Toolboxes), которые представляют собой коллекции М-файлов для решения определенной задачи или проблемы.

В действительности Toolboxes – это нечто большее, чем просто набор полезных функций; часто это результат работы многих исследователей по всему миру, которые объединяются в группы по самым различным интересам, начиная от нейтронных сетей, дифференциальных уравнений в частных производных, сплайн-аппроксимации, статистики и размытых множеств до проектирования робастных систем управления, теории сигналов, идентификации, а также моделирования линейных и нелинейных динамических систем с помощью исключительно эффективного пакета SIMULINK. Именно поэтому пакеты прикладных программ MATLAB Application Toolboxes, входящие в состав семейства продуктов MATLAB, позволяют находиться на уровне самых современных мировых достижений в разных областях науки и техники.

Вызов и выход из MATLAB

Вызов MATLAB-а. Для вызова системы MATLAB требуется двойное нажатие на иконку  в рабочем столе Windows. При инсталляции MATLAB-а стартовой директорией по умолчанию является **\$matlabroot\work**, где **\$matlabroot** есть директория, где установлены файлы системы MATLAB.

При вызове, система MATLAB автоматически выполняет главный М-файл (master M-file) **matlabrc.m.**, и файл **startup.m** (если последний существует). Файл **matlabrc.m**, которые расположен в директории **local**, зарезервирован фирмой The MathWorks, а в многопользовательских системах может быть использован также системным менеджером. Файл **startup.m** предназначен для задания ряда стартовых опций (возможностей) по усмотрению пользователя. Вы можете изменить исходные пути доступа (см. далее), ввести заранее определенные переменные в рабочее пространство, изменить текущую директорию и т.д. Стартовый файл **startup.m** следует ввести в директорию **\$matlabroot\toolbox\local** (более подробно с данным вопросом можно ознакомиться в справочных пособиях по MATLAB-y).

Выход из MATLAB-а. Для окончания сеанса работы с MATLAB следует выбрать опцию **Exit MATLAB** (Выход из MATLAB) в меню **File** на рабочем столе MATLAB-а, или напечатать **quit** (Выход) в командном окне **Command Window**.

При выходе, MATLAB выполняет специальный файл **finish.m**, относящийся к типу сценариев (см. далее), если только данный файл существует в текущей директории или где-либо на пути доступа системы MATLAB. Файл **finish.m** создается пользователем. Он должен содержать функции или операции, которые пользователь желает автоматически выполнить при выходе из системы MATLAB, например, такие как сохранение рабочего пространства или вызов диалогового окна, запрашивающего подтверждения выхода. В указанной выше директории **\$matlabroot\toolbox\local** имеются два файла, которые пользователь может использовать в качестве образца при создании своего файла **finish.m**:

- **finishsav.m** – Включает функцию **save**, что приводит к автоматическому запоминанию рабочего пространства при выходе из MATLAB-а.
- **finishdlg.m** – Выводит на экран подтверждающее диалоговое окно, которое позволяет аннулировать выход.


Рабочий стол (desktop) системы MATLAB

Рабочий стол системы MATLAB содержит следующие инструментальные окна, часть из которых не появляется при начальном запуске:

- **Command Window** (Командное Окно) – Выполняет все функции и команды системы MATLAB.
- **Command History** (История Команд) – Просмотр функций, введенных ранее в **Command Window**, их копирование и выполнение.
- **Launch Pad** (Окно Запуска) – Запускает все инструменты и обеспечивает доступ ко всем пакетам системы MATLAB.
- **Current Directory Browser** (Окно Просмотра Текущего Каталога) – Просмотр файлов MATLAB, а также сопутствующих файлов, а также выполнение таких операций над файлами, как поиск и открытие файлов.
- **Help Browser** (Окно Просмотра Помощи) – Поиск и просмотр документации по всем функциям и средствам системы MATLAB.
- **Workspace Browser** (Окно Просмотра Рабочего Пространства) – Просмотр и изменение

содержания рабочего пространства (workspace) системы MATLAB.

- **Array Editor** (Редактор Массивов Данных) – Просмотр содержимого массивов данных, записанных в виде таблицы и редактирование данных.
- **Editor/Debugger** (Редактор/Отладчик) – Для создания, редактирования и отладки М-файлов, т.е. файлов, содержащих функции системы MATLAB.

Общий вид рабочего окна MATLAB представлен ниже (рис.1). Каждое из перечисленных окон может быть выведено из конфигурации рабочего стола нажатием кнопки со стрелкой  в верхнем правом углу окна (см. рис. 1). Обратная операция, то есть ввод в общую конфигурацию, осуществляется выбором опции **Dock** в меню **View** соответствующего окна. Можно также изменить конфигурацию рабочего стола путем перемещения любого открытого окна в новое положение. Для этого нужно просто нажать левой клавишей мыши на выбранное название окна (**Title Bar**) и «перетащить» его в желаемое положение.

Для восстановления стандартной конфигурации рабочего стола MATLAB необходимо выбрать опцию **Default** (По Умолчанию) в подменю **Desktop Layout** (План Рабочего Стола) в меню **View** (Вид) любого открытого окна системы. Все окна MATLAB содержат также контекстное меню (context menu), которое вызывается нажатием правой кнопки мыши и содержит наиболее часто применяемые опции (функции), связанные с данным окном.

Таким образом, в системе MATLAB имеется возможность изменения вида рабочего стола путем открытия, закрытия, перемещения или изменения размеров каждого из индивидуальных окон.

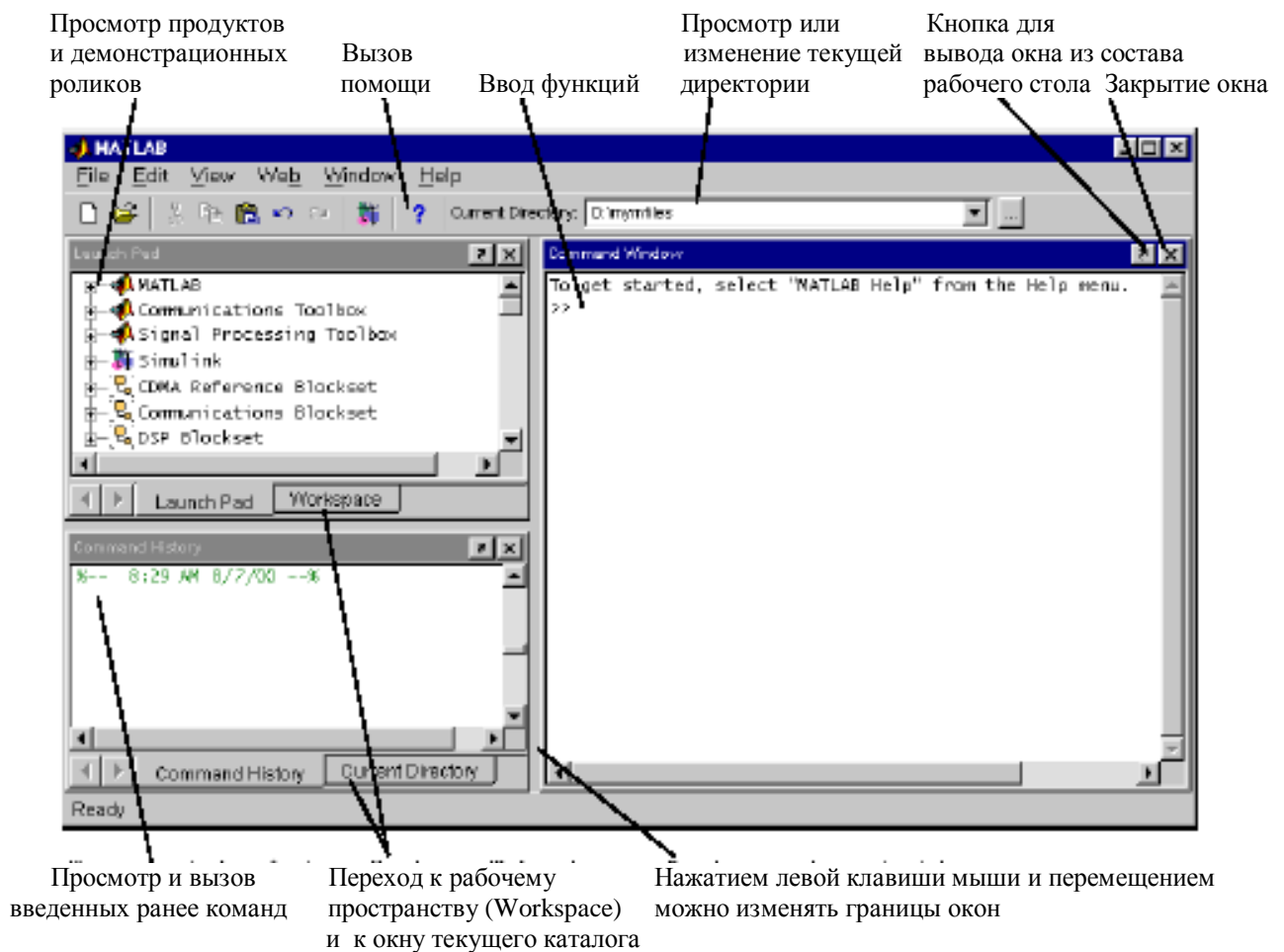
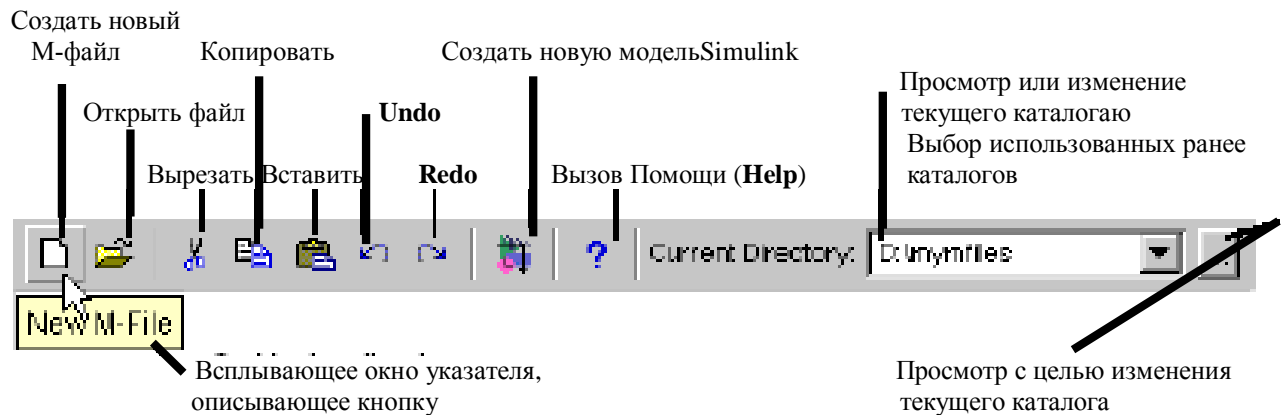


Рис. 1. Общий вид рабочего стола системы MATLAB

Общие свойства и возможности рабочего стола MATLAB

Ниже приводится вид инструментальной панели системы MATLAB и указано назначение основных кнопок.



Undo – отмена последнего действия; **Redo** – повторение последнего действия.

Рис. 2. Инструментальная линейка (Toolbar) рабочего стола

Command Window (Командное окно)

1. Выполнение функций и ввод переменных

Приглашение к вводу команды (`>>`) в **Command Window** означает, что MATLAB готов к приему. При появлении приглашения `>>` вы можете ввести переменную или выполнить команду. Например, для создания 3x3 матрицы **A** следует напечатать

```
A = [1 2 3; 4 5 6; 7 8 10]
```

При нажатии клавиш **Enter** (или **Return**) после набора строки, MATLAB реагирует выводом следующей записи

```
A =  
    1    2    3  
    4    5    6  
    7    8   10.
```

При этом переменная (матрица) запоминается в рабочем пространстве (workspace) MATLAB.

Внимание! Система MATLAB чувствительна к выбору регистра, т.е. MATLAB различает переменные `a` и `A` ! При написании команды (функции) также не следует использовать заглавные буквы !

Для выполнения функций следует напечатать функцию включая все аргументы и нажать **Enter**. При этом MATLAB отобразит в командном окне результат. Например, напечатав

```
magic(2)
```

получим

```
ans =  
    1    3  
    4    2.
```

Если вы хотите ввести несколько командных строк без их немедленного выполнения, воспользуйтесь комбинациями **Shift+Enter** после ввода каждой строки. Последующее нажатие **Enter** приведет к выполнению всех введенных строк.

Приглашение **K>>** в командном окне означает, что MATLAB находится в режиме отладки (debug mode). Данный режим будет подробно рассмотрен в дальнейшем.

2. Вычисление выделенных функций и выражений (Evaluating a Selection)

Для вычисления выделенных выражений в командном окне следует нажать правую кнопку мыши и выбрать в контекстном меню опцию **Evaluate Selection**. Данная операция невозможна если система MATLAB занята, например, выполняет М-файл.

3. Открытие выделения (Opening a Selection)

Для открытия М-файла некоторой функции следует выделить эту функцию в командном окне и выбрать в контекстном меню опцию **Open Selection**. Это приведет к открытию данного файла в окне Редактора/Отладчика (**Editor/Debugger**).

Внимание! Вы можете одновременно выполнять на MATLAB-е только одну функцию. Если MATLAB выполняет некоторую команду, то все последующие введенные команды запоминаются и выполняются только при окончании предыдущей !

4. Ввод нескольких функций в одну строку

Для ввода нескольких функций в одну командную строку, их нужно разделить точкой с запятой (;). Например, запись трех следующих функций в одной командной строке

```
format short; x = (1:10)'; logs = [x log10(x)]
```

и нажатие **Enter** приведет к выполнению этих функций слева направо и к распечатке таблицы десятичных логарифмов в пределах от 1 до 10.

5. Ввод длинных функций

Если запись не помещается на одной строке, следует использовать три точки, (...) для обозначения того, что запись будет продолжена на следующей строке; нажать **Enter** для перехода к следующей строке, и затем продолжить ввод записи. Например,

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

Для записей с одинарными кавычками, такими как строки символов, кавычки следует ставить на каждой строке. Например, вводя следующую длинную строку

```
headers = ['Author Last Name, Author First Name, ' ...  
           'Author Middle Initial']
```

получим

```
headers =  
Author Last Name, Author First Name, Author Middle Initial
```


Максимальное количество символов в одной строке равно 4096.

6. Окраска и выделение синтаксиса

Некоторые записи появляются в различных цветах с целью облегчить поиск элементов, таких, например, как парных сочетаний **if/else**. Имеются следующие основные варианты:

- При записи строки с начальной кавычкой, она окрашена в лиловый цвет. Когда вы закрываете кавычку, строка становится коричнево-красной.
- При написании ключевых слов, таких как зарезервированных для программирования (например, **for**, **else**, **while** и т.д.), а также троеточия (...), они окрашиваются в синий цвет. Слова, записанные между зарезервированными приобретают структурированный вид, т.е. имеют отступы, облегчающие чтение текста.
- Двойное нажатие на начальную или конечную скобку (например, обычную, квадратную [], или фигурную { }) приводит к селекции (окрашиванию) всех символов между данной скобкой и ее парой. При введении конечной скобки происходит кратковременное высвечивание соответствующей начальной.
- При напечатании символа процента (%), который в системе MATLAB является признаком начала строки комментария (эти строки не выполняются), соответствующая строка окрашивается в зеленый цвет.
- При вводе команды обращения к операционной системе, которые начинаются с восклицательного знака (!), строка окрашивается в золотой цвет.
- Сообщения об ошибках имеют красный цвет.

Можно изменить приведенную стандартную окраску, введя соответствующие изменения в опции **Preferences** в меню рабочего стола **File**. Там же предусмотрена возможность выбора шрифта и размера символов.

7. Редактирование командной строки

Ниже приводятся некоторые свойства системы MATLAB, дающие экономию времени при вводе:

Свойства буфера обмена. При работе в командном окне целесообразно использовать опции **Cut** (Вырезать), **Copy** (Копировать), **Paste** (Вставить), **Undo** (Отмена последнего действия), **Redo** (Повторение последнего действия) из меню **Edit** (Редактирование), или соответствующие кнопки инструментальной панели (см. рис. 2). Некоторые из этих опций доступны также из контекстного меню командного окна.

Свойства клавиши Tab (Табуляция). MATLAB автоматически завершает имя функции, переменной, названия файла или свойства дескриптора графического объекта (handle graphics property), если вы наберете соответствующие первые несколько букв и нажмете клавишу **Tab**. Если это однозначно определяемое имя, то оно будет автоматически завершено. Если же данные буквы входят в название нескольких функций, то повторное нажатие клавиши **Tab** вызовет список всех возможных функций. Например, напечатайте **cos** и нажмите **Tab**. Система MATLAB не отреагирует, что означает наличие многих имен начинающихся с **cos**. При повторном нажатии **Tab** MATLAB отобразит список всех имеющихся в наличии подходящих функций:

```
cos cosh costfun  
cos_tr cosint costs_march.
```

Вызов предыдущих строк. Используйте клавиши стрелок, табуляции и управления на клавиатуре для повторного вызова, редактирования и повторного использования функций, которые вы ввели ранее. Допустим, вы ввели по ошибке

$$\text{rho} = (1 + \text{sqt}(5))/2$$

При нажатии **Enter** MATLAB ответит:

Undefined function or variable 'sqt'.
(Неизвестная функция или переменная **sqt**),

поскольку вы неправильно ввели функцию **sqr**t (квадратный корень). Вместо того чтобы перепечатывать всю строку, можно нажать клавишу **↑**. Предыдущая строка будет повторно отображена. Используя клавишу **←** можно привести курсор в требуемое положение и добавить недостающую букву **r**. Повторные нажатия клавиши **↑** вызовут ранее введенные строки. Все вводимые вами функции запоминаются в буфере обмена. При этом можно воспользоваться свойством «интеллектуального повторного вызова» (*smart recall*) для повторного вызова ранее введенных функций, которое заключается в том, что достаточно набрать только первые несколько символов из названия требуемой функции. Например, напечатав буквы **plo** и нажав клавишу **↑** мы вызовем последнюю функцию, которая начинается с **plo**. Данное свойство чувствительно к выбору регистра.

Ниже приводится полный список клавиш, которые можно использовать в командном окне

Клавиши	Клавиши управления	Результат операции
↑	Ctrl+p	Вызов предыдущей строки
↓	Ctrl+n	Вызов <u>следующей</u> строки.
←	Ctrl+b	Переход на один символ <u>назад</u> .
→	Ctrl+f	Переход на один символ вперед.
Ctrl+ →	Ctrl+r	Переход на одно слово <u>направо</u> .
Ctrl+ ←	Ctrl+l	Переход на одно слово <u>влево</u> .
Home	Ctrl+a	Переход к началу строки.
End	Ctrl+e	Переход к концу строки.
Esc	Ctrl+u	Очистить строку.
Delete	Ctrl+d	Удалить символ после курсора.
Backspace	Ctrl+h	Удалить символ перед курсором.
	Ctrl+k	Удалить от курсора до конца строки.
Shift+home		Выделить до начала строки.
Shift+end		Выделить до конца строки.

8. Очистка командного окна

Для очистки командного окна следует выбрать опцию **Clear Command Window** из меню **Edit**. Эта операция не приводит к очистке рабочего пространства, а только удаляет все записи с экрана монитора. В дальнейшем вы можете использовать все свойства клавиш для повторного вызова введенных ранее функций.

Эквивалентная функция ! Для очистки командного окна можно воспользоваться функцией **clc**. Подобно **clc**, функция **home** переносит приглашение к вводу системы MATLAB (>>) к верхнему левому углу командного окна, но при этом содержимое всего окна не очищается и может быть прочитано стандартными приемами просмотра.

9. Подавление вывода результатов на экран

Если вы заканчиваете строку точкой с запятой (;), то при нажатии **Enter** MATLAB выполняет задачу (программу), но не выводит результаты на экран монитора. Это может быть особенно полезным при генерации больших матриц. Например, при вводе

A = magic(100);

и нажатии **Enter** MATLAB создает в рабочем пространстве матрицу **A** размера 100x100, но не выводит ее на экран.

10. Разбиение экранного вывода на страницы

Если выводимые результаты очень длинные и не помещаются в пределах экрана, то вывод может быть слишком быстрым для восприятия (то есть строки будут бежать очень быстро). В таких случаях можно воспользоваться функцией **more**. По умолчанию функция **more** блокирована (выключена). Если вы напечатаете **more on**, то MATLAB осуществляет вывод на экран постранично (по размеру экрана). После просмотра первой страницы следует нажать на одну из следующих клавиш

Клавиша	Действие
Enter	Переход к следующей строке
Пробел	Переход к следующей странице
q	Остановка вывода на экран

Постраничный вывод можно блокировать вводом функции **more off**.

11. Выбор формата и интервала между строками для числовых данных

По умолчанию, числовые данные в командном окне представляются как пятизначные числа с фиксированной запятой. Воспользовавшись опцией **Preferences** в меню **File** можно изменить формат вывода этих данных. При этом формат вывода действует только на экранное представление чисел, а не на саму процедуру вычислений или запоминания данных в MATLAB-е.

Эквивалентная функция ! Для выбора формата выводимых на экран числовых данных можно воспользоваться функцией **format**. Данная функция имеет силу только в процессе текущего сеанса работы, то есть при выходе из MATLAB-а ее действие аннулируется и восстанавливается стандартное представление данных.

Примеры форматов данных. Ниже даны несколько примеров различных форматов двумерного вектора

x = [4/3 1.2345e-6]

формат **short e**

1.3333e+000 1.2345e-006

формат **short**

1.3333 0.0000

формат +

++

Полное описание допустимых форматов дается в справках (Help) по данной функции. Дополнительные возможности контроля вывода дают функции **sprintf** и **fprintf**.

Выбор интервала между строками. Воспользовавшись опцией **Preferences** в меню **File** можно также контролировать промежуток между строками. Команда **format compact** подавляет пустые строки, что дает возможность обозрения большего количества информации в командном окне. Для возврата к пустым строкам, которые облегчают чтение и восприятие информации на экране, нужно воспользоваться командой **format loose**.

12. Распечатка содержания командного окна

Для распечатки содержания всего командного окна следует выбрать **Print** из меню **File**. Для распечатки только части текста нужно сперва выделить эту часть, и затем выбрать **Print Selection** в том же меню **File**.

13. Выполнение программ

Выполнение М-файлов. Для выполнения М-файлов, т.е. файлов которые содержат программы на языке MATLAB, следует воспользоваться процедурой, совершенно аналогичной процедуре выполнения любой другой стандартной функции MATLAB-а, т.е. необходимо напечатать имя М-файла в командном окне и нажать **Enter**. Для вывода на экран каждой функции в М-файле по мере ее исполнения можно использовать команду **echo**, т.е. при вводе этой команды MATLAB будет последовательно выводить на экран каждую функцию в исполняемом М-файле.

Прерывание выполнения программы. Вы можете прервать выполнение программы путем нажатия **Ctrl+c** или **Ctrl+Break** в любое время.

Выполнение внешних программ. Восклицательный знак **!** означает выход из оболочки MATLAB-а и передачу оставшейся части командной строки операционной системе. Данное свойство может быть полезным для вызова утилит или других программ без выхода из системы MATLAB. После выполнения утилит, операционная система возвращает управление системе MATLAB.

Открытие М-файла в окне Редактора/Отладчика. Для открытия М-файла следует выделить имя файла или функции в командном окне и затем, вызвав контекстное меню нажатием правой кнопки мыши, выбрать опцию **Open Selection**. Соответствующий М-файл будет открыт в окне Редактора/Отладчика (**Editor/Debugger**).

Анализ ошибок. Если при выполнении М-файла появляется сообщение об ошибке, то нужно подвести курсор к данному сообщению и нажать клавишу **Enter**. Произойдет открытие «нехорошего» М-файла в окне Редактора (**Editor**), причем файл будет «прокручен» до строки, содержащей ошибку.

Сохранение сеанса работы. Для сохранения в памяти сеанса работы в системе MATLAB предусмотрена специальная функция **diary** (Дневник). Эта команда создает копию вашего сеанса работы в специальном файле на диске, включая все команды ввода и отклики системы MATLAB, но исключая графики. Вы можете затем просматривать и редактировать полученный текстовый файл используя любой текстовый редактор. Например, для создания на вашем диске файла, названного допустим **sept23** («23 сентября»), который содержал бы все введенные вами в этот день функции и отклики системы MATLAB, следует введя командное окно функцию **diary('sept23.out')**. Для прекращения записи сеанса вводится : **diary('off')**

Примечание. В окне **Command History** (История Команд) содержится запись всех функций, выполненных в текущем и предыдущий сеансах.

Получение справок (Getting Help)

Система MATLAB обеспечивает исключительно широкие возможности для получения справок по всем командам и функциям. Однако, к сожалению, все это доступно только на английском языке и поэтому их использование может вызвать определенные затруднения у неподготовленного потребителя. Среди важных и полезных команд, которые обеспечивают быстрый доступ к справкам можно выделить три: **help**, **helpwin** и **lookfor**.

1. HELP. Оперативная справка, отображающая текст в командном окне.

Команда **help**, сама по себе, выводит на экран (в командную строку) все тематические направления системы MATLAB. Каждое тематическое направление соответствует имени определенного каталога (директории) в MATLAB-е (см. Приложение 1).

Команда **help('topic')** или **help topic**, где **topic** есть определенная директория (например, **matlab\polyfun**), выводит список всех функций в данном каталоге.

Команда **help fun**, где **fun** – имя функции, выводит на экран справку по данной функции, как она записана в соответствующем М-файле.

2. HELPWIN. Обеспечивает те же функции, что и **help**, но справка выводится в Окно Просмотра Помощи (**Help Browser**) (см. рис. 1).

3. LOOKFOR. Осуществляет поиск в первой строке комментариев по всем М-файлам по заданному ключевому слову. Так, например, **lookfor XYZ** осуществляет поиск слова **XYZ** в первой строке текста справки (HELP text) во всех М-файлах на так называемом пути доступа MATLAB (**MATLABPATH**). Для всех файлов где встречается это слово, команда **lookfor** выводит на экран названия файлов и первые строки комментариев.

Рабочее пространство системы MATLAB

Рабочее пространство (**workspace**) MATLAB-а состоит из множества переменных (называемых массивами (**array**)), созданных во время сеанса работы системы MATLAB и запомненных в памяти. Можно добавлять новые переменные в рабочее пространство путем использования функций, выполняющих М-файлы, загрузкой запомненных ранее рабочих пространств, или же путем непосредственного ввода переменных. Так, например, если вы напечатаете в командном окне:

t = 0 : pi/4 : 2*pi; y = sin(t); z = 5;

то рабочее пространство будет содержать две переменные **y** и **t**, каждая из которых имеет девять значений, и одну скалярную переменную **z**.

.

Окно Просмотра Рабочего Пространства (Workspace Browser)

Для выполнения операций просмотра и изменения содержимого рабочего пространства удобно использовать Окно Просмотра Рабочего Пространства (ОПРП), хотя многие используемые им функции также доступны и из командной строки, путем применения соответствующих команд. Для открытия ОПРП следует проделать одно из следующих действий:

- Из меню **View** рабочего стола MATLAB выбрать **Workspace**.
- В Окне Запуска (**Launch Pad**) нажать дважды на **Workspace**.

- Ввести команду **workspace** из командной строки MATLAB-а.
- При этом получаем следующее окно (для некоторого конкретного набора переменных):

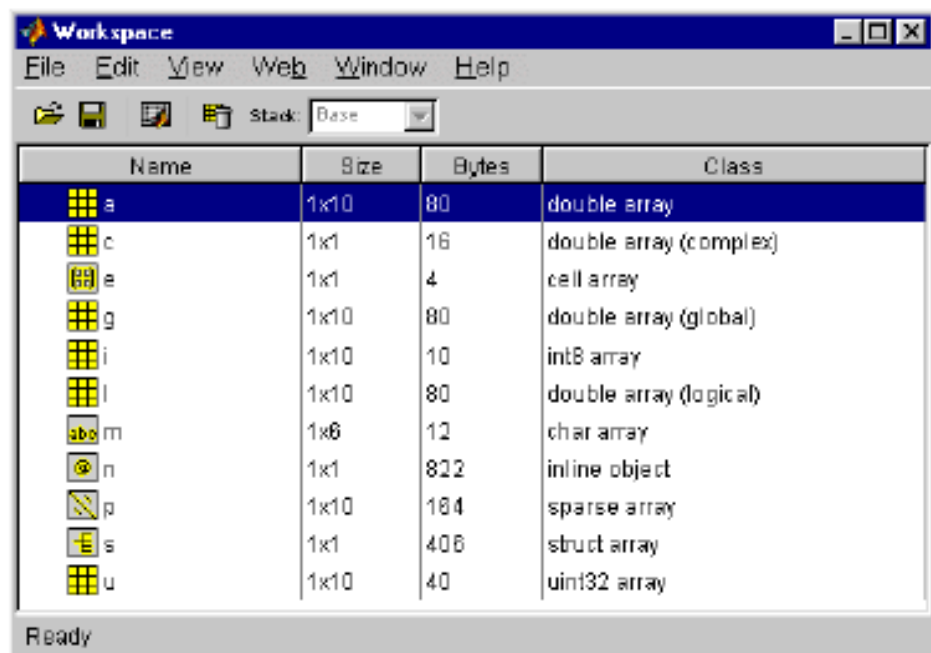


Рис. 3. Общий вид Окна Просмотра Рабочего Пространства

Просмотр текущего рабочего пространства. ОПРП показывает имя каждой переменной, размерность ее массива, размер в байтах и ее класс. Иконка в поле **Name** перед именем каждой переменной указывает на ее класс (классы переменных будут рассмотрены позднее).

Эквивалентные функции ! Команда **who** выводит в командное окно список всех переменных текущего рабочего пространства. Команда **whos** добавляет к списку переменных также информацию об их размерах и классе. Например, при вводе **who** система MATLAB отвечает:

Your variables are (ваши переменными являются):
A M S v

а при вводе **whos** имеем:


Name	Size	Bytes	Class
A	4x4	128	double array
M	8x1	2368	cell array
S	1x1	398	struct array
v	5x9	90	char array
Grand total is 286 elements using 2984 bytes			

где в последней строке указано общее число элементов и количество используемых байтов, и присутствуют следующие классы : double array – массив чисел удвоенной точности, cell array – массив ячеек, struct array – массив структур, char array – массив символов.

Сохранение текущего рабочего пространства. Рабочее пространство не сохраняется между отдельными сеансами работы системы MATLAB. Когда вы выходите из MATLAB-а, его рабочее пространство очищается. Вы можете сохранить все или часть переменных текущего рабочего пространства в так называемых MAT-файлах (MAT-file), которые являются специ-

альными бинарными (двоичными) файлами данных. В дальнейшем вы можете загрузить запомненные МАТ-файлы, как в течении того же сеанса, так и в последующих сеансах, для повторного использования запомненных переменных.

Сохранение всех переменных. Для сохранения всех переменных рабочего пространства с использованием ОПРП следует:

1. Из меню **File** или контекстного меню выбрать **Save Workspace As**, или щелкнуть мышью по кнопке  инструментальной линейки ОПРП. Откроется диалоговое окно **Save**.

2. Задать местоположение и имя файла (**File name**). MATLAB автоматически добавит расширение **.mat**.

3. Щелкнуть **Save**.

Переменные рабочего пространства при этом запомнятся в МАТ-файле с выбранным вами именем.


Сохранение части переменных. Для этого надо:

1. Выбрать переменную в ОПРП. Для выбора нескольких переменных следует использовать клавиши **Shift** или **Ctrl**.

1. Вызвать щелчком правой кнопки контекстное меню и выбрать **Save Selection As**. Дальнейшие действия описаны выше.

Эквивалентные функции ! Для сохранения переменных рабочего пространства можно использовать команду **save** с указанием имени файла куда вы хотите запомнить данные. Например, команда **save('june10')** запоминает все переменные рабочего пространства в бинарном файле **june10.mat**. Если вы не зададите имя файла, рабочее пространство запоминается в специальном файле под названием **matlab.mat** в текущей рабочей директории. Вы можете задать какие переменные сохранять, а также формат запоминания данных, например **ascii**. Так, команда **save 'june10' X Y** сохраняет в названном файле только переменные **X Y**.

Загрузка запомненного рабочего пространства. Для загрузки предварительно запомненного рабочего пространства следует:

1. Щелкнуть на кнопку  загрузки данных на инструментальной линейке ОПРП, или щелкнуть правой кнопкой на ОПРП и выбрать в контекстном меню опцию **Import Data**. Откроется диалоговое окно **Open** (Открыть).

2. Выбрать МАТ-файл, который вы хотите загрузить и щелкнуть **Open**. Переменные и их значения, запомненные ранее в данном МАТ-файле, будут загружены в текущее рабочее пространство системы MATLAB.


Эквивалентные функции ! Функция **load** предназначена для загрузки запомненного рабочего пространства. Например, команда **load('june10')** загружает в рабочее пространство все переменные из файла **june10.mat**.

Примечание. Если в сохраненном МАТ-файле **june10** содержатся переменные обозначенные **A, B, и C**, то загрузка **june10** помещает эти переменные в рабочее пространство. Если переменные с этими именами уже существуют в рабочем пространстве, то они заменяются новыми переменными из **june10**.

Очистка переменных рабочего пространства. С помощью ОПРП вы можете удалить любые переменные из рабочего пространства. Для этого нужно:

1. Выбрать переменную в ОПРП (или несколько переменных с использованием клавиш **Shift** или **Ctrl**). Для выбора всех переменных нужно выбрать опцию **Select All** из меню **Edit** или контекстного меню.

2. Выполнить любое из перечисленных действий:

- Нажать клавишу **Delete**.
- Выбрать опцию **Delete** из меню **Edit**.
- Щелкнуть по кнопке  на инструментальной панели ОПРП.
- Выбрать опцию **Delete Selection** из контекстного меню.

3. Если появиться диалоговое окно подтверждения, щелкнуть **Yes**.

Для удаления сразу всех переменных нужно выбрать опцию **Clear Workspace** из меню **Edit** или контекстного меню ОПРП.

Эквивалентная функция ! Функция **clear** выполняет те же функции. Например, команда **clear A M**

удаляет переменные **A** и **M** из рабочего пространства, а команда **clear** без обозначения аргументов удаляет все переменные. Наконец, команда **clear all** удаляет все переменные и функции, т.е. полностью очищает рабочее пространство системы MATLAB

Просмотр основного (Base) рабочего пространства и рабочего пространства функций с использованием стека (Stack). При выполнении М-файлов MATLAB назначает (выделяет) каждому файлу (функции) свое собственное рабочее пространство, которое называется рабочим пространством функции и не совпадает с основными рабочим пространством системы MATLAB. При редактировании исполняемых файлов вы можете переходить из основного рабочего пространства в рабочее пространство любой исполняемой функции при помощи поля **Stack** в ОПРП. Данное поле активизируется только в режиме отладки (более подробно эта возможность будет рассмотрена в дальнейшем).


Построение графиков переменных. Вы можете построить график любой переменной из рабочего пространства. Для этого надо щелкнуть правой кнопкой мыши на переменную в ОПРП, которую вы хотите отобразить графически, и из появившегося контекстного меню выбрать опцию **Graph Selection** и далее выбрать тип графика, который вы хотите построить. Соответствующий график появится в открывшемся специальном окне представления графической информации (figure window). В дальнейшем для краткости эти окна будут называться графическими окнами.

Просмотр и редактирование массивов данных при помощи редактора Array Editor

Редактор Массива Данных (РМД) **Array Editor** предназначен для визуального просмотра и редактирования одно- и двумерных числовых массивов, символьных строк и ячеек символьных строк.

Открытие РМД. Для вызова РМД из ОПРП следует:

1. Выделить в ОПРП желаемую переменную или переменные обычным образом.

2. Щелкнуть по кнопке  в инструментальной линейке ОПРП или выбрать из контекстного меню опцию **Open Selection** . В случае одной переменной можно также ограничиться двойным щелчком по выбранной переменной.
В результате появится следующее окно:

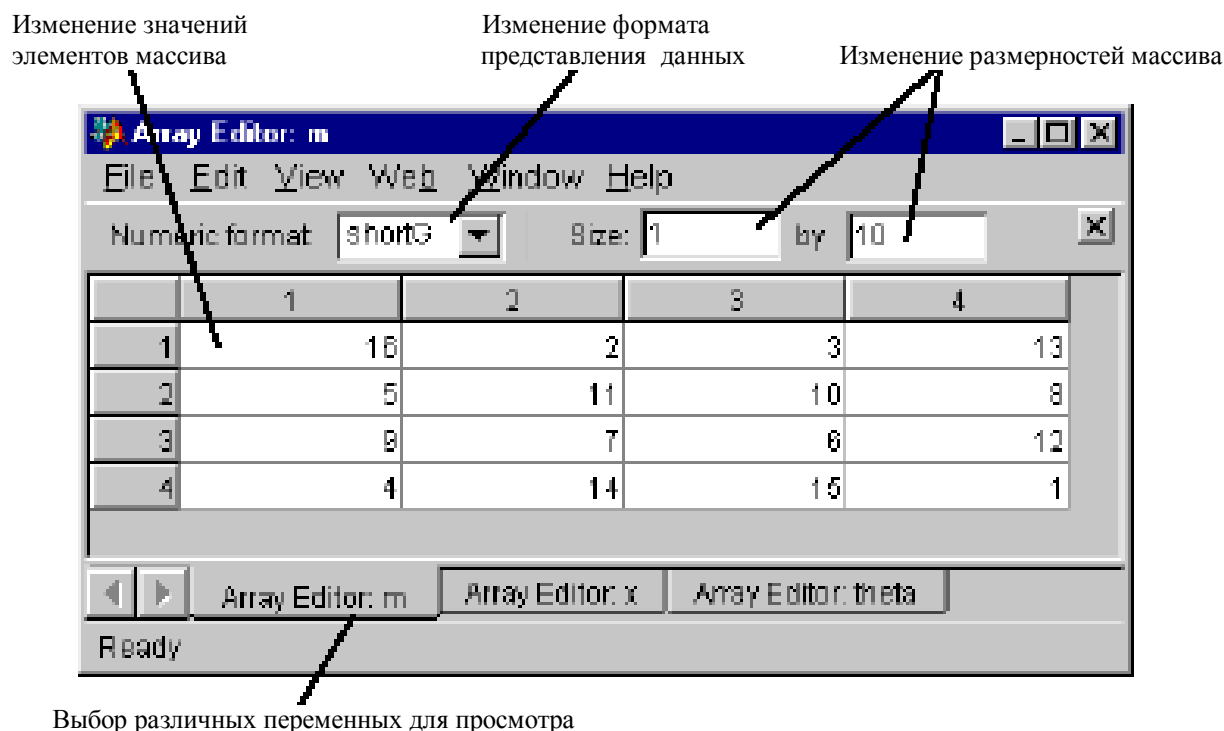


Рис.4. Общий вид окна Редактора Массива Данных

Повторение перечисленных выше шагов приведет к открытию дополнительных переменных в РМД. Доступ к каждой переменной осуществляется через ярлыки внизу окна РМД (см. рис. 4) или посредством меню **Window**.

Эквивалентная функция ! Для просмотра содержания переменной рабочего пространства достаточно напечатать ее имя в командной строке. Так, например, при вводе в командной строке переменной **m** (см. рис. 4) , MATLAB ответит

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

Для открытия РМД с требуемой переменной из командной строки MATLAB-а, можно воспользоваться функцией **openvar** с соответствующей переменной в качестве аргумента. Например, при вводе **openvar('m')** MATLAB откроет РМД с переменной **m**.

Изменение значений элементов в РМД. Для изменения значений элементов в РМД нужно щелкнуть на ту ячейку таблицы, которую вы хотите изменить. Далее следует ввести новое значение элемента и нажать **Enter** или щелкнуть по другой ячейке; произойдет соответствующее изменение. Для изменения размерностей массива, следует напечатать новые значения для числа строк и столбцов в поле **Size**. Если вы увеличиваете размер, то новые строки и

столбцы добавятся к концу таблицы и будут заполнены нулями. При уменьшении размера вы потеряете данные - MATLAB просто удалит последние строки и столбцы.

Пути доступа системы MATLAB

При поиске М-файлов или других файлов, хранящихся в вашей файловой структуре, MATLAB использует концепцию *путей доступа* (*search path*). Любой файл (функция), который вы хотите исполнить в системе MATLAB должен находиться в директории, находящейся на путях доступа или в текущей директории. По умолчанию, все файлы поставляемые с MATLAB и соответствующими пакетами прикладных программ фирмы MathWorks включены в пути доступа. Если вы создаете какой-либо файл, предназначенный для использования системой MATLAB, необходимо включить директорию, содержащую этот файл, в пути доступа системы MATLAB.

Внимание ! Если вы создаете свой собственный М-файл или модифицируете любой имеющийся М-файл, поставленный с системой MATLAB, сохраняйте их в директории, которая *не находится* на пути **\$matlabroot/toolbox/matlab**, где **\$matlabroot** – корневая директория системы MATLAB на вашем компьютере. Если вы храните какие-либо свои файлы в директории **\$matlabroot/toolbox/matlab**, они будут уничтожены при переустановке или установке новой версии MATLAB-а на вашем компьютере.

Суть концепции путей доступа. Все используемые вами файлы рассматриваются как находящиеся на путях доступа системы MATLAB. Когда вы включаете новую директорию в пути доступа, вы *добавляете* ее к имеющимся путям. Поддиректории (subdirectories) должны быть добавлены в пути доступа явным образом; они не включаются в пути доступа автоматически, при включении их родительских директорий. Пути доступа системы MATLAB хранятся в файле **pathdef.m**. Порядок расположения директорий на путях доступа имеет существенное значение. Система MATLAB ищет любой объект (переменную, функцию и т.д.), например, названный **foo**, следующим образом. Если вы вводите **foo** в командной строке, то система MATLAB выполняет следующие действия:

1. Ищет **foo** как переменную.
2. Проверяет, не является ли **foo** встроенной функцией.
3. Ищет в текущей директории файл названный **foo.m**.
4. Ищет по очереди во всех директориях на пути доступа MATLAB файл **foo.m**.

Хотя в действительности схема поиска является более сложной, эта упрощенная схема является достаточно точной для обычных М-файлов, с которыми имеет дело потребитель.

Порядок расположения директорий на путях доступа важен именно потому, что могут быть несколько файлов, имеющих одинаковое имя. Когда MATLAB ищет такую функцию, он выбирает только первую функцию, встретившуюся на путях доступа; остальные функции (файлы) оказываются в тени и не могут быть выполнены. Для того чтобы узнать какая функция выполняется, т.е. местоположение данной функции в файловой структуре, можно использовать команду **which FileName**, где **FileName** – имя выбранной функции.

Просмотр и изменение путей доступа. Для просмотра и изменения путей доступа системы MATLAB предусмотрено диалоговое окно **Set Path**, а также ряд эквивалентных команд (фу-

нкций), которые можно выполнить из командной строки. Для открытия указанного окна нужно выбрать опцию **Set Path** из меню **File** рабочего стола, или ввести команду **pathtool** из командной строки MATLAB –а. Откроется следующее диалоговое окно

При нажатии на эти кнопки происходят изменения в путях доступа текущего сеанса, но эти пути доступа не сохраняются автоматически для будущих сеансов

Кнопки для изменения путей доступа

Директории текущих путей доступа системы MATLAB

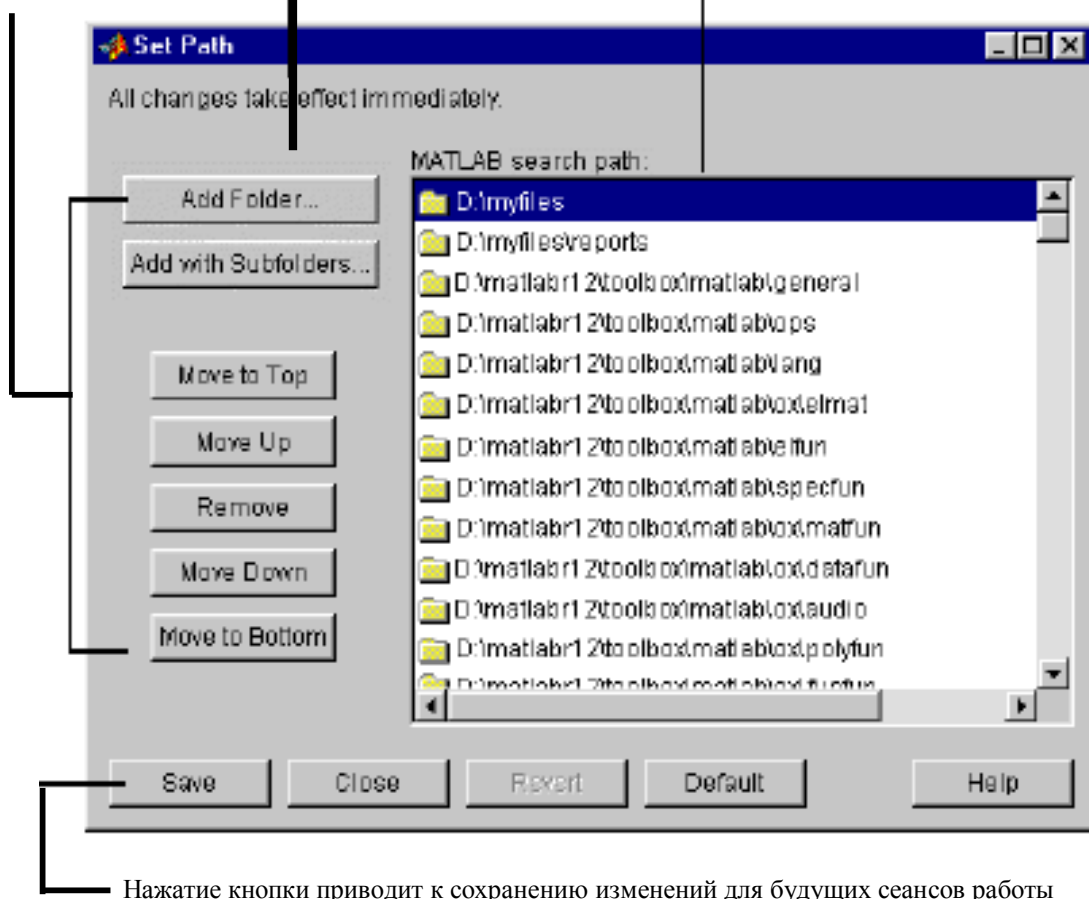


Рис. 5. Диалоговое окно Set Path для модификации путей доступа

На рис. 5 имеем следующие кнопки:

Add Folder...	- Добавить директорию (папку).
Add with Subfolders...	- Добавить директорию с поддиректориями.
Move to Top	- Перенести директорию в начало пути.
Move Up	- Перенести директорию на один шаг вверх.
Remove	- Удалить директорию.
Move Down	- Перенести директорию на один шаг вниз.
Move to Bottom	- Перенести директорию в конец пути.
Default	- Восстановить исходные пути доступа (по умолчанию).

Назначение данных кнопок ясно из их названий.

Операции с файлами

При операциях над файлами система MATLAB использует текущую директорию в качестве отправной точки. Любой файл, который вы хотите исполнить, должен находиться или в текущей директории (каталоге) или на пути доступа системы MATLAB. Аналогично, если вы хотите открыть какой-либо файл, начальной точкой диалогового окна **Open** всегда является текущая директория.

Поле текущей директории. Быстрый путь просмотра или изменения текущей директории состоит в использовании поля **Current Directory** (Текущая Директория) в инструментальной панели рабочего стола



Для изменения текущей директории из этого поля, нужно выполнить одно из следующих действий:

- Впечатать в данное поле путь к новой текущей директории.
- Щелкнуть на правой кнопке со стрелкой, что откроет список ранее использованных рабочих директорий, и выбрать из них желаемую.
- Щелкнуть на кнопку (...) (см. рис. 2) для выбора новой директории.

Окно Просмотра Текущего Каталога (Current Directory Browser) . Для поиска, просмотра, открывания и ввода изменений в директориях системы MATLAB можно воспользоваться Окном Просмотра Текущего Каталога (ОПТК). Все основные функции данного окна можно также выполнить задавая соответствующие команды из командной строки системы MATLAB. Если ОПТК не присутствует в рабочем столе MATLAB-а, то его можно открыть выбором опции **Current Directory** в меню **View**, или же введя команду **filebrowser** из командного окна (**Command Window**). При этом появится следующее окно

Данное поле позволяет произвести быстрый просмотр и изменение директорий

Щелкнув по кнопке «бинокль» можно исследовать содержимое любого М-файла (операция поиска по заданным символам)

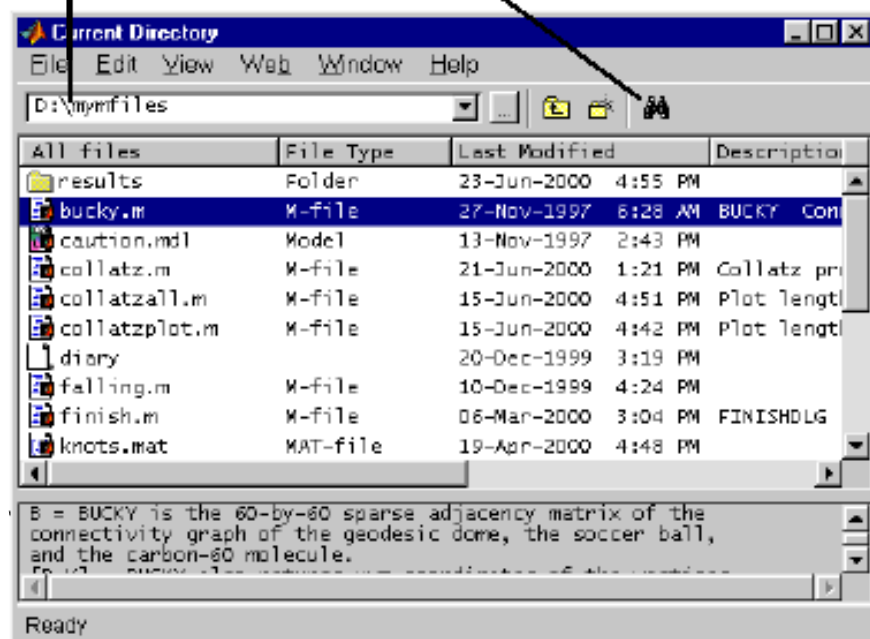


Рис. 6. Окно просмотра текущего каталога

Двойной щелчок по любому выделенному файлу откроет его содержание в соответствующем окне Редактора/Отладчика. В нижней части ОПТК (см. рис. 6) показана часть справки (help) для выделенного файла.


Изменение текущей директории и просмотр ее содержания в ОПТК. Для изменения текущей директории из ОПТК можно воспользоваться полем **Current Directory** этого окна, совершенно аналогично тому, как это было описано выше.

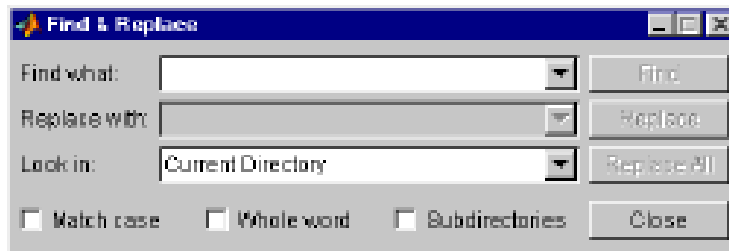
Эквивалентные функции ! Команда **dir** выводит в командное окно все содержимое текущей директории или любой другой директории, заданной как аргумент функции **dir**. Команда **what** действует аналогично, но выводит только те файлы, которые относятся к системе MATLAB. Команда **which FileName**, где **FileName** – имя файла, выводит путь доступа к данному файлу.

ОПТК позволяет также произвести любые добавление директорий к путям доступа системы MATLAB, а также создавать, копировать, переименовывать директории и т.д. (см. Руководство Пользователя).

Поиск и замена содержимого файлов. Из ОПТК можно осуществить поиск любой заданной строки символов в содержании файла. Если же файл открыт в окне Редактора/Отладчика, то можно также и заменить заданную строку на другую.

Поиск заданной строки в пределах файла (файлов). Для поиска заданной строки следует:

1. Щелкнуть по кнопке  инструментальной панели ОПТК (рис. 6). Появится следующее диалоговое окно «Найти и Заменить» (**Find & Replace**):

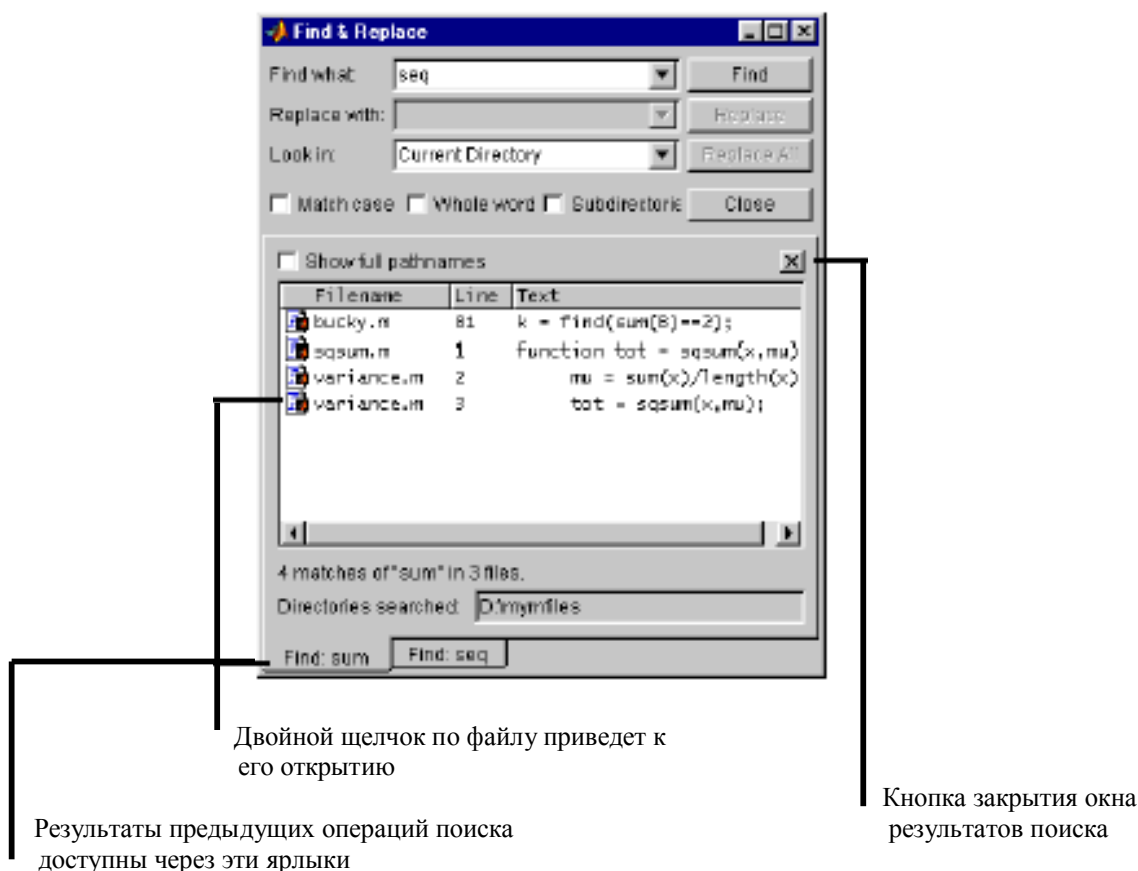


2. Для нахождения всех случаев наличия требуемой строки в файлах следует:

- Впечатать строку в поле **Find what**.
- Выбрать директорию поиска в поле **Look in**, или просто непосредственно напечатать имя директории в данном поле.
- Ограничить поиск выбором опций **Match case** (Учитывать регистр) и **Whole word** (Слово целиком).
- Выбрать опцию **Subdirectories** (Поддиректории), если вы хотите также просматривать и файлы в поддиректориях.

3. Щелкнуть по кнопке **Find**.

Результат поиска при этом будет отображен в нижней части диалогового окна **Find & Replace** как это показано на следующем рисунке. Этот результат включает название файла, номер соответствующей строки и содержимое данной строки.



4. Для открытия любого из М-файлов в списке результатов поиска нужно выполнить одно из нижеприведенных действий:

- Дважды щелкнуть по файлу.
- Выделить файл и нажать клавишу **Enter**.
- Щелкнуть правой кнопкой по файлу и выбрать опцию **Open** из контекстного меню.

Соответствующий файл будет открыт в Редакторе/Отладчике, причем он будет «прокручен» до той строки, которая содержит результат поиска.

5. Если вы до того проводили другой поиск (поиски), то все результаты предыдущих поисков будут доступны через «ярлыки» внизу текущего списка результатов поиска (см. рис.).

Замена заданной строки в пределах файла. После поиска заданной строки в пределах файла, вы можете автоматически заменить эту строку на любую другую. Для этого надо:

1. Открыть М-файл в окне Редактора/Отладчика системы MATLAB Editor. При этом нужно убедиться, что данный файл является текущим в окне Редактора/Отладчика.
2. Выбрать в поле **Look in** в диалоговом окне **Find & Replace** имя того файла, в котором вы хотите заменить строку. Кнопка **Replace** (Заменить) при этом активизируется.
3. Впечатать в поле **Replace with** текст, на который вы хотите заменить выбранную строку.

4. Щелкнуть **Replace** для замены строки символов в выбранной строке файла, или же щелкнуть по кнопке **Replace All**, если вы хотите заменить все найденные совпадения в текущем файле.

Текст будет заменен.

5. Для сохранения изменений нужно выбрать опцию **Save** из меню **File** в окне Редактора/ Отладчика.

Дуальность (двойственность) команд и функций

Команды системы MATLAB это выражения вида **load** или **help**. Многие команды допускают модификацию за счет определения операндов, например:

```
load August17.dat
help magic
type rank.
```

Альтернативный способ ввода подобных операндов в команды состоит в представлении их в виде символьных строк как аргументы функций.

```
load('August17.dat')
help('magic')
type('rank')
```

В этом состоит *дуальность команд/функций* системы MATLAB. Любая команда в форме

command argument

может также быть записана в функциональной форме

command('argument')

Преимущество функциональной формы записи проявляется когда символьный аргумент формируется машиной программно, из ряда разных кусков. Например, следующий пример загружает в рабочее пространство переменные из 31-го MAT-файла под названиями August1.dat, August2.dat, и т.д.

```
for d = 1:31
    s = ['August' int2str(d) '.dat']
    load(s)
end
```

Здесь использована функция **int2str**, которая преобразует целые числа в строку символов, что помогает сконструировать название файла, а также используются квадратные скобки для объединения трех символьных переменных в одно.

Действия над матрицами в системе MATLAB

Матрица является двумерным массивом действительных или комплексных чисел. Линейная алгебра и теория матриц определяют множество операций над матрицами, которые непосредственно поддерживаются (т.е. выполняются как стандартные операции) в MATLAB-е. В частности, сюда входят все элементарные действия над матрицами, решение систем линейных уравнений, нахождение собственных значений и векторов, а также сингулярных чисел и т.д. Ниже кратко рассмотрены действия над матрицами в системе MATLAB.

Формирование матриц в системе MATLAB

В дальнейшем для удобства будем считать термины *матрица* и *массив* эквивалентными. Более точно, матрица есть двумерный прямоугольный массив из действительных или комплексных чисел, который характеризует некоторое линейное преобразование. В MATLAB-е имеется множество встроенных функций, которые формируют (генерируют) различные типы матриц. Воспользуемся двумя из них для формирования пары матриц размера 3-х3, которые будут использоваться в дальнейшем в качестве примеров. Первый пример представляет симметричную матрицу Паскаля. Если ввести команду

```
A = pascal(3)
```

то система ответит

```
A =  
    1    1    1  
    1    2    3  
    1    3    6
```

Второй пример представляет несимметричную матрицу, известную под названием «волшебный квадрат» (magic square):

```
B = magic(3)
```

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

Еще один пример использования стандартной матрицы представляет собой прямоугольную 3х2 матрицу случайных целых чисел:

```
C = fix(10*rand(3,2))
```

```
C =  
    9    4  
    2    8  
    6    7
```

Здесь функция **rand(3,2)** генерирует 3х2 матрицу равномерно распределенных случайных чисел в диапазоне от 0 до 1, а функция **fix** осуществляет округление путем отбрасывания дробной части.

Вектор-столбец есть матрица размера $m \times 1$ matrix, *вектор-строка* – матрица размера $1 \times n$, а *скаляр* есть матрица размера 1×1 . Объединение отдельных чисел в массивы осуществляется

при помощи квадратных скобок, причем отдельные строки разделяются точкой с запятой, а переменные в каждой строке – запятой или пробелом (число пробелов может быть любым). Выражения

$$\mathbf{u} = [3; 1; 4]$$

$$\mathbf{v} = [2 \ 0 \ -1]$$

$$s = 7$$

дают вектор-столбец \mathbf{u} , вектор-строку \mathbf{v} и скаляр s (эти векторы также будут использоваться в дальнейшем при решении примеров):

$$\mathbf{u} = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} 2 & 0 & -1 \end{bmatrix}$$

$$s = 7$$

Сложение и вычитание матриц

Сложение и вычитание матриц определяется как и для массивов, то есть поэлементно. Эти операции требуют чтобы обе матрицы имели одинаковую размерность, или одна из них была *скаляром* (в последнем случае MATLAB прибавляет (или вычитывает) данный скаляр из всех элементов матрицы). Если размерности матриц не совпадают, в командное окно выдается сообщение об ошибке (красным цветом)

*Error using ==> +
Matrix dimensions must agree.*

Векторное произведение и транспонирование матриц

Вектор-строка и вектор-столбец могут быть перемножены в любом порядке (оператор умножения `*` расположен на верхнем регистре клавиши с цифрой 8). Результатом будет или скаляр (*внутреннее* произведение) или матрица (*внешнее* произведение). Для приведенных выше векторов \mathbf{v} и \mathbf{u} имеем :

$$\mathbf{x} = \mathbf{v} * \mathbf{u}$$
$$\mathbf{x} = 2$$

$$\mathbf{X} = \mathbf{u} * \mathbf{v}$$
$$\mathbf{X} = \begin{bmatrix} 6 & 0 & -3 \\ 2 & 0 & -1 \\ 8 & 0 & -4 \end{bmatrix}$$

Для действительных матриц, операция *транспонирования* меняет взаимное местоположение элементов a_{ij} a_{ji} , симметричных относительно главной диагонали. Для обозначения транспонирования MATLAB использует одиночную кавычку (апостроф) ('). Для нашей симметричной матрицы Паскаля $A' = A$. Однако матрица **B** не является симметричной и поэтому:

$$X = B'$$

$$X = \begin{bmatrix} 8 & 3 & 4 \\ 1 & 5 & 9 \\ 6 & 7 & 2 \end{bmatrix}$$

Транспонирование превращает вектор-строку в вектор-столбец и наоборот. Если **x** и **y** оба являются действительными векторами, то произведение $x*y$ не определено, но оба произведения $x'*y$ и $y'*x$ дают один и тот же скаляр. Это соотношение используется так часто, что имеет три различных имени: *скалярное* произведение, *внутреннее* произведение и *точечное* произведение.

Для *комплексного* вектора или матрицы, **z**, величина z' обозначает *комплексно-сопряженное* транспонирование. В MATLAB-е предусмотрены также поэлементные операции над элементами массивов. Признаком поэлементных операций служит *точка* после обозначения переменной. Так, транспонирование элементов матрицы **z** как массива чисел обозначается $z.'$, по аналогии с другими операциями на массивах чисел. Например, если

$$z = [1+2i \ 3+4i]$$

то

$$z' = \begin{bmatrix} 1-2i \\ 3-4i \end{bmatrix}$$

тогда как $z.'$ есть

$$z.' = \begin{bmatrix} 1+2i \\ 3+4i \end{bmatrix}$$

Для комплексных векторов, два скалярных произведения $x'*y$ и $y'*x$ комплексно сопряжены, а скалярное произведение $x'*x$ комплексного вектора с самим собой есть действительное число.

Произведение матриц

Для произведения двух совместимых **A** и **B** матриц в MATLAB-е достаточно записать в командной строке $C = A*B$. MATLAB самостоятельно проверит совместимость размерностей матриц и выдаст результат. Если матрицы несовместимы, выдается сообщение об ошибке:

*Error using ==> *
Inner matrix dimensions must agree.*

Индексирование (Subscripts)

Для краткого рассмотрения некоторых основных понятий, связанных с индексированием двумерных массивов (матриц), введем «волшебную» матрицу 4-го порядка:

F = magic(4)

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

Элемент в **i**-ой строке и **j**-ом столбце матрицы **F** обозначается через **F (i,j)**. Например, **F (4,2)** есть число в четвертой строке и втором столбце. Для нашего волшебного квадрата, **F(4,2)** есть 14. Таким образом, можно вычислить сумму элементов четвертого столбца матрицы **F**, напечатав

F (1,4) + F (2,4) + F (3,4) + F (4,4)

Это дает ответ

ans =
34

но, как мы увидим в дальнейшем, не является самым элегантным способом суммирования элементов одного столбца.

Имеется также возможность обращения к элементам матрицы при помощи одного индекса, **F(k)**. Это обычный способ обращения к элементам векторов (строк или столбцов). Но в MATLAB-е такой способ индексирования можно применить и к двумерным (в общем случае – многомерным) матрицам, так как система MATLAB хранит все многомерные массивы чисел в виде одного длинного вектора-столбца, сформированного из столбцов исходной матрицы. Так, для нашего волшебного квадрата, **F (8)** есть другой способ обращения к значению 14 хранящемуся в **F (4,2)**.

Если вы попытаетесь использовать элемент, находящийся вне размеров матрицы, это приведет к сообщению об ошибке

t = F (4,5)
Index exceeds matrix dimensions
(Индекс превышает размерность матрицы)

С другой стороны, если вы попытаетесь запомнить какое-либо число вне размеров матрицы, размер будет соответствующим образом увеличен, чтобы принять новое значение.

X = A;
X(4,5) = 17

X =
16 3 2 13 0
5 0 11 8 0
9 6 7 12 0
4 15 14 1 17

Двоеточие (Colon)

Двоеточие, `:`, является одним из наиболее важных операторов MATLAB-а. Оно встречается в нескольких разных формах. Выражение `1:10` есть вектор-строка, содержащий целые числа от **1** до **10**:

1 2 3 4 5 6 7 8 9 10

Чтобы получить неединичное приращение, нужно задать приращение. Например,

100 : -7 : 50

есть

100 93 86 79 72 65 58 51

а

0 : pi/4 : pi

есть

0 0.7854 1.5708 2.3562 3.1416

Индексы, содержащие двоеточия, допускают обращение к *частям* матриц. Так, выражение

F (1:k, j)

дает первые **k** элементов **j**-го столбца матрицы **F**. То есть,

sum(F (1:4, 4))

вычисляет, как и в примере выше, сумму элементов 4-го столбца. Но есть еще лучший путь. Двоеточие само по себе означает обращение ко *всем* элементам строки или столбца матрицы, а зарезервированное слово **end** есть обращение к *последним* строке или столбцу матрицы (в случае векторов-строк или столбцов слово **end** есть обращение к *последнему* элементу вектора). Значит,

sum(F (:, end))

вычисляет сумму элементов последнего столбца матрицы **F**. Ответ: **ans = 34**. Почему магическая сумма для волшебного квадрата 4 x 4 равна 34 ? Дело в том, что если целые числа от 1 до 16 (число элементов матрицы размера 4 x 4) упорядочены в четыре группы с равными суммами элементов, эта сумма должна быть равна

sum(1:16)/4

что, конечно, дает **ans = 34**.

Единичная матрица, нулевая матрицы и матрица из единиц.

Двумерные массивы случайных чисел

Единичная матрица, то есть матрица имеющая единицы на главной диагонали и нулевые остальные элементы, в MATLAB-е обозначается **eye**, причем **eye(n)** есть единичная квадратная матрица размера $n \times n$, **eye(m,n)** - прямоугольная единичная матрица размера $m \times n$, а **eye(size(A))** есть единичная матрица, имеющая размерность матрицы **A**. Например,

I = eye(3)

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$I = \text{eye}(3,5)$$

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$I = \text{eye}(4,2)$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Нулевая матрица, то есть матрица состоящая из нулей (массив нулей), в MATLAB-е обозначается **zeros**, причем **zeros (n)** есть нулевая квадратная матрица размера nxn, **zeros (m,n)** - прямоугольная нулевая матрица размера mxn, а **zeros (size(A))** есть нулевая матрица имеющая размерность матрицы **A**.

$$Z = \text{zeros}(2,4)$$

$$Z = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Наконец, матрица состоящая из единиц (массив единиц), в MATLAB-е обозначается **ones**, причем **ones (n)** есть квадратный массив единиц размера nxn, **ones (m,n)** – прямоугольный массив единиц размера mxn, а **ones (size(A))** есть массив единиц, имеющий размерность матрицы **A**.

$$S = 5 * \text{ones}(3, 3)$$

$$S = \begin{bmatrix} 5 & 5 & 5 \\ 5 & 5 & 5 \\ 5 & 5 & 5 \end{bmatrix}$$

Аналогично, функция **rand** дает возможность сформировать соответствующие массивы случайных чисел в диапазоне от 0 до 1, распределенных по равномерному закону, а функция **randn** – по нормальному закону.

$$N = \text{fix}(10 * \text{rand}(1,10))$$

$$N = \begin{bmatrix} 4 & 9 & 4 & 4 & 8 & 5 & 2 & 6 & 8 & 0 \end{bmatrix}$$

$$R = \text{randn}(4,4)$$

R =

```
1.0668 0.2944 -0.6918 -1.4410
0.0593 -1.3362 0.8580 0.5711
-0.0956 0.7143 1.2540 -0.3999
-0.8323 1.6236 -1.5937 0.6900
```

Решение систем линейных уравнений

Одной из важнейших задач в технических приложениях и расчетах является задача решения систем линейных уравнений. В матричных обозначениях, данная задача может быть сформулирована следующим образом. При заданных двух матрицах **A** and **B**, существует ли такая единственная матрица **X**, что **AX = B** или **XA = B**?

Для наглядности рассмотрим одномерный пример. Имеет ли уравнение

$$7x = 21$$

единственное решение? Ответ, разумеется, да. Это уравнение имеет единственное решение **x = 3**. Решение может быть легко получено обычным *делением*.

$$x = 21/7 = 3$$

Решение при этом обычно *не состоит* в определении обратной величины от числа 7 (т.е. величины $7^{-1} = 0.142857\dots$), и последующим умножением числа 7^{-1} на число 21. Это было бы более трудоемко и, если число 7^{-1} представлено конечным числом цифр (разрядов), менее точно. Аналогичные рассуждения применимы и к системам линейных алгебраических уравнений с более чем одной неизвестной; MATLAB решает такие уравнения без вычисления обратной матрицы. Хотя это и не является стандартным математическим обозначением, система MATLAB использует терминологию, связанную с обычным делением в одномерном случае, для описания общего случая решения совместной системы нескольких линейных уравнений. Два символа деления / (косая черта (по английски - *slash*)) и \ (обратная косая черта (*backslash*)) используются в двух случаях, когда неизвестная матрица появляется слева или справа от матрицы коэффициентов:

X = A\B обозначает решение матричного уравнения **AX = B**

X = B/A обозначает решение матричного уравнения **XA = B**.

Вы можете представлять себе это как процесс «деления» обеих частей уравнения **AX = B** или **XA = B** на **A**. Матрица коэффициентов **A** всегда находится в «знаменателе». Условие совместимости размерностей для **X = A\B** требует чтобы две матрицы **A** и **B** имели одинаковое число строк. Решение **X** тогда имеет такое же число столбцов как и **B**, а число ее строк будет равно числу столбцов **A**. Для **X = B/A**, строки и столбцы меняются ролями. На практике, линейные уравнения в виде **AX = B** встречаются более часто, чем в виде **XA = B**. Следовательно, обратная наклонная черта \ используется более часто, чем прямая /. Поэтому, в оставшейся части данного раздела мы ограничимся рассмотрением оператора \ ; соответствующие свойства оператора / можно вывести из тождества

$$(B/A)' = (A'\backslash B')$$

В общем случае не требуется, чтобы матрица коэффициентов **A** была бы квадратной. Если **A** имеет размер $m \times n$, то возможны три случая:

1. $m = n$ Квадратная система. Ищется точное решение.
2. $m > n$ Переопределенная система. Ищется решение методом наименьших квадратов.
3. $m < n$ Недоопределенная система. Находится базовое решение с самым большим числом m ненулевых компонент.

Оператор `\` использует различные алгоритмы для решения систем линейных уравнений с разными типами матриц коэффициентов. Различные случаи, которые диагностируются автоматически по типу матрицы коэффициентов, включают:

- Перестановки треугольных матриц
- Симметричные, положительно определенные матрицы
- Квадратные невырожденные матрицы
- Прямоугольные, переопределенные системы
- Прямоугольные, недоопределенные системы

Квадратные системы

Наиболее часто встречающейся ситуацией является квадратная матрица коэффициентов **A** и одномерный вектор-столбец **b** справа, т.е. $\mathbf{Ax} = \mathbf{b}$. Решение $\mathbf{x} = \mathbf{A} \backslash \mathbf{b}$ имеет при этом тот же размер, что и вектор **b**. Например,

$$\mathbf{x} = \mathbf{A} \backslash \mathbf{u}$$

$$\mathbf{x} = \begin{pmatrix} 10 \\ -12 \\ 5 \end{pmatrix}$$

где матрица **A** есть приведенная выше матрица Паскаля. Легко удостовериться, что $\mathbf{A} * \mathbf{x}$ в точности равно вектору **u** (численные значения этого вектора даны выше).

Если **A** и **B** являются квадратными и имеют одинаковый размер, то $\mathbf{X} = \mathbf{A} \backslash \mathbf{B}$ имеет тот же размер, например

$$\mathbf{X} = \mathbf{A} \backslash \mathbf{B}$$

$$\mathbf{X} = \begin{pmatrix} 19 & -3 & -1 \\ -17 & 4 & 13 \\ 6 & 0 & -6 \end{pmatrix}$$

Легко убедиться, что $\mathbf{A} * \mathbf{X}$ в точности равно **B**.

Оба этих примера имеют точное решение в виде целых чисел. Это связано с тем, что в качестве матрицы коэффициентов была выбрана матрица Паскаля **pascal(3)**, чей детерминант равен единице. Далее будут рассмотрены примеры влияния ошибок округления, возникающих в более реальных системах.

Квадратная матрица **A** является *сингулярной*, если ее столбцы не являются линейно независимыми. Если **A** – сингулярна, то решение $\mathbf{AX} = \mathbf{B}$ или не существует, или не является единственным. Оператор `\`, $\mathbf{A} \backslash \mathbf{B}$, выдает предупреждающее сообщение, если матрица **A** близка к сингулярной и сообщение об ошибке, если определено равенство нулю детерминанта матрицы **A**.

Переопределенные системы

Переопределенные системы совместных линейных уравнений часто встречаются в задачах аппроксимации экспериментальных данных при помощи различных эмпирических кривых. Рассмотрим следующий гипотетический пример. Величина y измеряется при различных значениях времени t , что дает следующие результаты

t	y
0.0	0.82
0.3	0.72
0.8	0.63
1.1	0.60
1.6	0.55
2.3	0.50

Эти данные могут быть введены в MATLAB при помощи выражений:

$$t = [0 \ 0.3 \ 0.8 \ 1.1 \ 1.6 \ 2.3]';$$

$$y = [0.82 \ 0.72 \ 0.63 \ 0.60 \ 0.55 \ 0.50]';$$

Данные могут быть аппроксимированы при помощи убывающей экспоненциальной функции.

$$y(t) = c_1 + c_2 e^{-t}$$

Это уравнение показывает, что вектор y может быть представлен в виде линейной комбинации двух векторов, один из которых является постоянным вектором, содержащим все единицы, а второй вектор имеет компоненты e^{-t} . Неизвестные коэффициенты c_1 и c_2 могут быть найдены подгонкой кривых по методу наименьших квадратов, которая основана на минимизации суммы квадратов отклонений экспериментальных данных от модели. Мы имеем шесть уравнений с двумя неизвестными, представленными 6×2 матрицей

$$E = [\text{ones}(\text{size}(t)) \ \exp(-t)]$$
$$E =$$

1.0000	1.0000
1.0000	0.7408
1.0000	0.4493
1.0000	0.3329
1.0000	0.2019
1.0000	0.1003

Решение методом наименьших квадратов находится при помощи оператора \backslash :

$$c = E \backslash y$$
$$c =$$

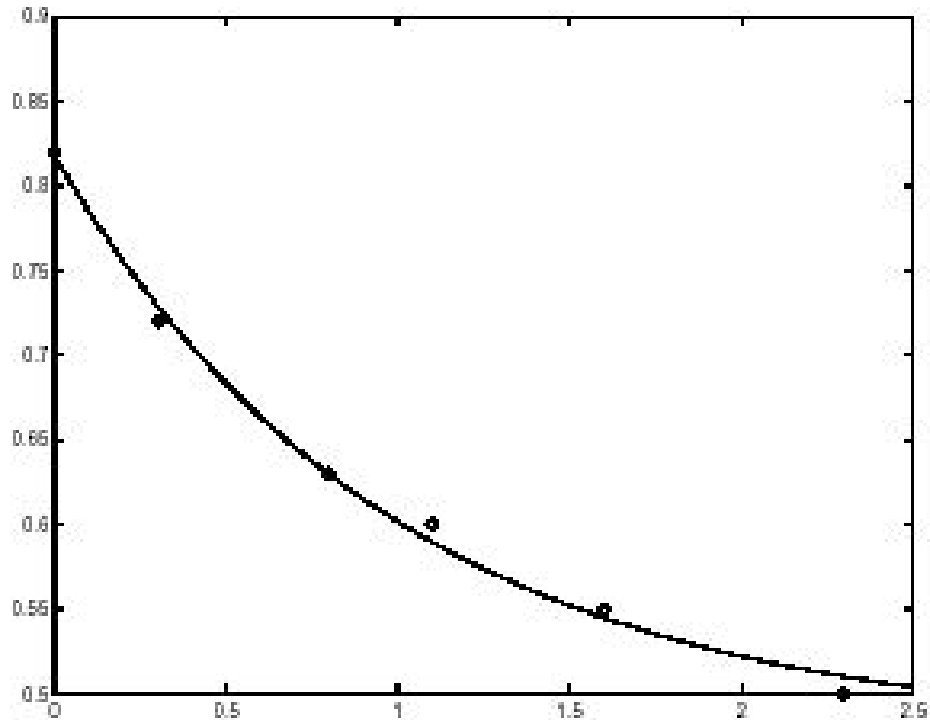
0.4760
0.3413

Иными словами, подгонка методом наименьших квадратов дает

$$y(t) = 0.476 + 0.3413 e^{-t}$$

Следующие выражения оценивают модель при равномерно распределенных моментах времени (с шагом 0.1), а затем строят график вместе с результатами экспериментальных данных.

```
T = (0 : 0.1 : 2.5)';
Y = [ones(size(T)) exp(-T)]*c;
plot(T, Y, '-', t, y, 'o')
```



Можно видеть, что значения $E \cdot c$ не совсем точно совпадают со значениями экспериментальных данных y , но эти отклонения могут быть сравнимы с ошибками измерений.

Прямоугольная матрица A называется матрицей *неполного ранга*, если ее столбцы линейно-независимы. Если матрица A имеет неполный ранг, то решение $AX = B$ не является единственным. Оператор `\` при этом выдает предупреждающее сообщение и определяет *основное* решение, которое дает минимально возможное число ненулевых решений.

Недоопределенные системы

Недоопределенные системы линейных уравнений содержат больше неизвестных чем уравнений. Когда они сопровождаются дополнительными ограничениями, то становятся сферой изучения *линейного программирования*. Сам по себе, оператор `\` работает только с системой без ограничений. При этом решение никогда не бывает единственным. MATLAB находит *основное* решение, которое содержит по меньшей мере m ненулевых компонент (где m - число уравнений), но даже это решение может быть не единственным. Ниже приводится пример, где исходные данные генерируются случайным образом.

```

R = fix (10*rand(2,4))
R =
    6  8  7  3
    3  5  4  1

```

```

b = fix (10*rand(2,1))
b =
    1
    2

```

Система уравнений $\mathbf{R}\mathbf{x} = \mathbf{b}$ содержит два уравнения с четырьмя неизвестными. Поскольку матрица коэффициентов \mathbf{R} содержит небольшие по величине целые числа, целесообразно представить решение в формате *rational* (в виде отношения двух целых чисел). Частное решение представленное в указанном формате есть:

```

p = R\b
p =
    0
    5/7
    0
   -11/7

```

Одно из ненулевых решений есть $\mathbf{p}(2)$, потому что второй столбец матрицы \mathbf{R} имеет наибольшую норму. Вторая ненулевая компонента есть $\mathbf{p}(4)$ поскольку четвертый столбец матрицы \mathbf{R} становится доминирующим после исключения второго столбца (решение находится методом QR-факторизации с выбором опорного столбца).

Обратные матрицы и детерминанты

Если матрица \mathbf{A} является квадратной и невырожденной, уравнения $\mathbf{A}\mathbf{X} = \mathbf{I}$ и $\mathbf{X}\mathbf{A} = \mathbf{I}$ имеют одинаковое решение \mathbf{X} . Это решение называется матрицей *обратной* к \mathbf{A} , обозначается через \mathbf{A}^{-1} и вычисляется при помощи функции `inv`. Понятие детерминанта (определителя) матрицы полезно при теоретических выкладках и некоторых типах символьных вычислений, но его масштабирование и неизбежные ошибки округления делают его не столь привлекательным при числовых вычислениях. Тем не менее, если это требуется, функция `det` вычисляет определитель квадратной матрицы. Например,

```

A = pascal (3)
A =
    1  1  1
    1  2  3
    1  3  6

d = det (A)
X = inv (A)
d =
    1

```

$$\mathbf{X} = \begin{bmatrix} 3 & -3 & 1 \\ -3 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Опять таки, поскольку \mathbf{A} является симметричной матрицей целых чисел и имеет единичный определитель, то же самое справедливо и для обратной матрицы. С другой стороны, для

$$\begin{aligned} \mathbf{B} &= \text{magic}(3) \\ \mathbf{B} &= \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \\ d &= \det(\mathbf{B}) \\ \mathbf{X} &= \text{inv}(\mathbf{B}) \\ d &= -360 \\ \mathbf{X} &= \begin{bmatrix} 0.1472 & -0.1444 & 0.0639 \\ -0.0611 & 0.0222 & 0.1056 \\ -0.0194 & 0.1889 & -0.1028 \end{bmatrix} \end{aligned}$$

Внимательное изучение элементов матрицы \mathbf{X} , или использование формата *rational*, показывает, что они являются целыми числами, разделенными на 360.

Если матрица \mathbf{A} является квадратной и несингулярной, то, пренебрегая ошибками округления, выражение $\mathbf{X} = \text{inv}(\mathbf{A}) * \mathbf{B}$ теоретически означает то же, что и $\mathbf{X} = \mathbf{A} \backslash \mathbf{B}$, а $\mathbf{Y} = \mathbf{B} * \text{inv}(\mathbf{A})$ теоретически есть то же, что и $\mathbf{Y} = \mathbf{B} / \mathbf{A}$. Однако вычисления включающие операторы \backslash и $/$ более предпочтительны, поскольку требуют меньше рабочего времени, меньшей памяти и имеют лучшие свойства с точки зрения определения ошибок.

Псевдообратные матрицы

Прямоугольные матрицы не имеют детерминантов и обратных матриц. Для таких матриц по крайней мере одно из уравнений $\mathbf{AX} = \mathbf{I}$ или $\mathbf{XA} = \mathbf{I}$ не имеет решения. Частично данный пробел восполняется так называемой *псевдообратной матрицей Мура-Пенроуза*, или просто *псевдообратной* матрицей, которая вычисляется при помощи функции **pinv**. На практике необходимость в этой операции встречается довольно редко. Желаящие могут всегда обратиться к соответствующим справочным пособиям.

Степени матриц и матричные экспоненты

Положительные целые степени

Если \mathbf{A} есть некоторая квадратная матрица, а p – положительное целое число, то \mathbf{A}^p эквивалентно умножению \mathbf{A} на себя p раз.

$$\mathbf{X} = \mathbf{A}^2$$

X =

**3 6 10
6 14 25
10 25 46**

Отрицательные и дробные степени

Если **A** является квадратной и невырожденной, то **A^(-p)** эквивалентно умножению **inv(A)** на себя **p** раз.

Y = B⁽⁻³⁾

Y =

**0.0053 -0.0068 0.0018
-0.0034 0.0001 0.0036
-0.0016 0.0070 -0.0051**

Дробные степени, например **A^(2/3)**, также допускаются; результаты при этом зависят от распределения собственных значений матрицы **A**.

Поэлементное возведение в степень

Оператор **.^** (с точкой !) осуществляет *поэлементное* возведение в степень. Например,

X = A.^2

A =

**1 1 1
1 4 9
1 9 36**

Вычисление корня квадратного из матрицы и матричной экспоненты

Для невырожденных квадратных матриц **A** функция **sqrtnm** вычисляет главное значение квадратного корня, т.е. если **X = sqrtnm(A)**, то **X*X = A**. Буква **m** в **sqrtnm** означает, что выполняется матричная операция. Это отличает данную функцию от **sqrt(A)**, которая, подобно **A^(1/2)** (обратите внимание на точку !), выполняет операцию извлечения корня *поэлементно*.

Система обыкновенных линейных дифференциальных уравнений первого порядка может быть записана в виде

$$dx/dt = Ax$$

где **x = x(t)** есть векторная функция от **t**, а **A** есть постоянная матрица не зависящая от **t**. Решение данной системы может быть выражено в виде *матричной экспоненты*.

$$x(t) = e^{At}x(0)$$

Функция **expm(A)** вычисляет матричную экспоненту. Рассмотрим пример системы дифференциальных уравнений со следующей 3x3 матрицей коэффициентов

A =

$$\begin{bmatrix} 0 & -6 & -1 \\ 6 & 2 & -16 \\ -5 & 20 & -10 \end{bmatrix}$$

и начальными условиями $\mathbf{x}(0)$

$$\mathbf{x0} = [1 \ 1 \ 1]'$$

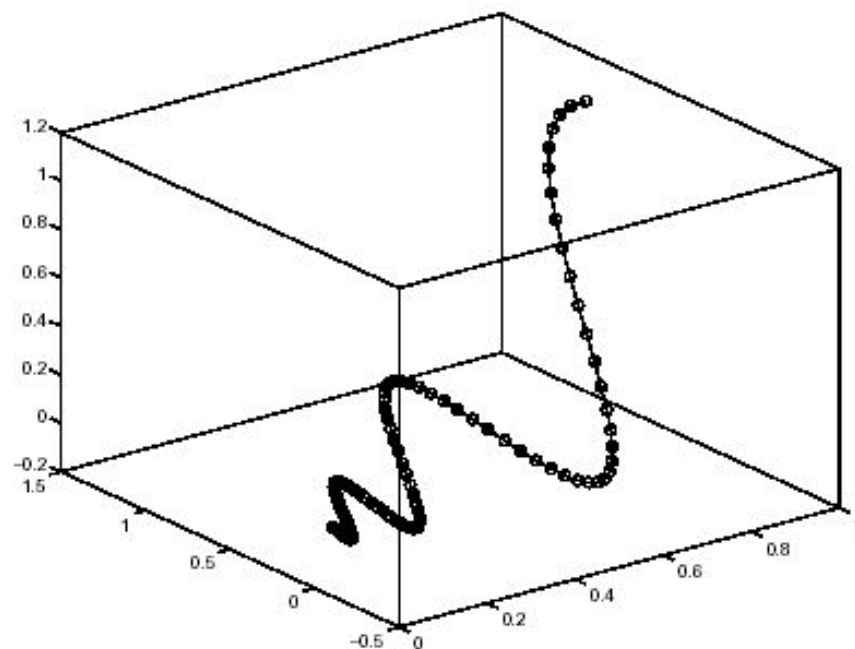
Использование матричной экспоненты для вычисления решения дифференциального уравнения в 101 точке с шагом 0.01 на интервале $0 \leq t \leq 1$ записывается в виде

```
X = [ ];
for t = 0 : 0.01 : 1
    X = [X expm(t*A)*x0];
end
```

Трехмерный график решения в фазовом пространстве может быть получен при помощи специальной функции

```
plot3(X(1,:), X(2,:), X(3,:), '-o')
```

Решение имеет вид спиральной функции сходящейся к началу координат (см. рис. ниже). Такое решение обусловлено комплексными собственными значениями матрицы коэффициентов \mathbf{A} .



Собственные значения и собственные векторы

Собственным значением и **собственным вектором** квадратной матрицы \mathbf{A} называются скаляр λ и вектор \mathbf{v} , удовлетворяющие условию

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

Диагональная декомпозиция

Имея диагональную матрицу Λ , составленную из собственных значений λ матрицы A и матрицу V , составленную из соответствующих собственных векторов v , можно записать

$$AV = V\Lambda$$

Если матрица V несингулярная, на основании данного выражения получаем спектральное разложение матрицы A

$$A = V\Lambda V^{-1}$$

Неплохой пример использования спектрального разложения дает рассмотренная выше матрица коэффициентов линейного дифференциального уравнения. Ввод выражения

$$\text{lambda} = \text{eig}(A)$$

дает следующий вектор-столбец собственных значений (два из них являются комплексно-сопряженными)

$$\begin{aligned} \text{lambda} = \\ & -3.0710 \\ & -2.4645 + 17.6008i \\ & -2.4645 - 17.6008i \end{aligned}$$

Действительные части всех собственных значения являются отрицательными, что обеспечивает устойчивость процессов в системе. Ненулевые мнимые части комплексно-сопряженных собственных значений обуславливают колебательный характер переходных процессов.

При двух выходных аргументах, функция **eig** вычисляет также собственные векторы и выдает собственные значения в виде диагональной матрицы

$$[V,D] = \text{eig}(A)$$

$$\begin{aligned} V = \\ & \begin{bmatrix} -0.8326 & 0.2003 - 0.1394i & 0.2003 + 0.1394i \\ -0.3553 & -0.2110 - 0.6447i & -0.2110 + 0.6447i \\ -0.4248 & -0.6930 & -0.6930 \end{bmatrix} \\ \\ D = \\ & \begin{bmatrix} -3.0710 & 0 & 0 \\ 0 & -2.4645+17.6008i & 0 \\ 0 & 0 & -2.4645-17.6008i \end{bmatrix} \end{aligned}$$

Первый собственный вектор (первый столбец матрицы V) является действительным, а два других являются комплексно-сопряженными. Все три вектора являются нормализованными по длине, т.е. их *Евклидова норма* $\text{norm}(v,2)$, равна единице.

Матрица $V \cdot D \cdot \text{inv}(V)$, которая в более сжатой форме может быть записана как $V \cdot D / V$, равна, в пределах погрешностей округления, матрице A . Аналогично, $\text{inv}(V) \cdot A \cdot V$, или $V \backslash A \cdot V$, равна, в пределах погрешностей округления, матрице D .

Дефектные матрицы

Некоторые матрицы не имеют спектрального разложения. Такие матрицы называются *дефектными* или *не диагонализуемыми*. Например, пусть матрица A имеет вид

$$A = \begin{pmatrix} 6 & 12 & 19 \\ -9 & -20 & -33 \\ 4 & 9 & 15 \end{pmatrix}$$

Для этой матрицы ввод $[V, D] = \text{eig}(A)$ дает

$$V = \begin{pmatrix} -0.4741 & -0.4082 & -0.4082 \\ 0.8127 & 0.8165 & 0.8165 \\ -0.3386 & -0.4082 & -0.4082 \end{pmatrix}$$
$$D = \begin{pmatrix} -1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 \\ 0 & 0 & 1.0000 \end{pmatrix}$$

Здесь имеются два положительных единичных кратных собственных значений. Второй и третий столбцы матрицы V являются одинаковыми и поэтому полного набора линейно-независимых собственных векторов не существует (и поэтому не существует обратная матрица V^{-1}).

Сингулярное разложение матриц

Сингулярным значением и соответствующими *сингулярными векторами* прямоугольной матрицы A называются скаляр σ и пара векторов u и v такие, что удовлетворяются соотношения

$$Av = \sigma u$$
$$A^T u = \sigma v$$

Имея диагональную матрицу сингулярных чисел Σ и две ортогональные матрицы U и V , сформированные из соответствующих собственных векторов, можно записать

$$AV = U \Sigma$$
$$A^T U = V \Sigma$$

Поскольку U и V являются ортогональными матрицами, это можно записать в виде *сингулярного разложения*

$$A = U \Sigma V^T$$

Полное сингулярное разложение матрицы A размера $m \times n$ включает $m \times m$ матрицу U , $m \times n$ матрицу Σ , и $n \times n$ матрицу V . Другими словами, обе матрицы U и V являются квадратными, а матрица Σ имеет тот же размер, что и A . Если A имеет намного больше строк чем столбцов, результирующая матрица U может быть достаточно большой, но большинство ее столбцов умножаются на нули в Σ . В таких ситуациях может быть использована так называемая

экономичная декомпозиция, которая сберегает как время так и память, за счет вывода матрицы U размера $m \times n$, матрицы Σ размера $n \times n$ и той же матрицы V .

Спектральное разложение является подходящим инструментом анализа матрицы, когда последняя осуществляет преобразование векторного пространства в себя, как это было в рассмотренном выше примере дифференциальных уравнений. С другой стороны, сингулярное разложение матриц удобно при отображении одного векторного пространства в другое, возможно с иной размерностью. Большинство систем совместных линейных уравнений относятся ко второй категории. Если матрица A является квадратной, симметричной и положительно-определенной, то ее спектральное и сингулярное разложения совпадают. Но при отклонении A от симметричной и положительно-определенной матрицы, разница между двумя разложениями возрастает. В частности, сингулярное разложение действительной матрицы всегда действительно, но спектральное разложение действительной несимметричной матрицы может быть и комплексным.

Для матрицы

$$A = \begin{pmatrix} 9 & 4 \\ 6 & 8 \\ 2 & 7 \end{pmatrix}$$

полное сингулярное разложение задается в форме

$$[U, S, V] = \text{svd}(A)$$

и приводит к следующим результатам

$$U = \begin{pmatrix} -0.6105 & 0.7174 & 0.3355 \\ -0.6646 & -0.2336 & -0.7098 \\ -0.4308 & -0.6563 & 0.6194 \end{pmatrix}$$

$$S = \begin{pmatrix} 14.9359 & 0 \\ 0 & 5.1883 \\ 0 & 0 \end{pmatrix}$$

$$V = \begin{pmatrix} -0.6925 & 0.7214 \\ -0.7214 & -0.6925 \end{pmatrix}$$

Вы можете убедиться, что матрица $U \cdot S \cdot V'$ равна A с точностью до ошибок округления. Для этого примера *экономичная* декомпозиция дает незначительный эффект.

$$[U, S, V] = \text{svd}(A, 0)$$

$$U = \begin{pmatrix} -0.6105 & 0.7174 \\ -0.6646 & -0.2336 \\ -0.4308 & -0.6563 \end{pmatrix}$$

$$S = \begin{pmatrix} 14.9359 & 0 \\ 0 & 5.1883 \end{pmatrix}$$

$$V =$$

$$\begin{bmatrix} -0.6925 & 0.7214 \\ -0.7214 & -0.6925 \end{bmatrix}$$

Как и в первом случае, матрица U^*S^*V' равна A с точностью до ошибок округления.

Полиномы и интерполяция

В этом разделе мы ознакомимся с основными функциями MATLAB-а, которые дают возможность осуществлять математические действия с полиномами и производить интерполяцию одно-, двух-, и многомерных данных.

Полиномы и действия над ними

Обзор полиномиальных функций

Функция	Описание
conv	Умножение полиномов.
deconv	Деление полиномов.
poly	Вычисление характеристического полинома матрицы или определение полинома с заданными корнями.
polyder	Вычисление производных от полиномов.
polyfit	Аппроксимация данных полиномом.
polyval	Вычисление значений полиномов в заданных точках.
polyvalm	Вычисление значений матричного полинома.
residue	Разложение на простые дроби (вычисление вычетов).
roots	Вычисление корней полинома.

Представление полиномов

MATLAB представляет полиномы как векторы-строки, содержащие коэффициенты полиномов по убывающим степеням. Например, рассмотрим следующее уравнение

$$p(x) = x^3 - 2x - 5$$

Это известный пример Валлиса (Wallis), использованный при первом представлении метода Ньютона во Французкой Академии. Мы будем использовать его в дальнейшем при рассмотрении примеров использования различных функций. Для ввода данного полинома в MATLAB, следует записать

$$p = [1 \ 0 \ -2 \ -5].$$

Корни полинома

Корни полинома вычисляются при помощи функции **roots** :

```
r = roots(p)
r =
    2.0946
   -1.0473 + 1.1359i
   -1.0473 - 1.1359i
```

MATLAB запоминает вычисленные корни как *вектор-столбец*. Функция **poly** выполняет обратную роль, то есть по заданным корням полинома вычисляет значения его коэффициентов (обратите внимание на значение второго коэффициента, который в идеале равен нулю).

```
p2 = poly(r)
p2 =
    1  8.8818e-16  -2  -5
```

Функции **poly** и **roots** являются взаимно-обратными функциями, с точностью до упорядочения коэффициентов, масштабирования и ошибок округления.

Характеристические полиномы

Функция **poly** вычисляет также коэффициенты характеристического полинома матрицы:

```
A = [1.2  3  -0.9; 5  1.75  6; 9  0  1];
poly(A)
ans =
    1.0000  -3.9500  -1.8500  -163.2750
```

Корни данного полинома, вычисленные при помощи функции **roots**, являются *собственными значениями* (*характеристическими числами*) матрицы **A**. (При практических расчетах, для вычисления собственных значений матриц целесообразно вычислять их посредством функции **eig**.)

Вычисление значений полинома

Функция **polyval** вычисляет значение полинома в заданных точках. Для вычисления **p** в точке **s = 5**, следует записать

```
polyval(p,5)
ans =
    110
```

Можно также вычислить значение матричного полинома. Так, вместо полинома Валлиса можно записать:

$$p(X) = X^3 - 2X - 5I$$

где **X** является квадратной матрицей, а **I** - единичной матрицей. Например, сформируем следующую квадратную матрицу **X**

$$\mathbf{X} = \begin{bmatrix} 2 & 4 & 5 & -1 & 0 & 3 \\ 7 & 1 & 5 \end{bmatrix};$$

и вычислим значение заданного выше полинома **p(X)** на данной матрице.

$$\mathbf{Y} = \text{polyvalm}(\mathbf{p}, \mathbf{X})$$

$$\mathbf{Y} = \begin{bmatrix} 377 & 179 & 439 \\ 111 & 81 & 136 \\ 490 & 253 & 639 \end{bmatrix}$$

Умножение и деление полиномов

Для умножения и деления полиномов предназначены соответственно функции **conv** и **deconv**. Рассмотрим полиномы $\mathbf{a}(s) = s^2 + 2s + 3$ и $\mathbf{b}(s) = 4s^2 + 5s + 6$. Для вычисления их произведения следует ввести

$$\mathbf{a} = [1 \ 2 \ 3]; \quad \mathbf{b} = [4 \ 5 \ 6];$$

$$\mathbf{c} = \text{conv}(\mathbf{a}, \mathbf{b})$$

MATLAB возвращает

$$\mathbf{c} = \begin{bmatrix} 4 & 13 & 28 & 27 & 18 \end{bmatrix}$$

Для получения из **c** полинома **b** воспользуемся функцией **deconv**:

$$[\mathbf{q}, \mathbf{r}] = \text{deconv}(\mathbf{c}, \mathbf{a})$$

$$\mathbf{q} = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$$

$$\mathbf{r} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

где **r** – остаток после деления (в данном случае нулевой вектор). В общем случае для полиномов **q**, **r**, **c**, **a** в функции **deconv** справедливо соотношение

$$\mathbf{c} = \text{conv}(\mathbf{q}, \mathbf{a}) + \mathbf{r}$$

Вычисление производных от полиномов

Функция **polyder** вычисляет производную любого полинома. Для получения производной от нашего полинома **p** = [1 0 -2 -5], введем

$$\mathbf{q} = \text{polyder}(\mathbf{p})$$

$$\mathbf{q} = \begin{bmatrix} 3 & 0 & -2 \end{bmatrix}$$

Функция **polyder** вычисляет также производные от произведения или частного двух полиномов. Например, создадим два полинома **a** и **b**:

a = [1 3 5]; **b** = [2 4 6];

Вычислим производную *произведения* **a*b** вводом функции **polyder** с *двумя* входными аргументами **a** и **b** и *одним* выходным:

c = polyder(**a**, **b**)

c =
8 30 56 38

Вычислим производную от *частного* **a/b** путем ввода функции **polyder** с *двумя* выходными аргументами:

[**q**, **d**] = polyder(**a**, **b**)

q =
-2 -8 -2
d =
4 16 40 48 36

где отношение двух полиномов **q/d** является результатом операции дифференцирования.

Аппроксимация кривых полиномами

Функция **polyfit** находит коэффициенты полинома заданной степени **n**, который аппроксимирует данные (или функцию **y(x)**) в смысле метода наименьших квадратов:

p = polyfit(**x**, **y**, **n**)

где **x** и **y** есть векторы, содержащие данные **x** и **y**, которые нужно аппроксимировать полиномом. Например, рассмотрим совокупность данных **x-y**, полученную экспериментальным путем

x = [1 2 3 4 5]; **y** = [5.5 43.1 128 290.7 498.4].

Аппроксимация функциональной зависимости **y(x)** в виде полинома третьего порядка

p = polyfit(**x**,**y**,3)

дает коэффициенты полинома

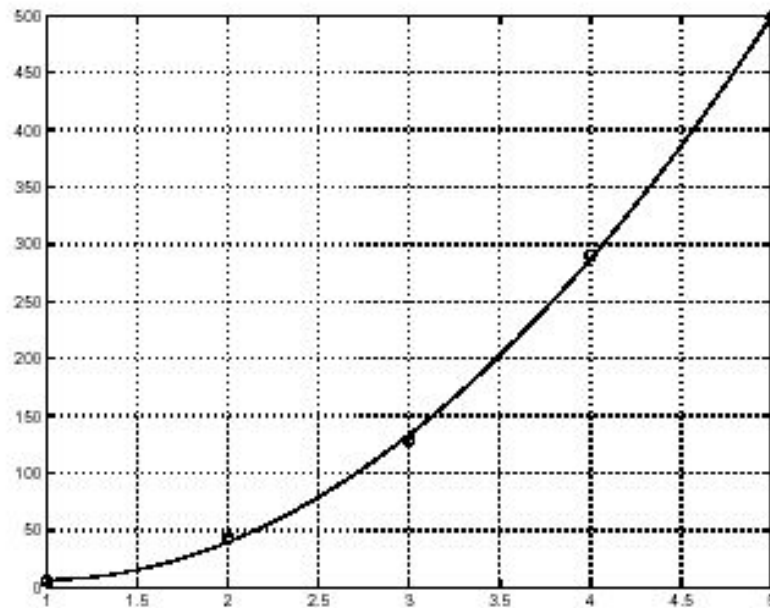
p =
-0.1917 31.5821 -60.3262 35.3400

Рассчитаем теперь значения полинома, полученного при помощи функции **polyfit**, на более мелкой шкале (с шагом 0.1) и построим для сравнения графики (это делает функция **plot**) реальных данных и аппроксимирующей кривой.

x2 = 1 : 0.1 : 5;

```
y2 = polyval(p, x2);
plot(x, y, 'o', x2, y2); grid on
```

где функция **grid on** служит для нанесения координатной сетки, а экспериментальные данные на графике отмечены маркерами **o**.



Как видно из рисунка, полином третьего порядка достаточно хорошо аппроксимирует наши данные.

Разложение на простые дроби

Функция **residue** вычисляет вычеты, полюса и многочлен целой части отношения двух полиномов. Это особенно полезно при представлении систем управления в виде передаточных функций. Для полиномов $a(s)$ и $b(s)$, при отсутствии кратных корней имеем

$$\frac{b(s)}{a(s)} = \frac{r_1}{s - p_1} + \frac{r_2}{s - p_2} + \dots + \frac{r_n}{s - p_n} + k(s)$$

где r есть вектор-столбец *вычетов*, p есть вектор-столбец *полюсов*, а k есть вектор-строка *целой части* дробно-рациональной функции. Рассмотрим передаточную функцию

$$W(p) = \frac{-4 + 8s^{-1}}{1 + 6s^{-1} + 8s^{-2}}$$

Для полиномов числителя и знаменателя этой функции имеем:

$$\mathbf{b} = [-4 \ 8]; \quad \mathbf{a} = [1 \ 6 \ 8].$$

Введя

$$[\mathbf{r}, \mathbf{p}, \mathbf{k}] = \text{residue}(\mathbf{b}, \mathbf{a})$$

получим

```

r =
    -12
     8
p =
    -4
    -2
k =
     [ ]

```

Функция **residue** с тремя входными (**r**, **p**, и **k**) и двумя выходными (**b2**, **a2**) аргументами выполняет обратную функцию свертки имеющегося разложения на простые дроби, в дробно-рациональную функцию отношения двух полиномов.

```
[b2, a2] = residue(r, p, k)
```

```

b2 =
    -4     8
a2 =
     1     6     8

```

т.е. из данных предыдущего примера мы восстановили исходную передаточную функцию. В случае кратных корней процедура несколько усложняется, но остается разрешимой.

Интерполяция

Интерполяция является процессом вычисления (оценки) промежуточных значений функций, которые находятся между известными или заданными точками. Она имеет важное применение в таких областях как теория сигналов, обработка изображений и других. MATLAB обеспечивает ряд интерполяционных методик, которые позволяют находить компромисс между точностью представления интерполируемых данных и скоростью вычислений и используемой памятью.

Обзор функций интерполяции

Функции	Описание
griddata	Двумерная интерполяция на неравномерной сетке.
griddata3	Трёхмерная интерполяция на неравномерной сетке.
griddatan	Многомерная интерполяция ($n \geq 3$).
interp1	Одномерная табличная интерполяция.
interp2	Двухмерная табличная интерполяция.
interp3	Трёхмерная табличная интерполяция.
interpft	Одномерная интерполяция с использованием быстрого преобразования Фурье.
interp	Многомерная табличная интерполяция.
pchip	Кубическая интерполяция при помощи полинома Эрмита.
spline	Интерполяция кубическим сплайном.

Одномерная интерполяция

Двумя основными типами одномерной интерполяции в MATLAB-е являются полиномиальная интерполяция и интерполяция на основе быстрого преобразования Фурье.

1. Полиномиальная интерполяция

Функция **interp1** осуществляет одномерную интерполяцию – важную операцию в области анализа данных и аппроксимации кривых. Эта функция использует полиномиальные методы, аппроксимируя имеющийся массив данных полиномиальными функциями и вычисляя соответствующие функции на заданных (желаемых) точках. В наиболее общей форме эта функция имеет вид

$$y_i = \text{interp1}(x, y, x_i, \text{method})$$

где **y** есть вектор, содержащий значения функции; **x** – вектор такой же длины, содержащий те точки (значения аргумента), в которых заданы значения **y**; вектор **xi** содержит те точки, в которых мы хотим найти значения вектора **y** путем интерполяции; **method** – дополнительная строка, задающая метод интерполяции. Имеются следующие возможности для выбора метода:

- *Ступенчатая интерполяция* (**method** = 'nearest'). Этот метод приравнивает значение функции в интерполируемой точке к ее значению в ближайшей существующей точке имеющихся данных.
- *Линейная интерполяция* (**method** = 'linear'). Этот метод аппроксимирует функцию между любыми двумя существующими соседними значениями как линейную функцию, и возвращает соответствующее значение для точки в **xi** (метод используется по умолчанию).
- *Интерполяция кубическими сплайнами* (**method** = 'spline'). Этот метод аппроксимирует интерполируемую функцию между любыми двумя соседними значениями при помощи кубических функций, и использует сплайны для осуществления интерполяции.
- *Кубическая интерполяция* (**method** = 'pchip' или 'cubic'). Эти методы идентичны. Они используют кусочную кубическую Эрмитову аппроксимацию и сохраняют монотонность и форму данных.

Если какой-либо из элементов вектора **xi** находится вне интервала, заданного вектором **x**, то выбранный метод интерполяции используется также и для экстраполяции. Как альтернатива, функция **yi = interp1(x, y, xi, method, extrapolval)** заменяет экстраполированные значения теми, которые заданы вектором **extrapval**. Для последнего часто используется нечисловое значение **NaN**.

Все методы работают на неравномерной сетке значений вектора **x**.

Рассмотрение скорости, требуемой памяти и гладкости методов. При выборе метода интерполяции всегда нужно помнить, что некоторые из них требуют большего объема памяти или выполняются быстрее, чем другие. Однако, вам может потребоваться использование любого из этих методов, чтобы достичь нужной степени точности интерполяции (гладкости результатов). При этом нужно исходить из следующих критериев.

- Метод ступенчатой аппроксимации является самым быстрым, однако он дает наихудшие результаты с точки зрения гладкости.
- Линейная интерполяция использует больше памяти чем ступенчатая и требует несколько большего времени исполнения. В отличие от ступенчатой аппроксимации, результирующая функция является непрерывной, но ее наклон меняется в значениях исходной сетки (исходных данных).
- Кубическая интерполяция сплайнами требует наибольшего времени исполнения, хотя требует меньших объемов памяти чем кубическая интерполяция. Она дает самый гладкий результат из всех других методов, однако вы можете получить неожиданные результаты, если входные данные распределены неравномерно и некоторые точки слишком близки.

- Кубическая интерполяция требует большей памяти и времени исполнения чем ступенчатая или линейная. Однако в данном случае как интерполируемые данные, так и их производные являются непрерывными.

Относительные качественные характеристики всех перечисленных методов сохраняются и в случае двух- или многомерной интерполяции.

2. Интерполяция на основе быстрого преобразования Фурье _

Функция **interpft** осуществляет одномерную интерполяцию с использованием быстрого преобразования Фурье (FFT). Этот метод вычисляет преобразование Фурье от вектора, который содержит значения периодической функции. Затем вычисляется обратное преобразование Фурье с использованием большего числа точек. Функция записывается в форме

$$y = \text{interpft}(x, n)$$

где **x** есть вектор, содержащий дискретные значения периодической функции, заданной на равномерной сетке, а **n** - число равномерно распределенных точек, в которых нужно оценить значения интерполируемой функции.

Двумерная интерполяция

Функция **interp2** осуществляет двумерную интерполяцию - важную операцию при обработке изображений и графического представления данных. В наиболее общей форме эта команда имеет вид

$$ZI = \text{interp2}(X, Y, Z, XI, YI, \text{method})$$

где **Z** есть прямоугольный массив, содержащий значения двумерной функции; **X** и **Y** являются массивами одинаковых размеров, содержащие точки в которых заданы значения двумерной функции; **XI** и **YI** есть матрицы, содержащие точки интерполяции (то есть промежуточные точки, в которых нужно вычислить значения функции); **method** – строка, определяющая метод интерполяции. В случае двумерной интерполяции возможны три различных метода:

- **Ступенчатая интерполяция** (**method = 'nearest'**). Этот метод дает кусочно-постоянную поверхность на области значений. Значение функции в интерполируемой точке равно значению функции в ближайшей заданной точке.

- **Билинейная интерполяция** (**method = 'linear'**). Метод обеспечивает аппроксимацию данных при помощи билинейной поверхности (плоскости) на множестве заданных значений двумерной функции. Значение в точке интерполяции является комбинацией значений четырех ближайших точек. Данный метод можно считать «кусочно-билинейным»; он быстрее и требует меньше памяти, чем бикубическая интерполяция.

- **Бикубическая интерполяция** (**method = 'cubic'**). Данный метод аппроксимирует поверхность при помощи бикубических поверхностей. Значение в точке интерполяции является комбинацией значений в шестнадцати ближайших точках. Метод обеспечивает значительно более гладкую поверхность по сравнению с билинейной интерполяцией. Это может быть ключевым преимуществом в приложениях типа обработки изображений. Особенно эффективным данный метод является в ситуациях, когда требуется непрерывность как интерполируемых данных, так и их производных.

Все эти методы требуют, чтобы **X** и **Y** были монотонными, то есть или всегда возрастающими или всегда убывающими от точки к точке. Эти матрицы следует сформировать с использованием функции **meshgrid**, или же, в противном случае, нужно убедиться, что «схема» то-

чек имитирует сетку, полученную функцией **meshgrid**. Перед интерполяцией, каждый из указанных методов автоматически отображает входные данные в равномерно распределенную сетку. Если **X** и **Y** уже распределены равномерно, вы можете ускорить вычисления добавляя звездочку к строке метода, например, **'*cubic'**.

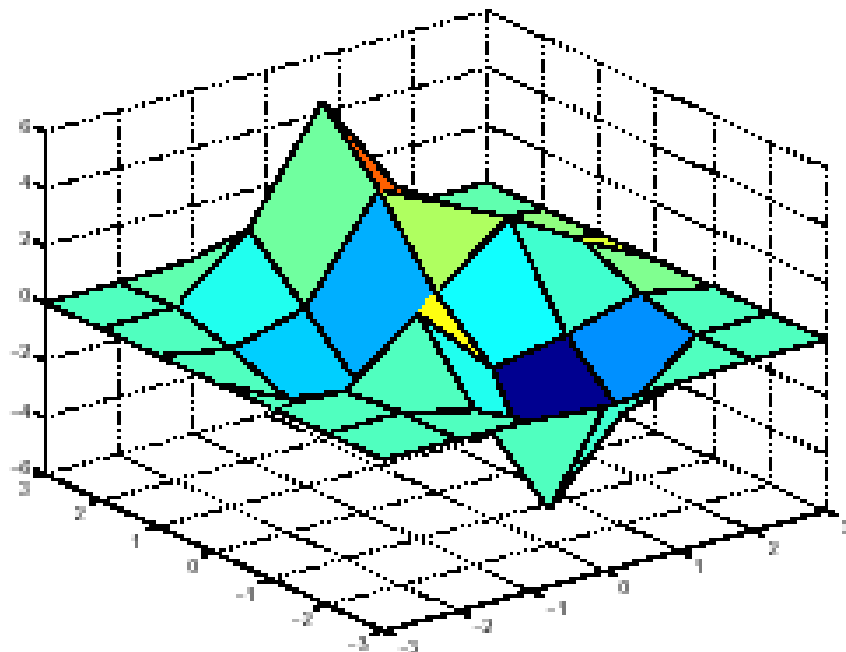
Сравнение методов интерполяции

Приведенный ниже пример сравнивает методы двумерной интерполяции в случае матрицы данных размера 7x7.

1. Сформируем функцию **peaks** на «грубой» сетке (с единичным шагом).

```
[x, y] = meshgrid(-3 : 1 : 3);  
z = peaks(x,y);  
surf(x,y,z)
```

где функция **meshgrid(-3:1:3)** задает сетку на плоскости **x** и **y** в виде двумерных массивов размера 7x7; функция **peaks(x,y)** является двумерной функцией, используемой в MATLAB-е в качестве стандартных примеров, а **surf(x,y,z)** строит окрашенную параметрическую поверхность. Соответствующий график показан ниже.



2. Создадим теперь более мелкую сетку для интерполяции (с шагом 0.25).

```
[xi,yi] = meshgrid(-3:0.25:3);
```

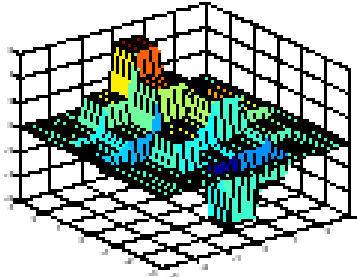
3. Осуществим интерполяция перечисленными выше методами.

```
zi1 = interp2(x,y,z,xi,yi,'nearest');
```

```
zi2 = interp2(x,y,z,xi,yi,'bilinear');
```

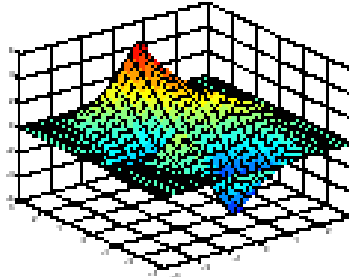
```
zi3 = interp2(x,y,z,xi,yi,'bicubic');
```

Сравним графики поверхностей для различных методов интерполяции.



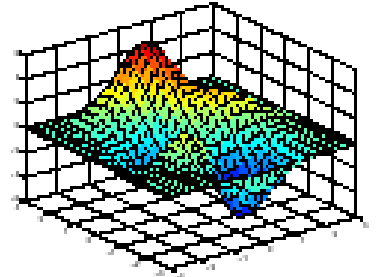
`surf(xi,yi,zi1)`

Метод **'nearest'**



`surf(xi,yi,zi2)`

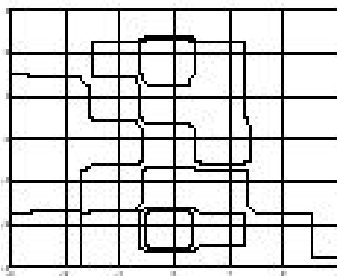
Метод **'bilinear'**



`surf(xi,yi,zi3)`

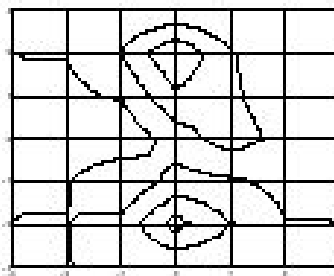
Метод **'bicubic'**

Интересно также сравнить линии уровней данных поверхностей, построенных при помощи специальной функции **contour**.



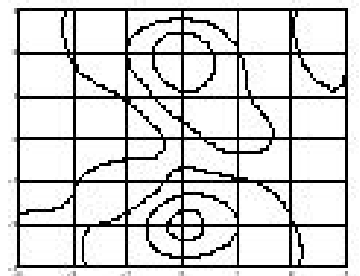
`contour(xi,yi,zi1)`

Метод **'nearest'**



`contour(xi,yi,zi2)`

Метод **'bilinear'**



`contour(xi,yi,zi3)`

Метод **'bicubic'**

Отметим, что бикубический метод производит обычно более гладкие контуры. Это, однако, не всегда является основной заботой. Для некоторых приложений, таких, например, как обработка изображений в медицине, метод типа ступенчатой интерполяции может быть более предпочтительным, так как он не «производит» никаких «новых» результатов наблюдений.

Анализ данных и статистика

В данном разделе будут рассмотрены некоторые основные возможности системы MATLAB в области анализа данных и статистической обработки информации. Помимо базовых функций, в системе MATLAB имеется также ряд специализированных пакетов, предназначенных для решения соответствующих задач в различных приложениях (на английском языке даны названия пакетов) :

- **Optimization** – Нелинейные методы обработки данных и оптимизация.
- **Signal Processing** – Обработка сигналов, фильтрация и частотный анализ.
- **Spline** – Аппроксимация сплайнами.

- **Statistics** – Углубленный статистический анализ, нелинейная аппроксимация и регрессия.
- **Wavelet** - Импульсная декомпозиция сигналов и изображений.

Внимание ! MATLAB выполняет обработку данных, записанных в виде двумерных массивов по столбцам ! Одномерные статистические данные обычно хранятся в отдельных векторах, причем n -мерные векторы могут иметь размерность $1 \times n$ или $n \times 1$. Для многомерных данных матрица является естественным представлением, но здесь имеются две возможности для ориентации данных. По принятому в системе MATLAB соглашению, различные переменные должны образовывать столбцы, а соответствующие наблюдения - строки. Поэтому, например, набор данных, состоящий из 24 выборок 3 переменных записывается в виде матрицы размера 24×3 .

Основные функции обработки данных

Перечень функций обработки данных, расположенных в директории MATLAB-a **datafun** приведен в **Приложении 8**.

Рассмотрим гипотетический числовой пример, который основан на ежечасном подсчете числа машин, проходящих через три различные пункта в течении 24 часов. Допустим, результаты наблюдений дают следующую матрицу **count**

count =

11	11	9
7	13	11
14	17	20
11	13	9
43	51	69
38	46	76
61	132	186
75	135	180
38	88	115
28	36	55
12	12	14
18	27	30
18	19	29
17	15	18
19	36	48
32	47	10
42	65	92
57	66	151
44	55	90
114	145	257
35	58	68
11	12	15
13	9	15
10	9	7

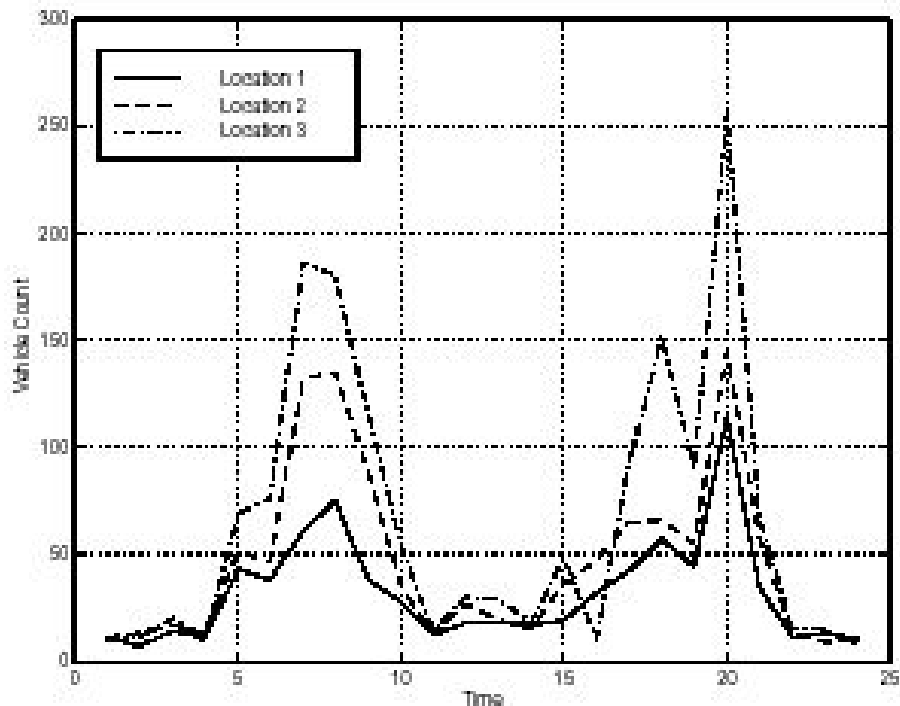
Таким образом, мы имеем 24 наблюдения трех переменных. Создадим вектор времени, **t**, состоящий из целых чисел от 1 до 24: **t = 1 : 24**. Построим теперь зависимости столбцов матрицы **counts** от времени и напомним график:

```

plot(t, count)
legend('Location 1','Location 2','Location 3',0)
xlabel('Time')
ylabel('Vehicle Count')
grid on

```

где функция **plot(t, count)** строит зависимости трех векторов-столбцов от времени; функция **legend('Location 1','Location 2','Location 3',0)** показывает тип кривых; функции **xlabel** и **ylabel** надписывают координатные оси, а **grid on** выводит координатную сетку. Соответствующий график показан ниже.



Применим к матрице **count** функции **max** (максимальное значение), **mean** (среднее значение) и **std** (стандартное, или среднеквадратическое отклонение).

```

mx = max(count)
mu = mean(count)
sigma = std(count)

```

В результате получим

```

mx =
    114    145    257

mu =
    32.00    46.5417    65.5833

sigma =
    25.3703    41.4057    68.0281

```

где каждое число в строке ответов есть результат операции вдоль соответствующего столбца матрицы **count**. Для определения индекса максимального или минимального элемента нужно в соответствующей функции задать второй выходной параметр. Например, ввод

```
[mx,indx] = min(count)
```

```
mx =
```

```
7    9    7
```

```
indx =
```

```
2   23   24
```

показывает, что наименьшее число машин за час было зарегистрировано в 2 часа для первого пункта наблюдения (первый столбец) и в 23 и 24 чч. для остальных пунктов наблюдения. Вы можете вычесть среднее значение из каждого столбца данных, используя внешнее произведение вектора, составленного из единиц и вектора **mu** (вектора средних значений)

```
e = ones(24, 1)
```

```
x = count - e*mu
```

Перегруппировка данных может помочь вам в оценке всего набора данных. Так, использование в системе MATLAB в качестве единственного индекса матрицы двоеточия, приводит к представлению этой матрицы как одного длинного вектора, составленного из ее столбцов. Поэтому, для нахождения минимального значения всего множества данных можно ввести

```
min(count(:))
```

что приводит к результату

```
ans =
```

```
7
```

Запись **count(:)** в данном случае привела к перегруппировке матрицы размера 24x3 в вектор-столбец размера 72x1.

Матрица ковариаций и коэффициенты корреляции

Для статистической обработки в MATLAB-е имеются две основные функции для вычисления ковариации и коэффициентов корреляции:

- **cov** – В случае вектора данных эта функция выдает дисперсию, то есть меру распределения (отклонения) наблюдаемой переменной от ее среднего значения. В случае матриц это также мера линейной зависимости между отдельными переменными, определяемая недиагональными элементами.
- **corrcoef** – Коэффициенты корреляции – нормализованная мера линейной вероятностной зависимости между переменными.

Применим функцию **cov** к первому столбцу матрицы **count**

```
cov(count(:,1))
```

Результатом будет дисперсия числа машин на первом пункте наблюдения

```
ans =
```

```
643.6522
```

Для массива данных, функция **cov** вычисляет матрицу ковариаций. Дисперсии столбцов массива данных при этом расположены на главной диагонали матрицы ковариаций. Остальные элементы матрицы характеризуют ковариацию между столбцами исходного массива. Для

матрицы размера $m \times n$, матрица ковариаций имеет размер $n \times n$ и является симметричной, то есть совпадает с транспонированной.

Функция **corrcoef** вычисляет матрицу коэффициентов корреляции для массива данных, где каждая строка есть наблюдение, а каждый столбец – переменная. *Коэффициент корреляции* – это нормализованная мера линейной зависимости между двумя переменными. Для некоррелированных (линейно-независимых) данных коэффициент корреляции равен нулю; эквивалентные данные имеют единичный коэффициент корреляции. Для матрицы $m \times n$, соответствующая матрица коэффициентов корреляции имеет размер $n \times n$. Расположение элементов в матрице коэффициентов корреляции аналогично расположению элементов в рассмотренной выше матрице ковариаций. Для нашего примера подсчета количества машин, при вводе

corrcoef(count)

получим

```
ans =
    1.0000    0.9331    0.9599
    0.9331    1.0000    0.9553
    0.9599    0.9553    1.0000
```

Очевидно, здесь имеется сильная линейная корреляция между наблюдениями числа машин в трех различных точках, так как результаты довольно близки к единице.

Конечные разности

MATLAB предоставляет три функции для вычисления конечных разностей.

Функция	Описание
diff	Разность между двумя последовательными элементами вектора. Приближенное дифференцирование.
gradient	Приближенное вычисление градиента функции.
del2	Пятиточечная аппроксимация Лапласиана.

Функция **diff** вычисляет разность между последовательными элементами числового вектора, то есть **diff(X)** есть $[X(2) - X(1) \ X(3) - X(2) \ \dots \ X(n) - X(n-1)]$. Так, для вектора **A**,

```
A = [9 -2 3 0 1 5 4];
diff(A)
```

MATLAB возвращает

```
ans =
   -11    5   -3    1    4   -1
```

Помимо вычисления первой разности, функция **diff** является полезной для определения определенных характеристик вектора. Например, вы можете использовать **diff** для определения, является ли вектор монотонным (значения элементов или всегда возрастают или убывают), или имеет ли он равные приращения и т.д. Следующая таблица описывает несколько различных путей использования функции **diff** с одномерным вектором **x**.

Применение (тест)	Описание
diff(x) == 0	Тест на определение повторяющихся элементов
all(diff(x) > 0)	Тест на монотонность
all(diff(diff(x)) == 0)	Тест на определение равных приращений

Обработка данных

В данном разделе рассматривается как поступать с:

- Отсутствующими значениями
- Выбросами значений или несовместимыми («неуместными») значениями

Отсутствующие значения

Специальное обозначение NaN, соответствует в MATLAB-е нечисловое значение. В соответствие с принятыми соглашениями NaN является результатом неопределенных выражений таких как 0/0. Надлежащее обращение с отсутствующими данными является сложной проблемой и зачастую меняется в различных ситуациях. Для целей анализа данных, часто удобно использовать NaN для представления отсутствующих значений или данных которые *недоступны*. MATLAB обращается со значениями NaN единообразным и строгим образом. Эти значения сохраняются в процессе вычислений вплоть до конечных результатов. Любое математическое действие, производимое над значением NaN, в результате также производит NaN. Например, рассмотрим матрицу, содержащую волшебный квадрат размера 3x3, где центральный элемент установлен равным NaN.

```
a = magic(3); a(2,2) = NaN;
```

```
a =  
      8      1      6  
      3 NaN      7  
      4      9      2
```

Вычислим сумму элементов всех столбцов матрицы:

```
sum(a)  
ans =  
    15 NaN    15
```

Любые математические действия над NaN распространяют NaN вплоть до конечного результата. Перед проведением любых статистических вычислений вам следует удалить все NaN-ы из имеющихся данных. Вот некоторые возможные пути выполнения данной операции.

Программа	Описание
i = find(~ isnan(x)); x = x(i)	Найти индексы всех элементов вектора, не равных NaN, и затем сохранить только эти элементы
x = x (find(~ isnan(x)))	Удалить все NaN-ы из вектора
x = x (~ isnan(x));	Удалить все NaN-ы из вектора (быстрее).
x (isnan(x)) = [];	Удалить все NaN-ы из вектора
X (any(isnan(X')), :) = [];	Удалить все строки матрицы X содержащие NaN-ы

Внимание. Для нахождения нечисловых значений NaN вам следует использовать специальную функцию **isnan**, поскольку при принятом в MATLAB-е соглашении, логическое сравнение NaN == NaN всегда выдает 0. Вы не можете использовать запись **x(x==NaN) = []** для удаления NaN-ов из ваших данных.

Если вам часто приходится удалять NaN-ы, воспользуйтесь короткой программой, записанной в виде М-файла.

```
function X = excise(X)
X(any(isnan(X')), :) = [ ];
```

Тогда, напечатав

```
X = excise(X);
```

вы выполните требуемое действие (*excise* по английски означает *вырезать*)

Удаление выбросов значений

Вы можете удалить выбросы значений или несовместимые данные при помощи процедур, весьма схожих с удалением NaN-ов. Для нашей транспортной задачи, с матрицей данных **count**, средние значения и стандартные (среднеквадратические) отклонения каждого столбца матрицы **count** равны

```
mu = mean(count)
```

```
sigma = std(count)
```

```
mu =
```

```
32.0000 46.5417 65.5833
```

```
sigma =
```

```
25.3703 41.4057 68.0281
```

Число строк с выбросами значений, превышающими утроенное среднеквадратическое отклонение от среднего значения можно получить следующим образом:

```
[n, p] = size(count)
```

```
outliers = abs(count - mu(ones(n, 1), :)) > 3*sigma(ones(n, 1), :);
```

```
nout = sum(outliers)
```

```
nout =
```

```
1 0 0
```

Имеется только один выброс в первом столбце. Удалим все наблюдение при помощи выражения

```
count(any(outliers'), :) = [ ];
```

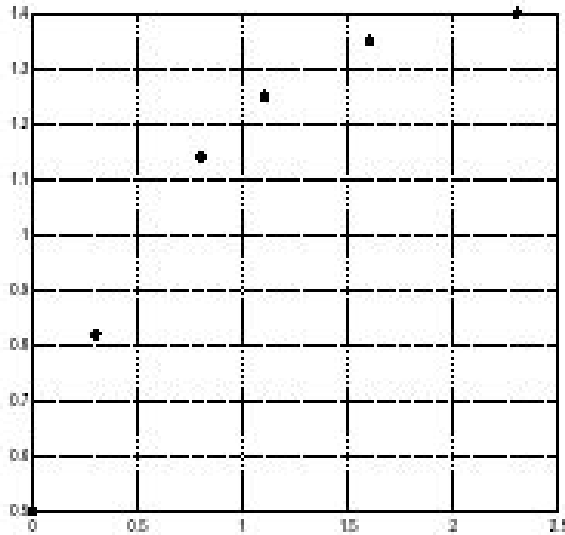
Регрессия и подгонка кривых

Часто бывает полезным или необходимым найти функцию, которая описывает взаимосвязь между некоторыми наблюдаемыми (или найденными экспериментально) переменными. Определение коэффициентов такой функции ведет к решению задачи переопределенной системы линейных уравнений, то есть системы, у которой число уравнений превышает число неизвестных. Указанные коэффициенты можно легко найти с использованием оператора обратного деления \ (*backslash*). Допустим, вы производили измерения переменной **y** при разных значениях времени **t**.


```
t = [0 0.3 0.8 1.1 1.6 2.3]';
```

```
y = [0.5 0.82 1.14 1.25 1.35 1.40]';
```

```
plot(t,y,'o'); grid on
```



В следующих разделах мы рассмотрим три способа моделирования (аппроксимации) этих данных:

- Методом полиномиальной регрессии
- Методом линейно-параметрической (linear-in-the-parameters) регрессии
- Методом множественной регрессии

Полиномиальная регрессия

Основываясь на виде графика, можно допустить, что данные могут быть аппроксимированы полиномиальной функцией второго порядка:

$$y = a_0 + a_1 t + a_2 t^2$$

Неизвестные коэффициенты a_0 , a_1 и a_2 могут быть найдены *методом среднеквадратической подгонки (аппроксимации)*, которая основана на минимизации суммы квадратов отклонений данных от модели. Мы имеем шесть уравнений относительно трех неизвестных,

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \\ 1 & t_4 & t_4^2 \\ 1 & t_5 & t_5^2 \\ 1 & t_6 & t_6^2 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

представляемых следующей матрицей 6x3:

```
X = [ones(size(t)) t t.^2]
```

```
X =      1.0000    0    0
      1.0000    0.3000    0.0900
      1.0000    0.8000    0.6400
      1.0000    1.1000    1.2100
      1.0000    1.6000    2.5600
      1.0000    2.3000    5.2900
```

Решение находится при помощи оператора \ :

```
a = X\y
```

```
a =
      0.5318
      0.9191
     -0.2387
```

Следовательно, полиномиальная модель второго порядка наших данных будет иметь вид

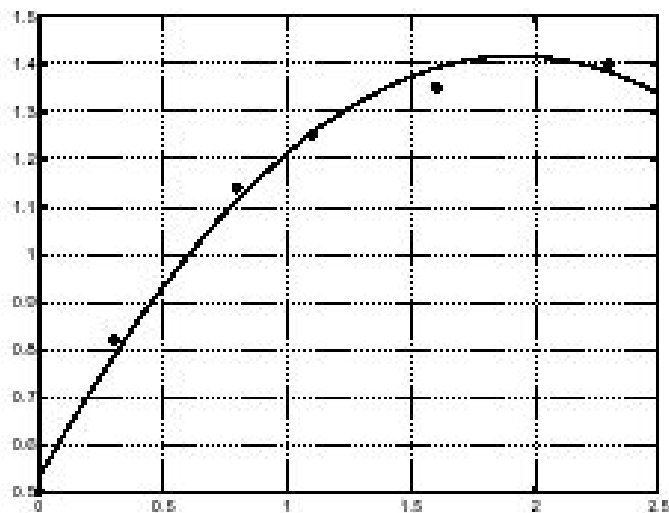
$$y = 0.5318 + 0.9191t - 0.2387 t^2$$

Оценим теперь значения модели на равноотстоящих точках (с шагом 0.1) и нанесем кривую на график с исходными данными.

```
T = (0 : 0.1 : 2.5)';
```

```
Y = [ones(size(T)) T T.^2]*a;
```

```
plot(T,Y,'-',t,y,'o'); grid on
```



Очевидно, полиномиальная аппроксимация оказалась не столь удачной. Здесь можно или повысить порядок аппроксимирующего полинома, или попытаться найти какую-либо другую функциональную зависимость для получения лучшей подгонки.

Линейно-параметрическая регрессия¹

Вместо полиномиальной функции, можно было-бы попробовать так называемую линейно-параметрическую функцию. В данном случае, рассмотрим экспоненциальную функцию

$$y = a_0 + a_1 e^{-t} + a_2 t e^{-t}$$

Здесь также, неизвестные коэффициенты a_0 , a_1 и a_2 могут быть найдены методом *наименьших квадратов*. Составим и решим систему совместных уравнений, сформировав регрессионную матрицу X , и применив для определения коэффициентов оператор \backslash :

$$X = [\text{ones}(\text{size}(t)) \quad \exp(-t) \quad t.*\exp(-t)];$$

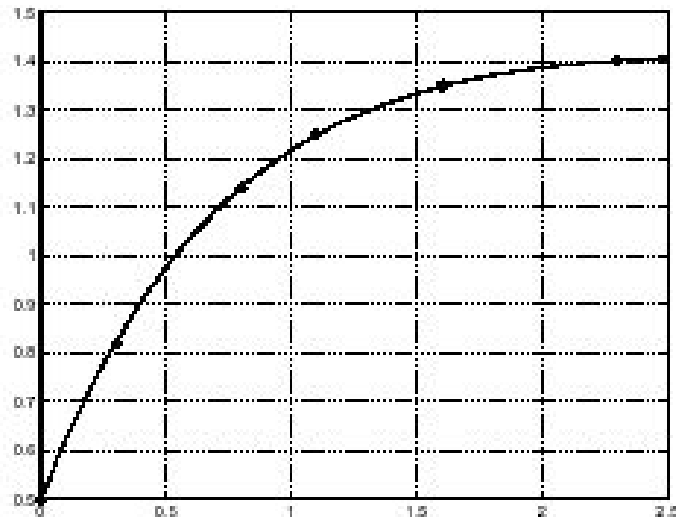
$$a = X \backslash y$$

$$a = \begin{array}{r} 1.3974 \\ - 0.8988 \\ 0.4097 \end{array}$$

Значит, наша модель данных имеет вид

$$y = 1.3974 - 0.8988 e^{-t} + 0.4097 t e^{-t}$$

Оценим теперь, как и раньше, значения модели на равноотстоящих точках (с шагом 0.1) и нанесем эту кривую на график с исходными данными.



Как видно из данного графика, подгонка здесь намного лучше чем в случае полиномиальной функции второго порядка.

¹ Данная терминология не совсем соответствует принятой в русско-язычных изданиях.

Множественная регрессия

Рассмотренные выше методы аппроксимации данных можно распространить и на случай более чем одной независимой переменной, за счет перехода к расширенной форме записи. Допустим, мы измерили величину y для некоторых значений двух параметров x_1 и x_2 и получили следующие результаты

$$x1 = [0.2 \ 0.5 \ 0.6 \ 0.8 \ 1.0 \ 1.1]';$$

$$x2 = [0.1 \ 0.3 \ 0.4 \ 0.9 \ 1.1 \ 1.4]';$$

$$y = [0.17 \ 0.26 \ 0.28 \ 0.23 \ 0.27 \ 0.24]';$$

Множественную модель данных будем искать в виде

$$y = a_0 + a_1x_1 + a_2x_2$$

Методы множественной регрессии решают задачу определения неизвестных коэффициентов a_0 , a_1 и a_2 путем минимизации *среднеквадратической ошибки* приближения. Составим совместную систему уравнений, сформировав матрицу регрессии X и решив уравнения относительно неизвестных коэффициентов, применяя оператор \backslash .

$$X = [\text{ones}(\text{size}(x1)) \ x1 \ x2];$$

$$a = X \backslash y$$

$$a = \begin{array}{c} 0.1018 \\ 0.4844 \\ -0.2847 \end{array}$$

Следовательно, модель дающая минимальную среднеквадратическую ошибку аппроксимации имеет вид

$$y = 0.1018 + 0.4844x_1 - 0.2847x_2$$

Для проверки точности подгонки найдем максимальное значение абсолютного значения отклонений экспериментальных и расчетных данных.

$$Y = X * a;$$

$$\text{MaxErr} = \max(\text{abs}(Y - y))$$

$$\text{MaxErr} = 0.0038$$

Эта ошибка дает основание утверждать, что наша модель достаточно адекватно отражает результаты наблюдений.

Графический интерфейс подгонки кривых

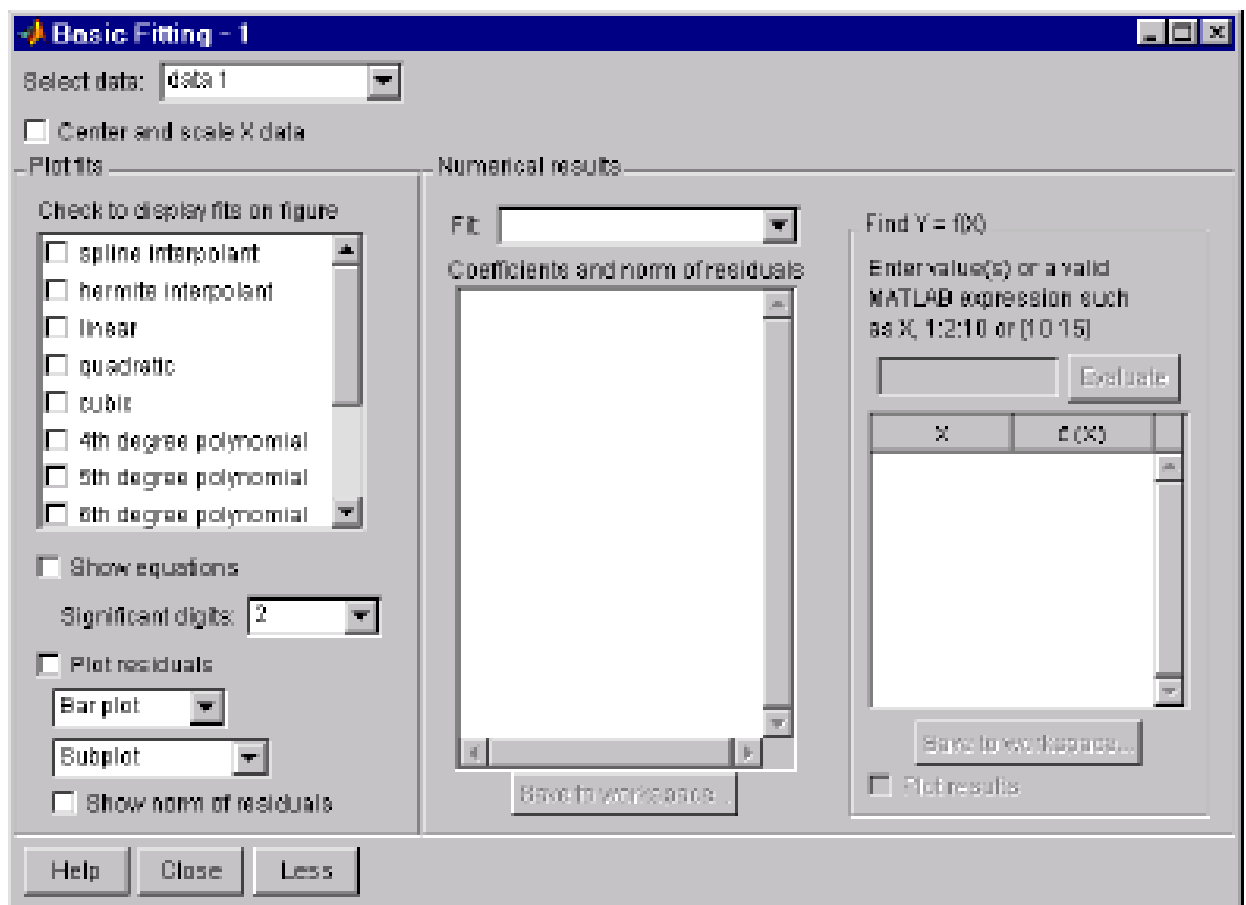
MATLAB дает возможность осуществлять аппроксимацию данных наблюдений при помощи специального графического Интерфейса Подгонки Кривых (ИПК) (в английском оригинале - **Basic Fitting interface**). Используя данный интерфейс, вы можете легко и быстро решить множество задач подгонки кривых, получая при этом самую разнообразную информацию о результатах вашей подгонки. ИПК предоставляет следующие возможности:

- Аппроксимирует данные используя сплайновый интерполянт, эрмитовый интерполянт, или же полиномиальный интерполянт до 10 порядка включительно.
- Осуществляет множество графических построений для заданных наборов данных.
- Строит графики невязок (ошибок подгонки).
- Анализирует численные результаты подгонки.
- Осуществляет интерполяцию или экстраполяцию данных подгонки.
- Аннотирует графики численными результатами подгонки и нормами ошибок аппроксимации.
- Запоминает результаты подгонки и вычислений в рабочее пространство MATLAB-а.

Основываясь на ваших конкретных задачах и приложениях, вы можете использовать ИПК, возможности, предоставляемые командным окном, или же комбинировать эти две возможности. Отметим, что ИПК предназначен только для работы с одномерными и двумерными данными.

Рассмотрение основных свойств ИПК

Общий вид ИПК показан ниже.



Для его вызова в подобном виде, нужно выполнить следующие три шага:

1. Построить какой либо график данных.
2. Выбрать опцию **Basic Fitting** из меню **Tools** вашего графического окна.
3. Нажать дважды на кнопку **More** в нижней части ИПК. В результате откроется окно с тремя панелями (см. рисунок), а сама надпись заменится на **Less**.

Рассмотрим основные опции ИПК.

Select data (Выбор данных) – В данном окне расположен список всех переменных, построенных на активном графике, с которым связан ИПК (на графике может быть построено несколько кривых). Используйте данный список для выбора требуемого (текущего) набора данных. Под текущим подразумевается тот набор данных, для которого вы хотите осуществить подгонку. За один раз вы можете осуществлять действия только с одним набором данных. С другой стороны, вы можете произвести различные подгонки для текущего набора данных за счет *изменения названия* этих данных. С этой целью можно воспользоваться так называемым Редактором Графиков (Plot Editor), который будет рассмотрен в дальнейшем.

Center and scale X data (Центрирование и масштабирование данных X) – Если данная опция выбрана, то данные центрируются (нуль переносится в среднее значение данных) и масштабируются к единичному стандартному отклонению (делятся на исходное стандартное отклонение). Это может потребоваться для повышения точности последующих математических вычислений. Если подгонка приводит к результатам, которые могут быть неточными, соответствующее предупреждение выводится на экран.

Plot fits (Подгонка кривых) – Эта панель позволяет визуально просмотреть результаты одной или более подгонок текущего набора данных.

- **Check to display fits on figure** (Отметьте методы для вывода на график) – Выберите методы подгонок, которые вы хотели бы использовать и *вывести* на график. Здесь имеются две основные возможности – выбор интерполянтов и выбор полиномов. Сплайновый интерполянт использует для аппроксимации сплайны, тогда как эрмитовый интерполянт использует специальную функцию **pchip** (Piecewise Cubic Hermite Interpolating Polynomial - Кусочно-кубический Эрмитовый Интерполяционный Полином). Полиномиальная подгонка использует функцию **polyfit**. Вы можете одновременно выбрать любые методы подгонки для аппроксимации ваших данных. Если ваш набор данных содержит N точек, вам следует использовать для аппроксимации полиномы с не более чем N коэффициентами. В противном случае, ИПК автоматически приравняет избыточное число коэффициентов нулю, что приводит к недоопределенности системы. Укажем, что при этом на дисплей выдается соответствующее сообщение.
- **Show equations** (Показать уравнения) – При выборе данной опции, уравнение подгонки выводится на ваш график.
- **Significant digits** (Значащие разряды) – Выберите число значащих разрядов для вывода на дисплей.
- **Plot residuals** (Построить графики разностей (невязок)) – При выборе данной опции, на график выводятся разности подгонок. Под разностью подгонки понимается разность между исходными данными и результатами подгонки для каждого значения аргумента исходных данных. Вы можете построить графики невязок как столбчатую диаграмму (bar plot), как график рассеяния (scatter plot), или же как линейный график. Построения можно осуществлять как в том же графическом окне, так и в отдельном. При использовании подграфиков (subplots) для построения графиков многомерных данных, графики разностей могут быть построены только в отдельном графическом окне.
- **Show norm of residuals** (Показать норму разностей) – При выборе опции, на график выводятся также значения норм разностей. Норма разности является мерой качества

подгонки, где меньшее значение нормы соответствует лучшему качеству. Норма рассчитывается при помощи функции **norm(V,2)**, где **V** есть вектор невязок.

Numerical results (Численные результаты) – Данная панель позволяет изучать численные характеристики каждой отдельной подгонки для текущего набора данных, без построения графиков.

- **Fit** (Метод подгонки) – Выберите метод подгонки. Соответствующие результаты будут представлены в окне под меню выбора метода. Заметим, что выбор метода в данной панели не оказывает воздействия на панель **Plot fits**. Поэтому, если вы хотите получить графическое представление, следует выбрать соответствующую опцию в панели **Plot fits**.
- **Coefficients and norm of residuals** (Коэффициенты и норма невязок) – В данном окне выводятся численные выражения для уравнения подгонки, выбранного в **Fit**. Отметим, что при первом открытии панели **Numerical Results**, в рассматриваемом окне выдаются результаты последней подгонки, выбранной вами в панели **Plot fits**.
- **Save to workspace** (Запомнить в рабочем пространстве) – Вызывает диалоговое окно, которое позволяет запомнить в рабочем пространстве результаты вашей подгонки.

Find Y = f(X) – Данная панель дает возможность произвести интерполяцию или экстраполяцию текущей подгонки.

- **Enter value(s)** (Введите данные) – Введите любое выражение, совместимое с системой MATLAB для оценки вашей текущей подгонки в промежуточных или выходящих за пределы заданных аргументов точек. Выражение будет вычислено после нажатия кнопки **Evaluate** (Вычислить), а результаты в табличной форме будут выведены в соответствующее окно ниже. Метод текущей подгонки при этом указан в меню **Fit**.
- **Save to workspace** (Запомнить в рабочем пространстве) – Вызывает диалоговое окно, которое позволяет запомнить в рабочем пространстве результаты вашей интерполяции.
- **Plot results** (Построить графики) – При выборе данной опции, результаты интерполяции выводятся в графической форме на график данных.

Уравнения в конечных разностях и фильтрация

MATLAB имеет специальные функции для работы с уравнениями в конечных разностях и фильтрами. Эти функции работают главным образом с векторами. Векторы используются для хранения дискретных сигналов или последовательностей, а также для обработки сигналов и анализа данных. Для систем со многими входами, каждая строка матрицы соответствует одной временной точке выборки сигналов, где каждый вход описывается как один вектор-столбец.

Функция

$$y = \text{filter}(\mathbf{b}, \mathbf{a}, \mathbf{x})$$

обрабатывает данные в векторе **x** посредством фильтра, описываемого векторами **a** и **b**, выдавая фильтрованные данные **y**. Функция **filter** может рассматриваться как эффективная реализация уравнения в конечных разностях. Структура функции **filter** является обобщенной структурой фильтра, образованного при помощи линий задержки, который описывается приведенными ниже уравнениями в конечных разностях, где **n** есть индекс (номер) текущей выборки, **na** есть порядок полинома, описываемого вектором **a**, а **nb** есть порядок полино-

ма, описываемого вектором **b**. Выход **y(n)** является линейной комбинацией текущего и предыдущих входов, то есть **x(n)** **x(n-1)** ..., и предыдущих выходов **y(n-1)** **y(n-2)** ...

$$a(1) y(n) = b(1) x(n) + b(2) x(n-1) + \dots + b(nb) x(n-nb+1) - a(2) y(n-1) - \dots - a(na) y(n-na+1)$$

Допустим, например, что мы хотим сгладить данные нашей задачи по движению автомобилей при помощи усредняющего фильтра, который выдает среднее количество машин за каждые 4 часа. Данный процесс можно выразить при помощи следующего уравнения в конечных разностях:

$$y(n) = (1/4) x(n) + (1/4) x(n-1) + (1/4) x(n-2) + (1/4) x(n-3)$$

Соответствующие векторы равны:

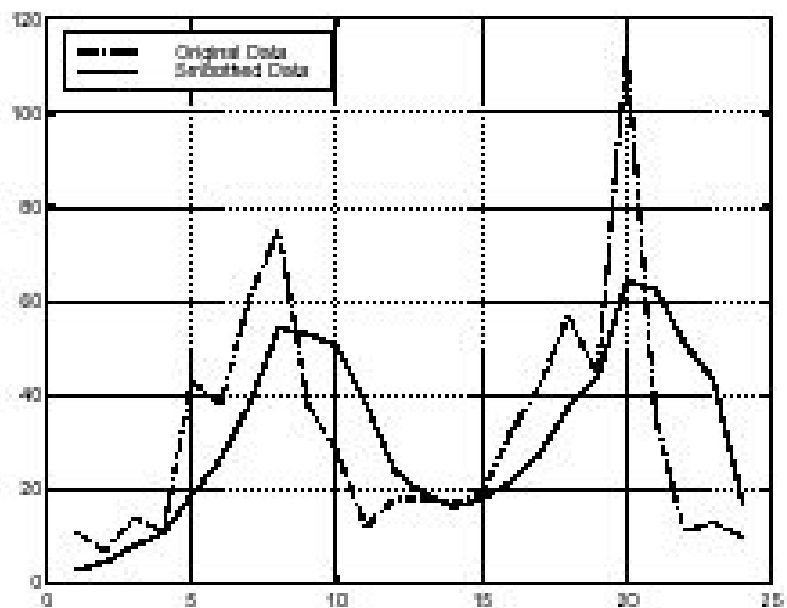
$$a = 1; \quad b = [1/4 \ 1/4 \ 1/4 \ 1/4];$$

Воспользуемся данными матрицы **count** из раздела **Анализ данных и статистика**. Для нашего примера, обозначим первый столбец матрицы **count** через вектор **x** :

$$x = \text{count}(:, 1);$$

Усредненные за 4 часа данные могут быть легко вычислены при помощи приведенной выше функции **y = filter(b, a, x)**. Сравним исходные и сглаженные данные, построив их на одном графике.

```
t = 1:length(x) ;
plot(t, x, '-.', t, y, '-');    grid on
legend('Original Data','Smoothed Data',2)
```

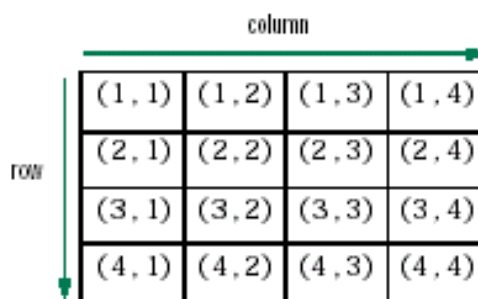


Исходные данные наблюдения представлены здесь штрих-пунктирной линией, а сглаженные за 4 часа данные – сплошной линией.

Для различных практических приложений, в специальном пакете **Signal Processing Toolbox** предусмотрены многочисленные функции для анализа сигналов и проектирования дискретных фильтров.

Многомерные Массивы

Многомерные массивы в системе MATLAB являются распространением обычных двумерных матриц. Как известно, матрицы имеют две размерности – строки (row) и столбцы (column).

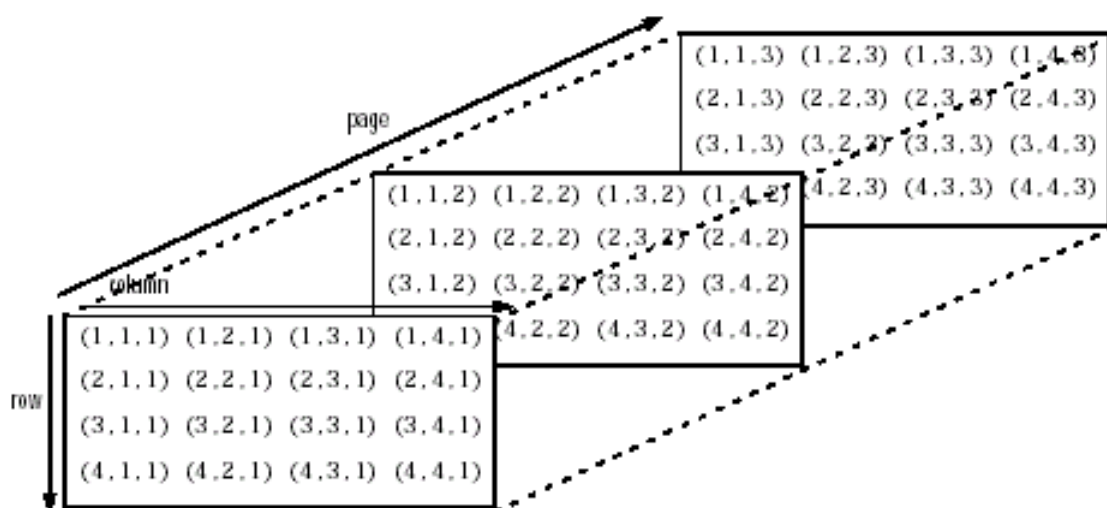


The diagram shows a 4x4 matrix with rows labeled 'row' and columns labeled 'column'. The elements are arranged in a grid with coordinates (row, column) in parentheses.

(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 1)	(3, 2)	(3, 3)	(3, 4)
(4, 1)	(4, 2)	(4, 3)	(4, 4)

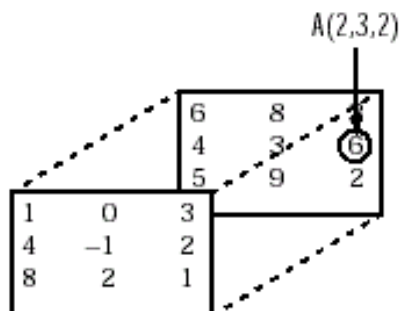
Вы можете выделить любой элемент двумерной матрицы при помощи двух индексов, где первый является индексом (номером) строки, а второй – индексом столбца. Многомерные массивы имеют дополнительную индексацию. Например, трехмерные массивы имеют три индекса:

- Первый индекс указывает размерность 1, то есть строки.
- Второй индекс указывает размерность 2, то есть столбцы.
- Третий индекс указывает на размерность 3. В данном пособии используется концепция *страницы* (page) для представления размерности 3 и выше.



Для обращения, например, к элементу второй строки и третьего столбца на странице 2 нужно воспользоваться индексацией (2,3,2) (см. рисунок ниже).

Если вы добавляете размерности к массиву, вы также добавляете индексы. Четырехмерный массив, например, имеет 4 индекса. Первые два из них указывают на пару строка-столбец, а следующие два характеризуют третью и четвертую размерности.



$$A(:, :, 1) =$$

1	0	3
4	-1	2
8	2	1

$$A(:, :, 2) =$$

6	8	3
4	3	6
5	9	2

Отметим, что общие функции обращения с многомерными массивами находятся в директории **datatypes**.

Создание Многомерных Массивов

При создании многомерных массивов можно воспользоваться теми же приемами, которые используются для двумерных матриц.

Создание массивов с использованием индексации

Один из способов формирования многомерного массива состоит в создании двумерного массива и соответствующего его расширения. Например, начнем с простого двумерного массива **A**.

$$A = [5 \ 7 \ 8; \ 0 \ 1 \ 9; \ 4 \ 3 \ 6];$$

A является массивом 3x3, то есть его размерности строк и столбцов равны трем. Для добавления третьей размерности к **A** запишем

$$A(:, :, 2) = [1 \ 0 \ 4; \ 3 \ 5 \ 6; \ 9 \ 8 \ 7].$$

MATLAB выдаст

$$A(:, :, 1) =$$

5	7	8
0	1	9
4	3	6

$$A(:, :, 2) =$$

1	0	4
3	5	6
9	8	7

Вы можете продолжить добавление строк, столбцов или страниц аналогичным образом.

Расширение Многомерных Массивов. Для расширения любой размерности массива **A** нужно:

- Увеличить или добавить соответствующий индекс и задать требуемые значения.
- Добавить такое же количество элементов к соответствующим размерностям массива. Так, для числовых массивов все строки должны иметь одинаковое число элементов, все страницы должны иметь одинаковое число строк и столбцов и т.д.

Вы можете воспользоваться свойством скалярного распространения системы MATLAB, совместно с оператором двоеточия, для заполнения всей размерности единственным числом:

```
A(:, :, 3) = 5;
```

```
A(:, :, 3)
```

```
ans =
```

```
5 5 5
5 5 5
5 5 5
```

Для превращения **A** в четырехмерный массив размерности 3x3x3x2 введите

```
A(:, :, 1, 2) = [1 2 3; 4 5 6; 7 8 9];
```

```
A(:, :, 2, 2) = [9 8 7; 6 5 4; 3 2 1];
```

```
A(:, :, 3, 2) = [1 0 1; 1 1 0; 0 1 1];
```

Отметим, что после первых двух вводов MATLAB добавляет в **A** требуемое количество нулей, чтобы поддержать соответствующие размеры размерностей (речь идет о первом элементе по четвертой размерности, то есть при четвертом индексе равном единице, массив **A** будет содержать три нулевые матрицы размера 3x3).

Создание массивов с применением функций MATLAB-а.

Вы можете использовать для создания многомерных массивов такие функции MATLAB-а как **randn**, **ones**, и **zeros**, совершенно аналогично способу используемому для двумерных матриц. Каждый вводимый аргумент представляет размер соответствующей размерности в результирующем массиве. Например, для создания массива нормально распределенных случайных чисел размера 4x3x2 следует записать:

```
B = randn(4,3,2).
```

Для создания массива, заполненного единственным постоянным значением можно воспользоваться функцией **repmat**. Эта функция копирует массив (в нашем случае массив размера 1x1) вдоль вектора размерностей массива.

```
B = repmat(5,[3 4 2])
```

```
B(:, :, 1) =
```

```
5 5 5 5
5 5 5 5
5 5 5 5
```

```
B(:, :, 2) =
```

```
5 5 5 5
5 5 5 5
5 5 5 5
```

Внимание! Любая размерность массива может иметь размер 0, что просто дает пустой массив (empty array) . Так, размер 10x0x20 является допустимым размером многомерного массива.

Создание многомерного массива при помощи функции **cat**.

Функция **cat** дает простой путь построения многомерных массивов; она объединяет набор массивов вдоль заданной размерности.

$$\mathbf{B} = \text{cat}(\text{dim}, \mathbf{A1}, \mathbf{A2} \dots)$$

где **A1**, **A2** и т.д. являются объединяемыми массивами. а **dim** есть размерность, вдоль которой они объединяются. Например, для создания нового массива из двух двумерных матриц при помощи функции **cat** запишем

$$\mathbf{B} = \text{cat}(3, [2 \ 8; 0 \ 5], [1 \ 3; 7 \ 9])$$

что дает трехмерный массив с двумя страницами

$$\mathbf{B}(:, :, 1) = \begin{bmatrix} 2 & 8 \\ 0 & 5 \end{bmatrix}$$

$$\mathbf{B}(:, :, 2) = \begin{bmatrix} 1 & 3 \\ 7 & 9 \end{bmatrix}$$

Функция **cat** принимает любые комбинации существующих и новых данных. Более того, вы можете осуществлять вложение данных функций. Приведенные ниже строки, к примеру, формируют четырехмерный массив:

$$\begin{aligned} \mathbf{A} &= \text{cat}(3, [9 \ 2; 6 \ 5], [7 \ 1; 8 \ 4]) \\ \mathbf{B} &= \text{cat}(3, [3 \ 5; 0 \ 1], [5 \ 6; 2 \ 1]) \\ \mathbf{D} &= \text{cat}(4, \mathbf{A}, \mathbf{B}, \text{cat}(3, [1 \ 2; 3 \ 4], [4 \ 3; 2 \ 1])) \end{aligned}$$

Функция **cat** автоматически добавляет, при необходимости, единичные индексы между размерностями. Например, для создания массива размера 2x2x1x2 можно ввести

$$\mathbf{C} = \text{cat}(4, [1 \ 2; 4 \ 5], [7 \ 8; 3 \ 2])$$

В данном случае функция **cat** вводит нужное число единичных размерностей для создания четырехмерного массива, чья последняя размерность не является единичной. Если бы аргумент **dim** был бы равен 5, последняя запись привела бы к массиву размера 2x2x1x1x2. Это добавляет еще одну единицу в индексации массива. Для обращения к значению 8 в четырехмерном случае нужно применить следующую индексацию

$$\mathbf{C}(1, 2, 1, 2)$$



Индекс единичной размерности

Определение характеристик многомерных массивов.

Для получения информации об имеющихся многомерных массивах можно воспользоваться стандартными командами **size** (дает размер массива), **ndims** (дает количество размерностей) и **whos** (последняя команда дает подробную информацию о всех переменных рабочего пространства системы MATLAB). Для вышеприведенного примера мы получим

$$\text{size}(\mathbf{C})$$

```
ans =
     2     2     1     2

ndims(C)
ans =
     4
```

Индексация

Многие концепции, используемые в двумерном случае, распространяются также на многомерные массивы. Для выделения (обращения) к какому-либо одному элементу многомерного массива следует воспользоваться целочисленной индексацией. Каждый индекс указывает на соответствующую размерность: первый индекс на размерность строк, второй индекс на размерность столбцов, третий на первую размерность страниц и так далее. Рассмотрим массив случайных целых чисел **nddata** размера 10x5x3:

```
nddata = fix (8*randn (10, 5, 3));
```

Для обращения к элементу (3,2) на странице 2 массива **nddata** нужно записать **nddata(3,2,2)**. Вы можете также использовать векторы как массив индексов. В этом случае каждый элемент вектора должен быть допустимым индексом, то есть должен быть в пределах границ, определенных для размерностей массива. Так, для обращения к элементам (2,1), (2,3), и (2,4) на странице 3 массива **nddata**, можно записать

```
nddata (2, [1 3 4], 3).
```

Оператор двоеточия и индексирование многомерных массивов.

Стандартная индексация MATLAB-а при помощи оператора двоеточия (colon) применима и в случае многомерных массивов. Например, для выбора всего третьего столбца страницы 2 массива **nddata** используется запись **nddata(:, 3, 2)**. Оператор двоеточия также полезен и для выделения определенных подмножеств данных. Так, ввод **nddata(2:3,2:3,1)** дает массив (матрицу) размера 2x2, который является подмножеством данных на странице 1 массива **nddata**. Эта матрица состоит из данных второй и третьей строки и второго и третьего столбца первой страницы многомерного массива. Оператор двоеточия может использоваться для индексации с обеих сторон записи. Например, для создания массива нулей размера 4x4 записываем:

```
C = zeros (4,4)
```

Теперь, чтобы присвоить значения подмножества 2x2 массива **nddata** четырём элементам в центре массива **C** запишем

```
C(2:3,2:3) = nddata (2:3,1:2,2)
```

Устранение неопределенностей в многомерной индексации

Некоторые выражения, такие как

```
A(:, :, 2) = 1:10
```

Являются неоднозначными, поскольку они не обеспечивают достаточного объема информации относительно структуры размерности, в которую вводятся данные. В представленном выше случае, делается попытка задать одномерный вектор в двумерном объекте. В таких ситуациях MATLAB выдает сообщение об ошибке. Для устранения неопределенности, нужно

убедиться, что обеспечена достаточная информация о месте записи данных, и что как данные так и место назначения имеют одинаковую форму. Например,

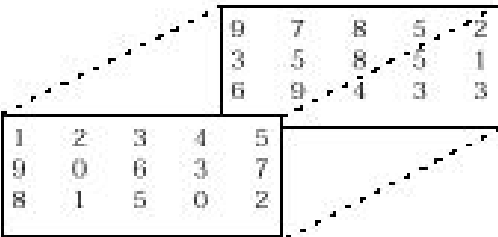
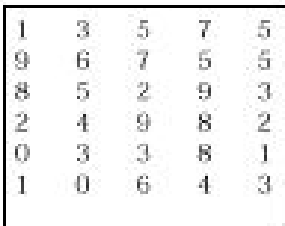
$$A(1,:2) = 1:10.$$

Изменение формы (Reshaping)

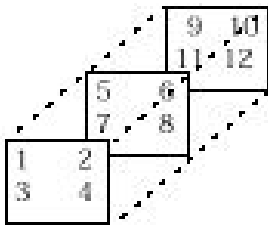
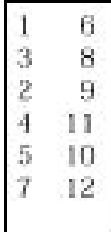
Если вы не меняете форму или размер, массивы в системе MATLAB сохраняют размерности, заданные при их создании. Вы можете изменить размер массива путем добавления или удаления элементов. Вы можете также изменить форму массива изменяя размерности строк, столбцов и страниц, при условии сохранения тех же элементов. Функция **reshape** выполняет указанную операцию. Для многомерных массивов эта функция имеет вид

$$B = \text{reshape}(A, [s1 \ s2 \ s3 \ \dots])$$

где **s1**, **s2**, и так далее характеризуют желаемый размер для каждой размерности преобразованной матрицы. Отметим, что преобразованный массив должен иметь то же число элементов, что и исходный массив (иными словами, произведение размеров массивов должно быть неизменным).

M	reshape(M, [6 5])
	

Функция **reshape** «действует» вдоль столбцов. Она создает преобразованную матрицу путем взятия последовательных элементов вдоль каждого столбца исходной матрицы.

C	reshape(C, [6 2])
	

Ниже в качестве примеров приведены несколько примеров массивов, которые могут быть получены из массива **nddata** (обратите внимание на размерности).

$$B = \text{reshape}(\text{nddata}, [6 \ 25])$$

```
C = reshape(nddata,[5 3 10])
```

```
D = reshape(nddata,[5 3 2 5])
```

Удаление единичных размерностей.

Система MATLAB создает единичные размерности, когда вы задаете их при создании или преобразовании массива, или же в результате вычислений приводящих к появлению указанных размерностей.

```
B = repmat (5, [2 3 1 4] ) ;
```

```
size(B)
ans =
    2    3    1    4
```

Функция **squeeze** удаляет единичные размерности из массива.

```
C = squeeze(B);
```

```
size(C)
ans =
    2    3    4
```

Функция **squeeze** не оказывает воздействия на двумерные массивы – векторы-строки остаются строками.

Вычисления с многомерными массивами

Многие вычислительные и математические функции MATLAB-а принимают в качестве аргументов многомерные массивы. Эти функции действуют на определенные размерности многомерных массивов, в частности, на отдельные элементы, векторы или матрицы.

Действия над векторами

Функции которые действуют над векторами, такие как **sum**, **mean**, и т.д., по умолчанию обычно действуют вдоль первой неединичной размерности многомерного массива. Многие из этих функций дают возможность задать размерность вдоль которой они действуют. Однако, есть и исключения. Например, функция **cross**, которая определяет векторное произведение двух векторов, действует вдоль первой неединичной размерности, имеющей размер 3.

Внимание! Во многих случаях эти функции имеют другие ограничения на входные аргументы – например, некоторые функции, допускающие многомерные входные массивы, требуют чтобы массивы имели одинаковый размер.

Поэлементное воздействие

Те функции MATLAB-а, которые действуют поэлементно на двумерные массивы, такие как тригонометрические и экспоненциальные функции, работают совершенно аналогично и в многомерном случае. Например, функция **sin** возвращает массив того же размера, что и входной массив. Каждый элемент выходного массива является синусом соответствующего элемента входного массива. Аналогично, все арифметические, логические операторы и операторы отношения действуют с соответствующими элементами многомерных массивов (которые должны иметь одинаковые размеры каждой размерности). Если один из операндов является скаляром, а второй – скаляром, то операторы применяют скаляр ко всем элементам массива.

Действия над плоскостями и матрицами

Функции, действующие над плоскостями или матрицами, такие как функции линейной алгебры или матричные функции в директории **matfun**, не принимают в качестве аргументов многомерные массивы. Иными словами, вы не можете использовать функции в директории **matfun**, или операторы *****, **^**, ****, или **/**, с многомерными массивами. Попытка использования многомерных массивов или операндов в таких случаях приводит к сообщению об ошибке. Вы можете, тем не менее, применить матричные функции или операторы к матрицам внутри многомерных массивов. Например, сообразим трехмерный массив **A**

```
A = cat (3 , [1 2 3; 9 8 7; 4 6 5], [0 3 2; 8 8 4; 5 3 5], [6 4 7; 6 8 5; 5 4 3]);
```

Применение функции **eig** ко всему многомерному массиву дает сообщение об ошибке:

```
eig(A)
??? Error using → eig
Input arguments must be 2-D.
```

Вы можете, однако, применить функцию **eig** к отдельным плоскостям в пределах массива. Например, воспользуемся оператором двоеточия для выделения одной страницы (допустим, второй):

```
eig(A(:, :, 2))
ans =
    -2.6260
    12.9129
     2.7131
```

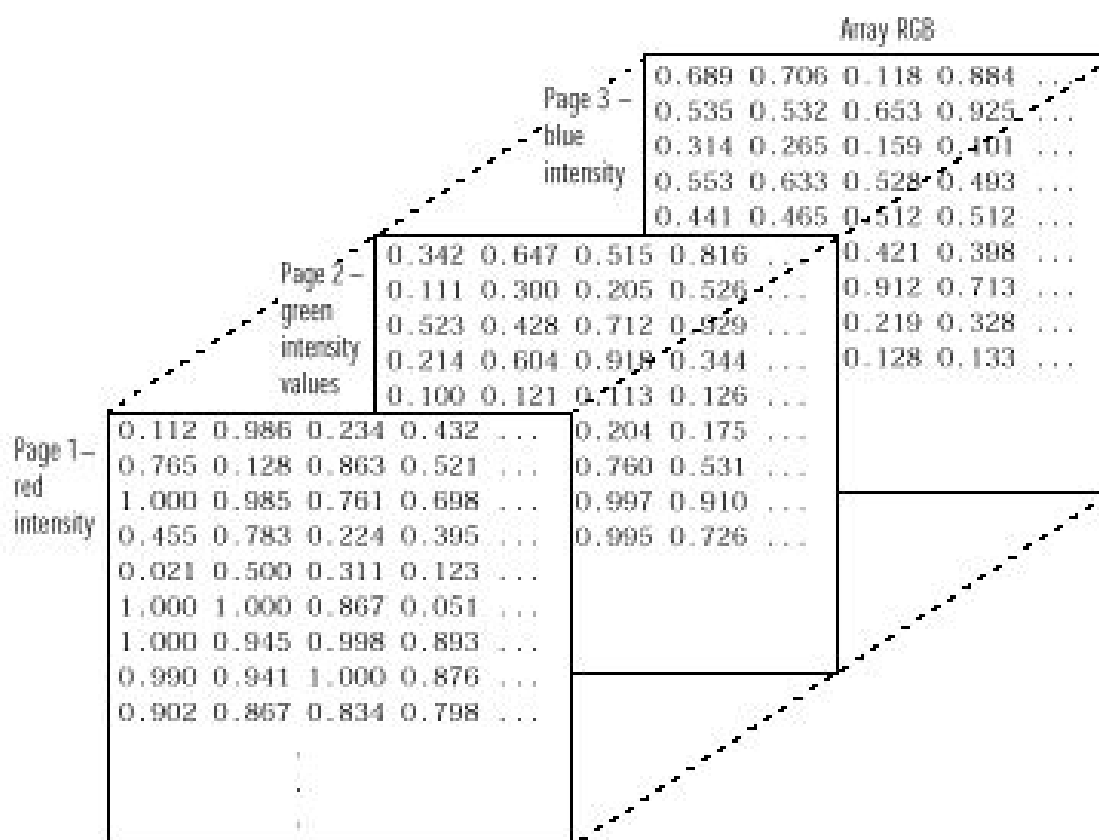
Внимание! В первом случае, где не используется оператор двоеточия, для избежания ошибки нужно использовать функцию **squeeze**. Например, ввод **eig (A(2,:,:))** приводит к ошибке так как размер входа есть [1 3 3]. Выражение **eig(squeeze(A(2, :, :)))**, однако, передает функции **eig** допустимую двумерную матрицу.

Организация данных в многомерных массивах

Вы можете использовать два возможных варианта представления данных при помощи многомерных массивов:

- Как плоскости (или страницы) двумерных данных. В дальнейшем вы можете обращаться с этими страницами как с матрицами.
- Как многомерные данные. Например, вы можете иметь четырехмерный массив, где каждый элемент соответствует температуре или давлению воздуха, измеренным на равномерно распределенной трехмерной (пространственной) сетке в комнате.

В качестве конкретного примера рассмотрим представление какого-либо изображения в формате RGB. Напомним, что в формате RGB изображение хранится в виде трех двумерных матриц одинакового размера, каждая из которых характеризует интенсивность одного цвета – красного (Red), зеленого (Green) и синего (Blue) - в соответствующей точке. Общая картина при этом получается в результате наложения трех указанных матриц. Для отдельного изображения, использование многомерных массивов является, вероятно, наиболее легким путем для запоминания данных и доступа к ним.



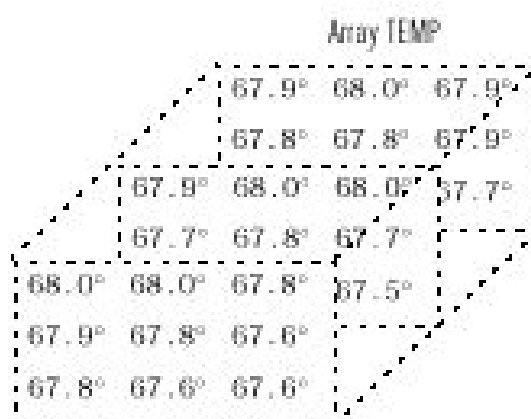
Пусть все изображение хранится в файле **RGB**. Для доступа к полной плоскости изображения в одном цвете, допустим – красном, следует записать

red_plane = RGB (:,:,1);

Для доступа к части всего изображения можно использовать запись

subimage = RGB (20:40, 50:85, :)

Изображение в формате RGB является хорошим примером данных, для которых может потребоваться доступ к отдельным плоскостям, для операций типа фильтрации или просто демонстрации. В других задачах, однако, сами данные могут быть многомерными. Рассмотрим, например, набор температур, измеренных на равномерной пространственной сетке какого-либо помещения.



В данном случае пространственное положение каждого значения температуры является составной частью набора данных, то есть физическое расположение в трехмерном пространстве является частью информации. Такие данные также весьма приспособлены для представления при помощи многомерных массивов (см. рисунок выше).

Здесь, чтобы найти среднее значение всех измерений, то есть среднюю температуру воздуха в комнате, можно записать

mean (mean (mean (TEMP)))

где через **TEMP** обозначен массив четырехмерных данных.

Для получения вектора «серединных» температур (элемента (2,2)) комнаты на каждой странице, то есть в каждом сечении, запишем

B = TEMP (2, 2, :).

ОРГАНИЗАЦИЯ И ХРАНЕНИЕ ДАННЫХ

Для хранения различных типов данных в системе MATLAB используются так называемые *структуры* (structure) и *ячейки* (cell). Структуры (иногда их называют массивами структур) служат для хранения массивов различных типов данных, организованных по принципу именованных полей. Ячейки (или массивы ячеек) являются специальным классом массивов системы MATLAB, чьи элементы состоят из ячеек, в которых могут храниться любые другие массивы данных, применяемые в MATLAB-е. Как структуры, так и ячейки обеспечивают иерархический механизм для хранения самых различных типов данных. Они отличаются друг от друга прежде всего способом организации базы данных. При использовании структур доступ к данным осуществляется при помощи наименований полей, тогда как в массивах ячеек доступ осуществляется при помощи матричной индексации.

В приведенных ниже таблицах дается краткое описание функций MATLAB-а, предназначенных для работы с массивами структур и ячеек

Структуры

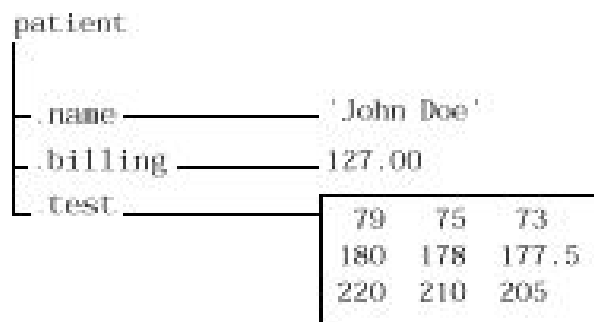
Функция	Описание
fieldnames	Получить имена полей
getfield	Получить содержание поля
isfield	Истинно, если поле есть в структуре
isstruct	Истинно, если структура
rmfield	Удалить поле
setfield	Установить содержимое поля
struct	Создать массив структур
struct2cell	Преобразовать структуру в массив ячеек

Ячейки

Функция	Описание
cell	Создать массив ячеек
cell2struct	Преобразовать массив ячеек в структуру
celldisp	Показать содержимое массива ячеек
cellfun	Применить функцию к массиву ячеек
cellplot	Показать графическую структуру массива ячеек
deal	Обмен данными между любыми классами массивов
iscell	Истинно для массивов ячеек
num2cell	Преобразовать числовой массив в массив ячеек

МАССИВЫ СТРУКТУР

Структуры это массивы данных с поименованными «хранилищами» данных, называемыми *полями*. Поля структуры могут содержать данные любого типа. Например, одно поле может содержать текстовую строку, представляющую имя (name), второе поле может содержать скалярную переменную, являющуюся счетом за лечение (billing), третье может содержать матрицу результатов медицинских анализов (test) и так далее.



Как и обычным массивам данных, структурам присущи основные свойства массивов. Одна структура является структурой размера 1x1, точно так же как число 5 является числовым массивом размера 1x1. Вы можете строить структуры с любой допустимой размерностью или формы, включая многомерные массивы структур.

Создание массивов структур

Имеется два следующих способа создания структур:

- Путем использования операторов присваивания.
- С использованием функции **struct**.

Создание массивов структур с применением операторов присваивания.

Вы можете построить простую структуру размера 1x1 путем прямого присваивания значений индивидуальным полям. MATLAB при этом автоматически конструирует соответствующую

структуру. Например, создадим 1x1 структуру данных пациента лечебницы, показанную в начале данного раздела. Для этого следует ввести следующие записи:

```
patient.name = 'John Doe';  
patient.billing = 127.00;  
patient.test = [79 75 73; 180 178 177.5; 220 210 205];
```

Если ввести теперь в командной строке запись

```
patient
```

то MATLAB ответит

```
name: 'John Doe'  
billing: 127  
test: [3x3 double]
```

patient является массивом, представляющим собой структуру с тремя полями. Для расширения данного массива нужно просто добавить соответствующие индексы после имени структуры:

```
patient(2).name = 'Ann Lane';  
patient(2).billing = 28.50;  
patient(2).test = [68 70 68; 118 118 119; 172 170 169];
```

Структура **patient** имеет теперь размер [1 2]. Отметим, что если массив структур содержит более одного элемента, то MATLAB уже не выводит на экран содержание отдельных полей при вводе имени структуры. Взамен, на дисплей выдаются общая информация о содержимом структуры, то есть имена полей:

```
Patient
```

```
patient =
```

```
1x2 struct array with fields:  
    name  
    billing  
    test
```

Для получения данной информации вы можете также использовать функцию **fieldnames**. Данная функция выдает массив ячеек содержащих названия полей в форме строки. Если вы расширяете структуру, MATLAB заполняет те поля, в которые вы не ввели данные, пустыми матрицами так, что:

- Все структуры в массиве имеют одинаковое число полей.
- Все соответствующие поля имеют одинаковые имена.

Например, при вводе

```
patient(3).name = 'Alan Johnson'
```

структура **patient** принимает размер 1x3. При это оба поля **patient(3).billing** и **patient(3).test** содержат пустые матрицы.

Внимание! Размеры данных в одноименных полях могут быть различными. В нашем примере со структурой **patient** поля **name** могут иметь различную длину, поля **test** могут содержать массивы числовых данных различных размеров и так далее.

Создание массива структур с использованием функции **struct**.

Вы можете заранее создать массив структур применив функцию **struct**. Ее основная форма имеет вид

str_array = struct ('поле1',знач1,'поле2',знач2, ...)

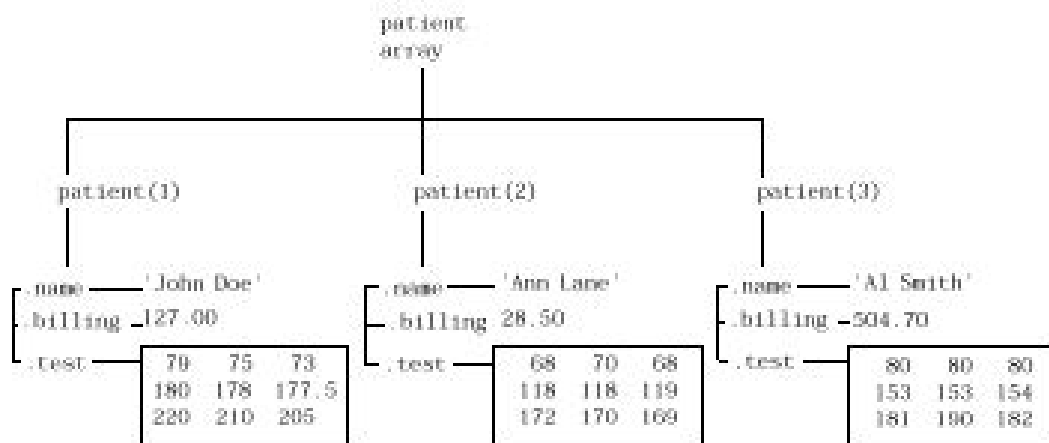
где аргументами являются имена полей и их соответствующие значения. Значением поля может быть или одно значение, представленное любой допустимой конструкцией в MATLAB-е, или массив ячеек данных (массивы ячеек рассмотрены в следующем разделе). Все значения полей в списке аргументов должны иметь одинаковый вид (единственное значение или массив ячеек).

Вы можете использовать различные методы для задания массива структур. Эти методы отличаются способом инициализации полей структуры. В качестве примера рассмотрим задание структуры размера 1x3 с именем **weather** (погода), имеющую поля **temp** (температура) и **rainfall** (дождевые осадки). Три различные способа задания такой структуры даны в приведенной ниже таблице.

Метод	Синтаксис	Задание
Функция struct	weather(3) = struct('temp',72,'rainfall',0.0);	Структура weather(3) инициализируется с указанными значениями полей. Поля остальных двух структур в массиве, weather(1) и weather(2) , содержат в качестве данных пустые матрицы.
Сочетание функций struct и repmat	weather = repmat (struct ('temp', 72, 'rainfall', 0.0), 1, 3);	Все структуры в массиве weather инициализируются с использованием одинаковых значений одноименных полей.
Функция struct с использованием синтаксиса ячеек	weather = struct ('temp',{68, 80, 72}, 'rainfall', {0.2,0.4,0.0});	Структуры в массиве weather инициализируются с разными значениями полей, заданных массивом ячеек.

Обращение к данным в массивах структур.

Используя индексацию массива структур, можно осуществить обращение к данным любого поля или любого элемента поля в массиве структуры. Аналогичным образом, вы можете задать значение любого поля или элемента поля структуры. В качестве примера, используемого в данном разделе, рассмотрим структуру, представленную на приведенном ниже рисунке.



Вы можете обратиться к подмассивам путем добавления стандартной индексации к имени массива структур. Например, следующая запись приводит к структуре размера 1x2

```
mypatients = patient(1:2)
```

1x2 struct array with fields:

```
name  
billing  
test
```

Первая структура в массиве **mypatients** совпадает с первой структурой в массиве **patient**:

```
mypatients(1)
```

```
ans =  
name: 'John Doe'  
billing: 127  
test: [3x3 double].
```

Для обращения к полю определенной структуры, нужно добавить точку (.) после имени структуры, с указанием далее имени поля:

```
str = patient(2) . name
```

```
str =  
Ann Lane
```

Для обращения к элементам внутри полей, следует добавить требуемые индексы к имени поля. Если поле содержит числовой массив, нужно использовать индексацию цифровых массивов. Если поле содержит массив ячеек, используйте соответствующую индексацию ячеек и так далее. Например,

```
test2b = patient(3).test(2,2)
```

```
test2b =  
153
```

Аналогичную форму записи следует использовать и для задания значений переменных внутри поля, например,

```
patient(3).test(2,2) = 7
```

Вы можете также одновременно извлечь данные одноименных полей многомерной структуры. Например, запись ниже создает вектор 1x3, содержащий все значения счетов полей **billing** fields.

```
bills = [patient.billing]
```

```
bills =  
127.0000  28.5000  504.7000
```

Аналогично, вы можете создать массив ячеек, содержащий данные температур **test** для первых двух структур.

```
tests = {patient(1:2).test}
```

```
tests =  
[3x3 double] [3x3 double]
```

Обращение к полям структуры с применением функций **setfield** и **getfield**

Прямая индексация обычно является наиболее эффективным способом задания или получения значений полей структуры. Если, однако, вы знаете только название поля в виде строки, например, если вы использовали функцию **fieldnames** для получения имени поля в пределах М-файла – то для указанных операций с данными этих полей можно также применить функции **setfield** и **getfield**.

Функция **getfield** позволяет получить значение или значения поля или элемента поля и имеет следующий синтаксис

```
f = getfield(array,{array_index},'field',{field_index})
```

где индекс **field_index** является необязательным, а **array_index** является необязательным для массива структур размера 1x1. Данный синтаксис соответствует записи

```
f = array(array_index).field(field_index)
```

Например, для обращения к полю **name** во второй структуре массива **patient** запишем

```
str = getfield(patient,{2},'name')
```

Аналогично, функция **setfield** дает возможность задать значения полей используя синтаксис

```
f = setfield (array,{array_index},'field',{field_index},value)
```

Определение размера массива структур

Для получения размера массива структур или размера любого поля структуры. можно воспользоваться функцией **size**. При вводе в качестве аргумента функции **size** имени структуры, данная функция возвращает вектор размерностей массива. Если задать аргумент в форме

массив(n).поле, функция **size** возвращает размер содержимого поля. Например, для нашей структуры **patient** размера 1x3, запись **size(patient)** возвращает вектор [1 3]. Выражение **size(patient(1,2).name)** возвращает длину строки имени элемента (1,2) структуры **patient**.

Добавление полей к структуре

Вы можете добавить поле ко всем структурам в массиве добавлением поле к любой одной структуре. Например, для добавления поля номера социальной страховки к массиву **patient** можно воспользоваться записью вида

```
patient(2).ssn = '000-00-0000'
```

При этом поле **patient(2).ssn** второго пациента имеет заданное значение. Все другие структуры в массиве структур также имеют поле **ssn**, но эти поля содержат пустые матрицы до тех пор, пока вы не зададите в явном виде соответствующие значения.

Удаление поля из структуры

Вы можете удалить любое поле заданной структуры при помощи функции **rmfield**. Ее наиболее общая форма имеет вид

```
struc2 = rmfield(array,'field')
```

где **array** это массив структур, а **'field'** является именем поля, которое вы хотите удалить. Например, чтобы удалить поле **name** из массива **patient**, нужно ввести:

```
patient = rmfield(patient,'name')
```

Применение функций и операторов

Вы можете осуществлять операции над полями и над элементами полей точно так же, как над любыми другими массивами системы MATLAB. Для выбора данных, над которыми нужно произвести действия нужно использовать индексацию. Например, следующее выражение вычисляет среднее значение вдоль строк массива **test** в **patient(2)**:

```
mean((patient(2).test)')
```

Зачастую бывают различные возможности для применения функций или операторов к полям массива структур. Один из путей суммирования всех полей **billing** в структуре **patient** выглядит следующим образом:

```
total = 0;  
for j = 1:length(patient)  
    total = total + patient(j).billing;  
end
```

Для упрощения подобных операций, MATLAB предоставляет возможность производить действия одновременно со всеми одноименными полями массива структур. Для этого нужно просто заключить выражение (допустим, **array.field**) в квадратные скобки внутри применяемой функции. Например, вы можете решить приведенную выше задачу, записав

```
total = sum ([patient.billing])
```


Подобная запись эквивалентна использованию так называемого списка, разделенного запятой (comma-separated list)

```
total = sum ([patient(1).billing , patient(2).billing ,...])
```

Такой синтаксис наиболее полезен в случаях, когда поле является скалярным операндом.

Создание функций для операций над массивами структур

Вы можете записать свои функции в виде М-файлов для работы со структурами любой нестандартной формы. При этом вам придется осуществить собственный контроль ошибок. Иными словами, вам следует убедиться, что осуществляется проверка действий над выбранными полями.

В качестве примера, рассмотрим набор данных, который описывает измерения в различных моментах времени различных токсинов в источнике питьевой воды. Данные состоят из 15 различных наблюдений, где каждое наблюдение содержит три независимых замера. Вы можете организовать эти данные в виде набора 15 структур, где каждая структура имеет три поля, по одному для каждого проведенного измерения.

Приведенная ниже функция **concen**, действует над массивом структур со специфическими характеристиками. Их характеристики должны содержать поля **lead** (свинец), **mercury** (ртуть), и **chromium** (хром).

```
function [r1, r2] = concen(toxtest);  
% Create two vectors. r1 contains the ratio of mercury to lead  
% at each observation. r2 contains the ratio of lead to chromium.  
r1 = [toxtest.mercury]./[toxtest.lead];  
r2 = [toxtest.lead]./[toxtest.chromium];  
% Plot the concentrations of lead, mercury, and chromium  
% on the same plot, using different colors for each.  
lead = [toxtest.lead];  
mercury = [toxtest.mercury];  
chromium = [toxtest.chromium];  
plot(lead,'r'); hold on  
plot(mercury,'b')  
plot(chromium,'y'); hold off
```

Данная функция создает два вектора. **r1** содержит отношение ртути к свинцу в каждом наблюдении, а **r2** содержит отношение свинца к хрому. Далее эта функция строит кривые концентрации свинца, ртути и хрома на одном графике, используя разные цвета (красный – свинец, синий – ртуть, желтый – хром).

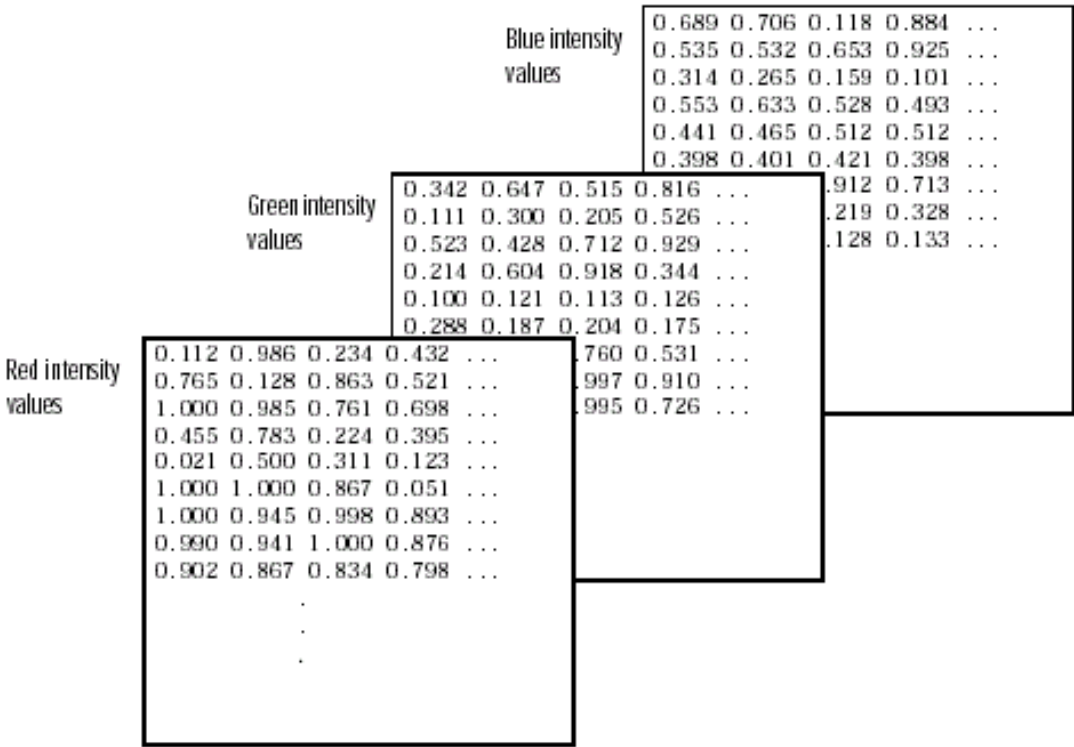
Попробуйте применить данную функцию на примеры структуры **test** со следующими данными

```
test(1).lead = .007; test(2).lead = .031; test(3).lead = .019;  
test(1).mercury = .0021; test(2).mercury = .0009;  
test(3).mercury = .0013;  
test(1).chromium = .025; test(2).chromium = .017;  
test(3).chromium = .10;
```

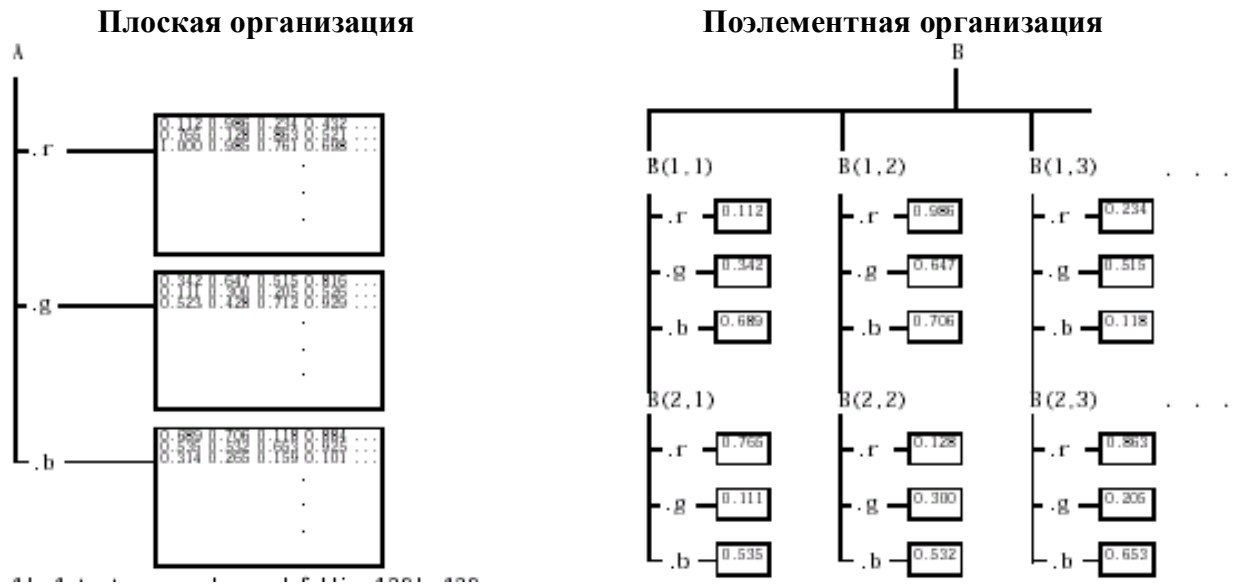
Организация данных в массиве структур

Ключ к организации массива структур состоит в выборе способа, которым вы хотите обращаться к подмассивам данных или отдельным данным структуры. Это, в свою очередь, определяет как вы должны построить массив, содержащий структуры и как выбирать поля

структуры. Например, рассмотрим RGB изображение размера 128x128, запомненное в трех различных массивах : **RED**, **GREEN** и **BLUE**.



Имеются по меньшей мере две возможности для организации таких данных в массив структур.



Плоская организация

В этом варианте, каждое поле структуры представляет полную плоскость изображения в красном, зеленом или синем цветах. Вы можете создать такую структуру используя запись

```

A.r = RED;
A.g = GREEN;
A.b = BLUE;

```

Подобный подход позволяет вам легко извлекать полное изображение в отдельных составляющих цветов, для решения таких задач как фильтрация. Например, для обращения ко всей красной плоскости нужно просто записать

```

red_plane = A.r;

```

Плоская организация имеет то дополнительное преимущество, что массив структур можно без труда дополнить другими изображениями. Если у вас есть набор изображений, вы можете запомнить их как **A(2)**, **A(3)**, и так далее, где каждая структура содержит полное изображение.

Недостаток плоской организации становится очевидным, когда вам нужно обратиться к отдельным частям изображения. В этом случае вы должны оперировать с каждым полем в отдельности:

```

red_sub = A.r (2:12, 13:30);
grn_sub = A.g (2:12, 13:30);
blue_sub = A.b (2:12, 13:30);

```

Поэлементная организация

Данный вариант имеет то преимущество, что обеспечивает простой доступ к подмножествам данных. Для организации данных в данной форме нужно использовать команды

```

for i = 1:size(RED,1)
  for j = 1:size(RED,2)
    B(i,j) .r = RED(i,j);
    B(i,j) .g = GREEN(i,j);
    B(i,j) .b = BLUE(i,j);
  end
end

```

При поэлементной организации, вы можете осуществить обращение к подмножествам данных при помощи единственного выражения:

```

Bsub = B(1:10, 1:10);

```

Однако, обращение к полной плоскости изображения при поэлементном методе требуется цикл :

```

red_plane = zeros(128,128);
for i = 1 : (128*128)
  red_plane(i) = B(i).r;
end

```

Поэлементная организация не является лучшим выбором для большинства приложений, связанных с обработкой изображений. Однако, она может быть лучшей для других приложений, когда вам требуется часто обращаться к отдельным подмножествам полей структуры. Пример в следующем разделе демонстрирует данный тип приложений

Пример - Простая база данных

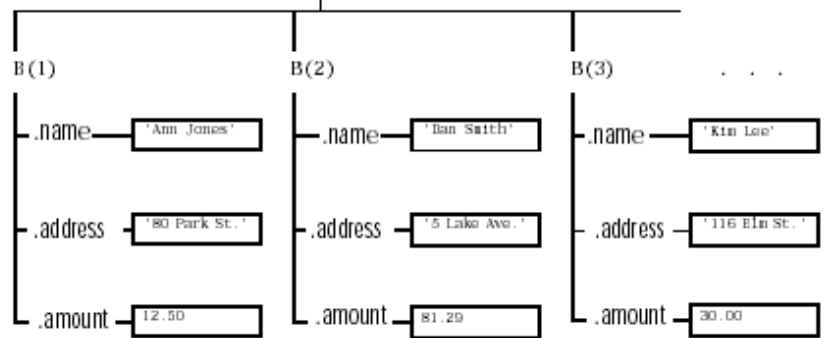
Рассмотрим организацию простой базы данных.

А Плоская организация



```
A.name = strvcats('Ann Jones','Dan Smith',...);
A.address = strvcats('80 Park St.','5 Lake Ave.',...);
A.amount = [12.5;81.29;30; ...];
```

В Поэлементная организация



```
B(1).name = 'Ann Jones';
B(1).address = '80 Park St.';
B(1).amount = 12.5;
```

```
B(2).name = 'Dan Smith';
B(2).address = '5 Lake Ave.';
B(2).amount = 81.29;
```

Оба возможных способов организации базы данных имеет определенные преимущества, зависящие от того как вы хотите осуществить доступ к данным:

- Плоская организация обеспечивает более легкую возможность вычислений одновременно над всеми полями. Например, чтобы найти среднее значение всех данных в поле **amount** следует записать:

а) При плоской организации

```
avg = mean(A.amount);
```

б) При поэлементной организации

```
avg = mean([B.amount]);
```

Поэлементная организация дает более легкий доступ ко всей информации, связанной с одним клиентом. Рассмотрим М-файл, названный **client.m**, который осуществляет вывод на экран имени и адреса любого клиента. При использовании плоской организации, следует вводить в качестве аргументов индивидуальные поля:

```
function client(name,address, amount)
```

```
disp(name)
```

```
disp(address)
```

```
disp(amount)
```

Для вызова функции **client** для второго клиента записываем,

```
client(A.name(2,:),A.address(2,:), A. amount (2,:))
```

При использовании поэлементной организации вводится вся структура

function client(B)
disp(B)

Для вызова функции **client** для второго клиента при этом просто записываем,

client(B(2))

- Поэлементная организация позволяет более просто расширять поля массивов строк. Если вы заранее не знаете максимальную длину строки при плоской организации, вам может потребоваться часто корректировать поля **name** или **address**, чтобы ввести более длинные строки.

Обычно данные не диктуют выбора организации базы данных. Скорее, вы сами должны решить, как вы хотите осуществлять доступ и операции над данными.

Вложенные структуры

Поле структуры может содержать другую структуру, и даже массив структур. Если вы уже имеете некоторую структуру, то для вложения новых структур в любое поле данной структуры вы можете воспользоваться как функцией **struct**, так и применить непосредственно оператор присваивания

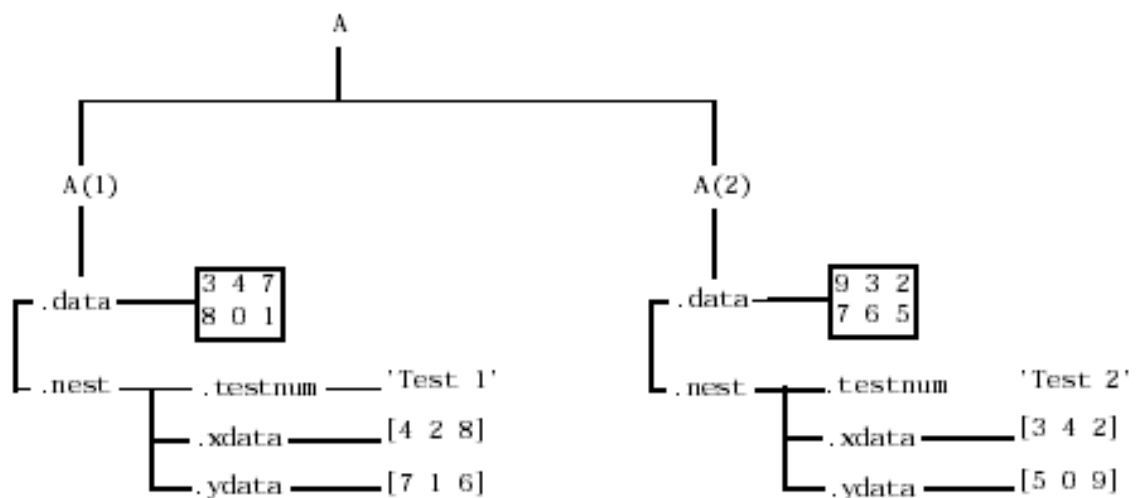
Создание вложенных структур при помощи функции **struct**

Для создания вложенных структур вы можете применить функцию **struct**. Например, создадим массив структур размера 1x1 со вложенной в поле **nest** структурой:

```
A = struct('data',[3 4 7; 8 0 1],'nest',struct('testnum','Test 1','xdata',[4 2 8],'ydata',[7 1 6]));
```

Применим теперь операторы присваивания для добавления второго элемента к массиву **A**:

```
A(2).data = [9 3 2; 7 6 5];  
A(2).nest.testnum = 'Test 2';  
A(2).nest.xdata = [3 4 2];  
A(2).nest.ydata = [5 0 9];
```



Индексация вложенных структур

Для обращения к вложенным структурам, нужно просто добавить имена вложенных полей с использованием точечных разделителей. Первая текстовая строка в индексированном выражении определяет массив структур, а последующие выражения дают доступ к полям, содержащим другие структуры. Например, массив **A**, созданный ранее, имеет три уровня вложения:

- Для обращения к вложенной структуре внутри **A(1)** запишем **A(1).nest**.
- Для обращения к полю **xdata** во вложенной структуре в **A(2)** запишем **A(2).nest.xdata**.
- Для обращения к элементу 2 поля **ydata** в **A(1)**, запишем **A(1).nest.ydata(2)**.

МАССИВЫ ЯЧЕЕК

Массивы ячеек это массивы данных системы MATLAB элементы которых являются *ячейками* и могут служить «хранилищами» для других массивов данных. Например, одна ячейка массива ячеек может содержать матрицу действительных чисел, другая ячейка – массив текстовых строк, а третья – вектор комплексных значений.

cell 1,1	cell 1,2	cell 1,3																			
<table><tr><td>3</td><td>4</td><td>2</td></tr><tr><td>9</td><td>7</td><td>6</td></tr><tr><td>8</td><td>5</td><td>1</td></tr></table>	3	4	2	9	7	6	8	5	1	<table><tr><td>'Anne Smith'</td></tr><tr><td>'9/12/94'</td></tr><tr><td>'Class II'</td></tr><tr><td>'Obs. 1'</td></tr><tr><td>'Obs. 2'</td></tr></table>	'Anne Smith'	'9/12/94'	'Class II'	'Obs. 1'	'Obs. 2'	<table><tr><td>.25+3i</td><td>8-16i</td></tr><tr><td>34+5i</td><td>7+.92i</td></tr></table>	.25+3i	8-16i	34+5i	7+.92i	
3	4	2																			
9	7	6																			
8	5	1																			
'Anne Smith'																					
'9/12/94'																					
'Class II'																					
'Obs. 1'																					
'Obs. 2'																					
.25+3i	8-16i																				
34+5i	7+.92i																				
cell 2,1	cell 2,2	cell 2,3																			
<table><tr><td>[1.43 2.98</td></tr><tr><td>5.67]</td></tr></table>	[1.43 2.98	5.67]	<table><tr><td>7</td><td>2</td><td>14</td></tr><tr><td>8</td><td>3</td><td>45</td></tr><tr><td>52</td><td>16</td><td>3</td></tr></table>	7	2	14	8	3	45	52	16	3	<table><tr><td>'text'</td><td><table><tr><td>4</td><td>2</td></tr><tr><td>1</td><td>5</td></tr></table></td></tr><tr><td>[4 2 7]</td><td>.02 + 8i</td></tr></table>	'text'	<table><tr><td>4</td><td>2</td></tr><tr><td>1</td><td>5</td></tr></table>	4	2	1	5	[4 2 7]	.02 + 8i
[1.43 2.98																					
5.67]																					
7	2	14																			
8	3	45																			
52	16	3																			
'text'	<table><tr><td>4</td><td>2</td></tr><tr><td>1</td><td>5</td></tr></table>	4	2	1	5																
4	2																				
1	5																				
[4 2 7]	.02 + 8i																				

Вы можете конструировать массивы ячеек любых допустимых размерностей и форм, включая многомерные массивы ячеек.

Создание массивов ячеек

Вы можете создавать массивы ячеек двумя способами:

- Используя операторы присваивания.
- Используя функцию **cell**, а затем назначая данные созданных ячеек.

Применение операторов присваивания

Вы можете создать массив ячеек путем присваивания данных индивидуальным ячейкам, по одной ячейке за один раз. MATLAB при этом автоматически создает требуемый массив ячеек. Существуют два способа индексации данных ячеек:

- **Индексация ячеек**

Заклучите индексы ячейки в обычные скобки с использованием стандартной индексации массивов. Заклучите содержимое ячейки в правой стороне оператора присваивания в фигурные скобки “{ }”. Например, создадим массив ячеек **A** размера 2x2.

```
A(1,1) = {[1 4 3; 0 5 8; 7 2 9]};
A(1,2) = {'Anne Smith'};
A(2,1) = {3+7i};
A(2,2) = {-pi:pi/10:pi};
```

Внимание! Запись “{ }” обозначает пустой массив ячеек, точно так же как “[]” обозначает пустую матрицу для числовых массивов. Вы можете использовать пустой массив ячеек в любых выражениях с массивами ячеек.

- **Индексация содержимого ячеек**

Заклучите индексы ячейки в фигурные скобки, применяя стандартные обозначения массивов. Задайте содержимое ячейки в правой части оператора присваивания в обычном виде.

```
A{1,1} = [1 4 3; 0 5 8; 7 2 9];
A{1,2} = 'Anne Smith';
A{2,1} = 3+7i;
A{2,2} = -pi:pi/10:pi;
```

Различные примеры, приведенные ниже, используют оба приведенных синтаксиса. Обе формы записи являются вполне взаимозаменяемыми.

Внимание! Если вы уже имеете числовой массив с заданным именем, не пытайтесь создать массив ячеек с помощью операторов присваивания, не уничтожив предварительно числовой массив. Если вы не очистите числовой массив, MATLAB примет, что вы пытаетесь «смешать» синтаксисы ячеек и числовых массивов и выдаст сообщение об ошибке.

MATLAB выводит содержимое массива ячеек на дисплей в сжатой форме. Для нашего массива **A** мы получим.

```
A =
    [3x3 double]    'Anne Smith'
    [3.0000+ 7.0000i] [1x21 double]
```

Для вывода полного содержания ячеек, нужно воспользоваться функцией **celldisp**. Для графического вывода на дисплей архитектуры ячейки служит функция **cellplot**. Если вы назначаете данные ячейке, которая находится вне размерности имеющегося массива ячеек, MATLAB автоматически расширяет массив, чтобы включить заданный вами элемент. При этом промежуточные ячейки заполняются пустыми матрицами. Например, приведенный ниже оператор присваивания превращает массив ячеек **A** размера 2x2 в массив размера 3x3.

```
A(3,3) = {5};
```

Все остальные ячейки третьего столбца и третьей строки при этом будут содержать пустые матрицы.

Использование фигурных скобок для построения массивов ячеек

Фигурные скобки, “{}”, являются такими же конструкциями массивов ячеек, как квадратные скобки являются конструкторами числовых массивов. Фигурные скобки используются совершенно аналогично квадратным скобкам, за тем исключением, что их можно использовать для вложения массивов ячеек (см. ниже).

При конструировании массивов с использованием фигурных скобок нужно использовать пробелы или запятые для разделения столбцов, и точки с запятой для разделения строк. Например, ввод

$$C = \{[1 \ 2], [3 \ 4]; [5 \ 6], [7 \ 8]\};$$

приводит к следующему массиву ячеек размера 2x2

cell 1,1 [1 2]	cell 1,2 [3 4]
cell 2,1 [5 6]	cell 2,2 [7 8]

Для объединения отдельных массивов ячеек в новые массивы, вы можете использовать квадратные скобки, как и при объединении числовых массивов.

Задание массивов ячеек при помощи функции **cell**

Функция **cell** позволяет создавать пустые массивы ячеек заданного размера. Например, следующее выражение создает пустой массив ячеек размера 2x2.

$$B = \text{cell}(2, 3);$$

Для заполнения ячеек массива **B** нужно применить операторы присваивания:

$$B(1,3) = \{1:3\};$$

Доступ к данным массивов ячеек

Вы можете извлекать данные из массивов ячеек или же запоминать данные в имеющемся или вновь созданном массиве ячеек двумя способами:

- Использованием индексации содержимого ячеек при помощи обычных индексов.
- Использованием индексов, заключенных в фигурные скобки.

Доступ к данным массивов ячеек с использованием фигурных скобок

Вы можете использовать индексирование содержимого в правой части выражения для обращения ко всем данным в какой-либо отдельной ячейке. Для этого в левой части выражения следует задать переменную для записи содержимого ячейки. Заключите индексы ячеек в фигурные скобки. Это означает, что вы обращаетесь к содержимому ячейки. Рассмотрим следующий массив **N** размера 2x2:

$$\begin{aligned} N\{1,1\} &= [1 \ 2; 4 \ 5]; \\ N\{1,2\} &= 'Name'; \end{aligned}$$

$$\begin{aligned} N\{2,1\} &= 2 - 4i; \\ N\{2,2\} &= 7; \end{aligned}$$

Вы можете получить строку в $N\{1,2\}$ записав

$$c = N\{1,2\}$$

При вводе данной строки MATLAB выдаст

$$c = \text{Name}$$

Внимание! В операторах присваивания вы можете использовать индексацию содержимого только для обращения к одной ячейке, а не к подмножеству ячеек. Например, оба выражения $A\{1, : \} = \text{value}$ и $B = A\{1, : \}$ являются неправильными.

Для обращения к подмножествам содержимого одной ячейки нужно объединить индексирование. Например, чтобы получить элемент $(2,2)$ массива в ячейке $N\{1,1\}$, следует записать:

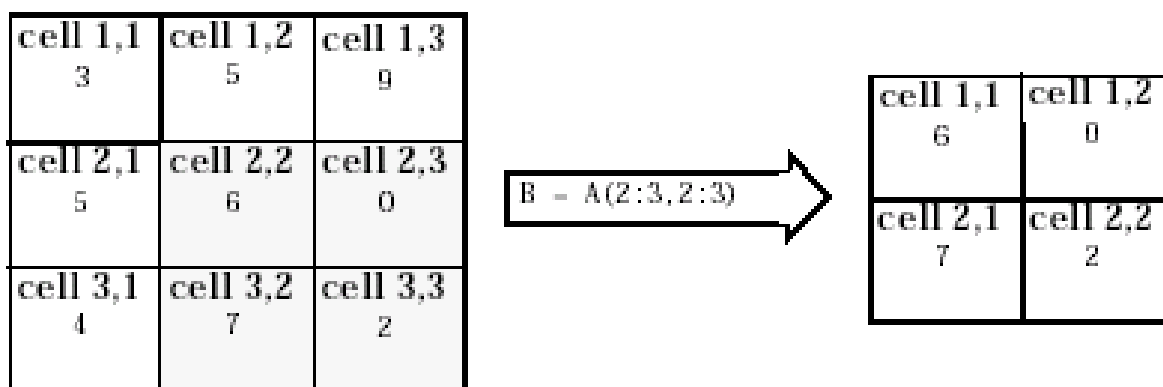
$$d = N\{1,1\}(2,2)$$

что даст

$$d = 5$$

Обращение к подмножествам массива ячеек

Для присваивания любого множества ячеек некоторой переменной, нужно воспользоваться индексацией содержимого ячеек. При этом оператор двоеточия служит для доступа к подмножествам ячеек в пределах массива ячеек.



Удаление ячеек

Вы можете удалить полностью любую размерность массива ячеек с использованием одного выражения. Как и в стандартном случае удаления массивов, нужно использовать векторное индексирование при удалении строк или столбцов массива ячеек, с приравниванием данной размерности пустой матрице, например

$$A(:, 2) = []$$

При удалении ячеек, фигурные скобки вообще не используются в соответствующих выражениях.

Изменение формы (размерностей) массива ячеек

Как и в случае любых других массивов, для изменения формы массива ячеек можно применить функцию **reshape**. При этом общее число ячеек должно остаться тем же, то есть вы не можете использовать данную функцию для добавления или удаления ячеек.

```
A = cell(3, 4);
size(A)
ans =
    3    4

B = reshape(A,6,2);
size(B)
ans =
    6    2
```

Замена списка переменных массивами ячеек

Массивы ячеек могут быть использованы для замены списка переменных MATLAB-а в следующих случаях:

- В списке входных аргументов.
- В списке выходных переменных.
- В операциях отображения на дисплей.
- При конструировании массивов (квадратные скобки и фигурные скобки).

Если вы используете оператор двоеточия для индексации набора ячеек в сочетании с фигурными скобками, то MATLAB обращается с каждой ячейкой как с отдельной переменной. Например, допустим вы имеете массив ячеек **T**, где каждая ячейка содержит отдельный вектор. Выражение **T{1:5}** эквивалентно списку векторов в первых пяти ячейках массива **T**, то есть оно равносильно записи

[**T{1}** , **T{2}** , **T{3}** , **T{4}** , **T{5}**]

Рассмотрим массив ячеек **C**:

```
C(1) = {[1 2 3]};
C(2) = {[1 0 1]};
C(3) = {1:10};
C(4) = {[9 8 7]};
C(5) = {3};
```

Для свертки векторов в **C(1)** и **C(2)** с использованием функции **conv**, нужно записать

```
d = conv(C{1:2})

d =
    1    2    4    2    3
```

Для вывода на дисплей векторов со второго по четвертый введем

C{2:4}

Это даст

```
ans =  
      1  0  1  
  
ans =  
      1  2  3  4  5  6  7  8  9 10  
  
ans =  
      9  8  7
```

Аналогично, вы можете создать новый числовой массив используя выражение

B = [C{1}; C{2}; C{4}]

что приводит к

```
B =  
      1  2  3  
      1  0  1  
      9  8  7
```

Вы можете также использовать соответствующую индексацию в левой части оператора присваивания для создания нового массива ячеек, где каждая ячейка содержит один выходной аргумент

[D{1:2}] = eig (B)

```
D =  
[3x3 double] [3x3 double]
```

Напомним, что при задании двух выходных аргументов, выходом функции **eig(B)** является модальная матрица, составленная из нормированных собственных векторов матрицы **B** и диагональная матрица собственных значений. Вы можете вывести в командное окно действительные значения собственных векторов и значений вводя **D{1}** и **D{2}**.

Применение функций и операторов

Для применения функций или операторов к содержимому ячеек нужно воспользоваться соответствующей индексацией. Например, зададим массив ячеек **A**

```
A{1, 1} = [1 2; 3 4];  
A{1, 2} = randn (3,3);  
A{1, 3} = 1 : 5;
```

Тогда, для применения функции **sum** к содержимому первой ячейки массива запишем

B = sum (A{1,1})

Что приводит к следующему результату

$$B = \begin{matrix} & 4 & 6 \\ 4 & & \\ 6 & & \end{matrix}$$

Для применения той же функции к нескольким ячейкам не вложенных массивов ячеек, нужно применить цикл:

```
for i = 1:length(A)
    M{i} = sum(A{1,i});
end
```

Организация данных в массивах ячеек

Массивы ячеек являются полезными для создания базы данных, состоящих из массивов различных значений и типов. Массивы ячеек являются предпочтительнее структур в приложениях, где:

- Вам нужен доступ ко многим полям данных при помощи одного обращения.
- Вы хотите иметь доступ к подмножеству данных в виде списка значений.
- У вас нету фиксированного набора имен полей.
- Вам приходится часто удалять поля из структуры.

Как пример обращения к набору множества полей при помощи одного выражения допустим, что ваши данные состоят из:

- Массива размера 3x3, состоящего из измерений, полученных экспериментально.
- Строки из 15 символов, содержащей имя инженера.
- Массива размера 3x4x5, содержащего записи измерений за последние 5 экспериментов.

Для многих приложений, наилучшим способом создания базы данных являются структуры. Однако, если вы постоянно имеете дело только с первыми двумя полями данных, то массив ячеек может быть более удобным для целей индексации.

Приведенный ниже пример показывает как можно обратиться к первым двум элементам массива ячеек **TEST**.

```
[newdata, name] = deal (TEST{1:2})
```

а следующий пример демонстрирует то же при организации данных в виде структуры с тем же именем **TEST**:

```
newdata = TEST.measure
name = TEST.name
```

Вложение массивов ячеек

Массив ячеек может содержать другой массив ячеек и даже массив массивов ячеек (Массивы, не содержащие другие массивы ячеек называются *листовыми ячейками (leaf cells)*.) Для создания вложенных массивов ячеек вы можете использовать вложенные фигурные скобки, функцию **cell**, или непосредственное применение операторов присваивания.

Создание вложенных массивов при помощи вложенных фигурных скобок

Для указанной в заголовке цели достаточно вложить в требуемую ячейку пару фигурных скобок. Например, введем следующие команды

```
clear A
A(1,1) = {magic(5)};
A(1,2) = { { [ 5 2 8; 7 3 0; 6 7 3] 'Test 1'; [2 - 4i 5 + 7i] {17 []} } }
```

что даст

```
A =
[5x5 double] {2x2 cell}
```

Отметим, что правая часть второго оператора присваивания заключена в две пары фигурных скобок. Первая пара характеризует ячейку **cell (1,2)** массива ячеек **A**. Второй “набор” скобок представляет массив ячеек размера 2x2 внутри внешней ячейки.

Создание вложенных массивов при помощи функции **cell**

Для вложения массива ячеек при помощи функции **cell**, нужно назначить выход функции **cell** существующей ячейке. Например,

1. Создадим пустой массив размера 1x2

```
A = cell (1, 2);
```

2. Создадим массив ячеек размера 2x2 внутри **A(1,2)**.

```
A(1,2) = {cell(2,2)};
```

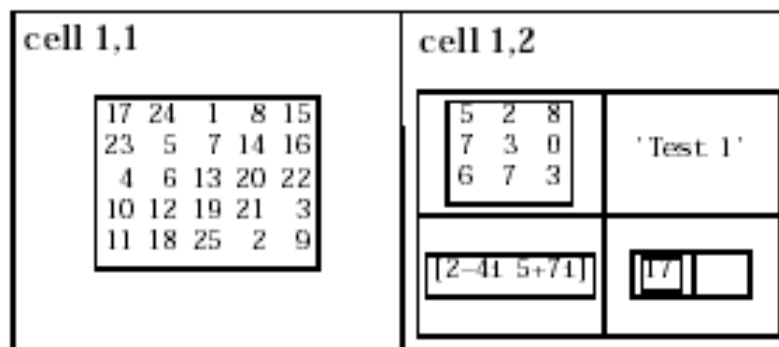
3. Заполним массив **A**, используя выражения

```
A(1,1) = {magic(5)};
A{1,2}(1,1) = {[5 2 8; 7 3 0; 6 7 3]};
A{1,2}(1,2) = {'Test 1'};
A{1,2}(2,1) = {[2-4i 5+7i]};
A{1,2}(2,2) = {cell(1,2)}
A{1,2}{2,2}(1) = {17};
```

Отметим использование фигурных скобок до последнего уровня вложенных индексов. Вы также можете конструировать вложенные массивы ячеек непосредственно с использованием операторов присваивания, как это показано в шаге 3 выше.

Индексация вложенных массивов ячеек

Для индексации вложенных ячеек нужно объединить выражения индексов. Первый набор индексов обеспечивает доступ к верхнему уровню ячеек, а последующие наборы скобок обеспечивают последовательный доступ к последующим уровням. Например, следующий массив имеет три уровня вложения



- Для доступа к массиву 5x5 ячейке (1,1) используйте **A{1,1}**.
- Для доступа к массиву 3x3 в позиции (1,1) ячейки (1,2) используйте **A{1,2}{1,1}**.
- Для доступа к ячейке 2x2 в ячейке (1,2) используйте **A{1,2}**.
- Для доступа к пустой ячейке в позиции (2,2) ячейки (1,2) запишем **A{1,2}{2,2}{1,2}**.

Преобразования между массивами ячеек и числовыми массивами

Для перехода от формата массива ячеек к числовому массиву следует воспользоваться программой, включающей цикл. Например, создадим массив ячеек **F**:

```
F{1,1} = [1 2; 3 4];
F{1,2} = [-1 0; 0 1];
F{2,1} = [7 8; 4 1];
F{2,2} = [4i 3+2i; 1 - 8i 5];
```

Используем теперь три вложенных цикла для копирования содержимого массива **F** в числовой массив **NUM**.

```
for k = 1:4
    for i = 1:2
        for j = 1:2
            NUM(i,j,k) = F{k}(i,j);
        end
    end
end
```

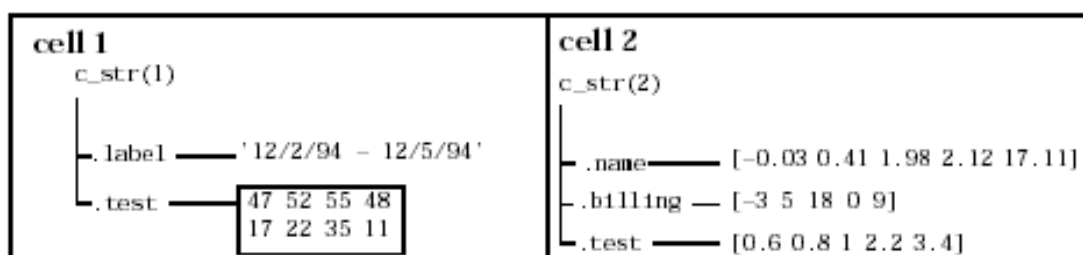
Аналогично, вы должны использовать петли **for** для присваивания каждого значения числового массива одной ячейке массива ячеек:

```
G = cell(1,16);
for m = 1:16
    G{m} = NUM(m);
end
```

Массивы ячеек, содержащие структуры

Для хранения групп структур с различной архитектурой полей можно использовать массивы ячеек

```
c_str = cell(1,2);
c_str{1}.label = '12/2/94 - 12/5/94';
c_str{1}.obs = [47 52 55 48; 17 22 35 11];
c_str{2}.xdata = [-0.03 0.41 1.98 2.12 17.11];
c_str{2}.ydata = [-3 5 18 0 9];
c_str{2}.zdata = [0.6 0.8 1 2.2 3.4];
```



Ячейка 1 массива **c_str** содержит структуру с двумя полями, где в одном поле хранится строка символов, а во втором - вектор. Ячейка 2 содержит структуру с тремя полями векторов. При создании массивов ячеек, содержащих структуры, вы должны применить индексирование фигурными скобками. Аналогично, вы должны применить фигурные скобки для получения структур, содержащихся внутри ячеек. Общий синтаксис при этом имеет вид:

cell_array{index}.field

Например, чтобы получить содержимое поля **label** структуры в ячейке 1 нужно записать

c_str{1}.label

Многомерные массивы ячеек

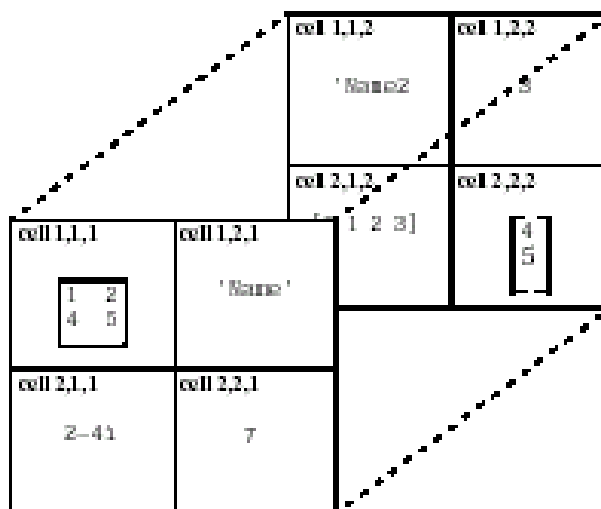
Как и в случае числовых массивов, общие принципы создания многомерных массивов ячеек основаны на распространении понятия двумерных массивов ячеек. Для создания многомерных массивов ячеек вы можете применить функцию **cat**, совершенно аналогично ее применению в случае числовых массивов.

Например, создадим простой трехмерный массив ячеек **C** из двух массивов **A** и **B**:

```
A{1,1} = [1 2; 4 5];
A{1,2} = 'Name';
A{2,1} = 2 - 4i;
A{2,2} = 7;
B{1,1} = 'Name2';
B{1,2} = 3;
B{2,1} = 0:1:3;
B{2,2} = [4 5]';
```

C = cat(3,A,B);

Общая структура индексации массива ячеек **C** имеет вид

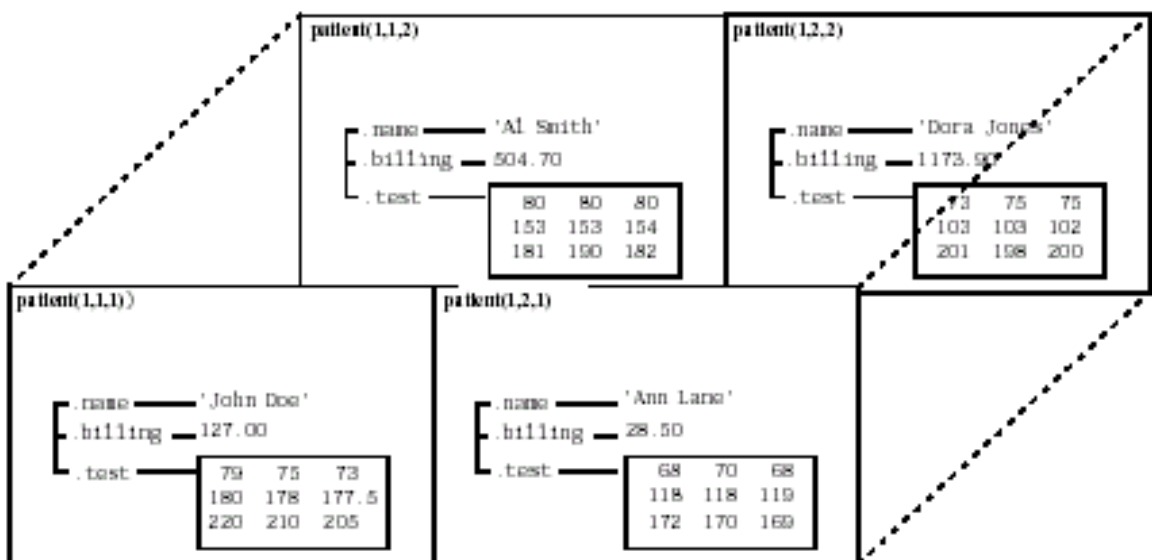


Многомерные массивы структур

Многомерные массивы структур являются распространением обычных двумерных, то есть плоских структур. Подобно другим типам многомерных массивов, вы можете строить их как прямым присваиванием, так и применением функции **cat**.

```
patient(1,1,1).name = 'John Doe';  
patient(1,1,1).billing = 127.00;  
patient(1,1,1).test = [79 75 73; 180 178 177.5; 220 210 205];  
patient(1,2,1).name = 'Ann Lane';  
patient(1,2,1).billing = 28.50;  
patient(1,2,1).test = [68 70 68; 118 118 119; 172 170 169];  
patient(1,1,2).name = 'Al Smith';  
patient(1,1,2).billing = 504.70;  
patient(1,1,2).test = [80 80 80; 153 153 154; 181 190 182];  
patient(1,2,2).name = 'Dora Jones';  
patient(1,2,2).billing = 1173.90;  
patient(1,2,2).test = [73 75 75; 103 103 102; 201 198 200];
```

Геометрически данную структуру можно отобразить следующим образом



Применение функций к многомерным массивам структур

Для применения функций к многомерным массивам структур, нужно использовать индексирование полей. Например, найдем сумму столбцов структуры `test` в `patient(1,1,2)`:

```
sum((patient(1,1,2).test));
```

Аналогично, просуммируем все поля `billing` в многомерном массиве `patient`:

```
total = sum([patient.billing]);
```


ПРОГРАММИРОВАНИЕ НА MATLAB-е

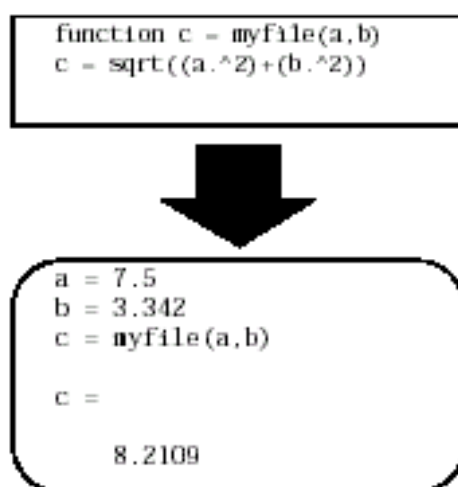
Программирование на языке MATLAB : Быстрый старт

М-файлы языка MATLAB могут быть или *сценариями* (*scripts*), которые просто выполняют серию операторов (выражений), или же они могут быть *функциями* (*functions*), допускающими также входные аргументы и выходные переменные. Вы можете создать М-файлы используя текстовый редактор и затем использовать их как любую другую функцию или команду системы MATLAB.

В простейшем случае процесс выглядит так:

1. Вы создаете М-файл используя текстовый редактор.
2. Вызываете М-файл из командной строки или же из другого М-файла.

Схематически это можно отобразить следующим образом:



Типы М-файлов

Как указывалось выше, имеется два типа М-файлов, общие свойства которых приведены в таблице

М-сценарии	М-функции
<ul style="list-style-type: none">• Не допускают входных и выходных переменных	<ul style="list-style-type: none">• Допускают входные и выходные аргументы
<ul style="list-style-type: none">• Оперируют в данными из рабочей области	<ul style="list-style-type: none">• Внутренние переменные по умолчанию являются локальными по отношению к функциям
<ul style="list-style-type: none">• Предназначены для автоматизации последовательности шагов, которые нужно выполнять много раз	<ul style="list-style-type: none">• Предназначены для расширения возможностей языка MATLAB (библиотеки функций, пакеты прикладных программ)

Что такое М-файл ?

В данном разделе мы рассмотрим основные части, из которых состоит М-функция. Допустим, мы имеем следующую функцию **fact**, вычисляющую факториал целого числа:

```
function f = fact (n)    % Строка определения функции
```

```
% FACT Factorial.      % Первая строка помощи (H1 line)
% FACT(N) returns the factorial of N, H! % Текст помощи (Help text)
% usually denoted by N!
% Put simply, FACT(N) is PROD(1:N).
```

```
f = prod(1:n); % Тело функции
```

Эта функция имеет некоторые элементы, которые являются общими для всех функций системы MATLAB:

- *Строка определения функции.* Эта строка задает имя функции, а также число и порядок входных и выходных аргументов.
- *Строка H1 (H1 line).* H1 обозначает «первую строку» помощи. MATLAB выводит эту строку в командное окно, когда вы пользуетесь функцией **lookfor** или запрашиваете помощь по всей директории.
- *Текст помощи (Help text).* MATLAB выводит в командное окно данный текст вместе со строкой H1, когда вы запрашиваете помощь по конкретной функции, то есть вводите **help Имя_Функции**.
- *Тело функции.* Эта часть функции содержит коды (команды), которые выполняют вычисления и определяют значения всех выходных переменных.

Обеспечение помощи для вашей программы

Вы можете снабдить пользователя информацией (помощью) о вашей программе, путем включения раздела текста помощи в начало М-файла. Этот раздел начинается со строки, следующей непосредственно за строкой определения функции и заканчивается на первой пустой строке, или строке тела функции. Каждая строка текста (эти строки окрашены в зеленый цвет) помощи должна начинаться символом процента (%). MATLAB выводит в командное окно данный текст каждый раз когда вы вводите

help Имя_Функции

Вы можете также написать текст помощи для всей директории, путем создания файла со специальным именем **Contents.m**, который находится в вашей директории. Этот файл должен содержать только строки комментариев, то есть каждая строка должна начинаться со знака процента. MATLAB выводит на дисплей строки файла **Contents.m** всякий раз, когда вы вводите в командное окно строку

help Имя_Директории

Если данная директория не содержит файл **Contents.m**, то при вводе **help Имя_Директории** в командное окно выводится первая строка помощи (H1 line) для каждого файла директории.

Создание М-файлов: Использование текстовых редакторов

М-файлы представляют собой обычные текстовые файлы, которые вы создаете с использованием текстового редактора. MATLAB содержит встроенный редактор, хотя в принципе можно воспользоваться любым другим текстовым редактором.

Внимание! Для вызова редактора нужно в меню **File** выбрать **New** и затем **M-File**.

Другой способ вызова редактора М-файла из командной строки состоит в использовании функции **edit**. For example, при вводе

```
edit foo
```

MATLAB открывает встроенный текстовый редактор на файле **foo.m**. Если не указать имени файла, то будет вызван редактор с новым, незаглавленным файлом. Вы можете создать функцию **fact**, приведенную выше, путем открытия вашего текстового редактора, ввода показанных строк, и запоминанием текста в файле под названием **fact.m** в вашей текущей директории.

После того как вы создали этот файл, его можно найти в списке файлов вашей текущей директории, для чего надо ввести команду

what

Можно также распечатать в командном окне файл командой

type fact

Наконец, вы можете вычислить факториал любого целого числа, например, 5-и

```
fact(5)
ans =
    120
```

Внимание! Сохраняйте все созданные или измененные вами М-файлы в директории (каталоге), который не находится в дереве каталогов MATLAB-а. Если вы сохраните ваши М-файлы в дереве каталогов MATLAB-а, они могут быть уничтожены при установке новой версии MATLAB-а.

Сценарии

Сценарии являются простейшим типом М-файлов, поскольку они не имеют входных или выходных аргументов. Они полезны для автоматизации последовательности команд, таких как обычные вычисления, которые приходится часто выполнять в командном окне. Сценарии работают над существующими данными в рабочем пространстве; вы также можете создавать новые данные при помощи сценариев. Все переменные, созданные в результате выполнения сценариев, остаются в главном рабочем пространстве MATLAB-а, так что вы можете использовать их для дальнейших вычислений.

Простой пример сценария

Приведенные ниже выражения вычисляют функцию **rho** для нескольких тригонометрических функций угла **theta**, и строят серию графиков в полярной системе координат

```
% An M-file script to produce          % Линия комментариев
% "flower petal" plots                  %
theta = -pi:0.01:pi;                    % Вычисления
rho(1,:) = 2*sin(5*theta).^2;
rho(2,:) = cos(10*theta).^3;
rho(3,:) = sin(theta).^2;
rho(4,:) = 5*cos(3.5*theta).^3;
for i = 1:4
    polar(theta,rho(i,:))                % Вывод на графики
    pause
end
```

Попробуйте ввести эти команды в М-файл, названный **petals.m**. Этот файл является теперь сценарием MATLAB-а. Ввод команды **petals** (лепестки) в командной строке MATLAB –а приводит к выполнению команд сценария. Команда **pause** приостанавливает выполнение

цикла до нажатия какой-либо клавиши (например, **Return**). Таким образом, после того как сценарий отображает один график из четырех, нажатие клавиши **Return** приводит к появлению следующего. Здесь мы не имеем входных или выходных переменных; сценарий **petals** создает требуемые ему переменные в основном рабочем пространстве MATLAB-а. Когда выполнение сценария завершено, все созданные переменные (**i**, **theta**, и **rho**) остаются в рабочем пространстве. Вы можете убедиться в этом, вводя команду **whos** в командной строке.

Функции

Функции представляют собой М-файлы, которые принимают входные аргументы и выдают выходные. Они работают над переменными в своем собственном рабочем пространстве, которое не совпадает с основным рабочим пространством, доступным из командной строки MATLAB-а.

Простой пример функции

Функция **average** является простым М-файлом, который вычисляет среднее значение элементов вектора.

```
function y = average(x)
% AVERAGE Mean of vector elements.
% AVERAGE(X), where X is a vector, is the mean of vector elements.
% Non-vector input results in an error.

[m,n] = size(x);
if ~(m == 1 | n == 1) | (m == 1 & n == 1)
    error('Input must be a vector')
end
y = sum(x) / length(x) ;    % Фактические вычисления
```

При вводе не векторной величины, данная функция выдает сообщение об ошибке (более точно, на дисплей выводится фраза «**Вход должен быть вектором**»). Вы можете ввести эти команды в М-файл, названный **average.m**. Функция **average** допускает единственный вход и возвращает единственный выходной аргумент. Для обращения к данной функции, введите

```
z = 1:99;
average(z)
```

что даст следующий результат

```
ans =
    50
```

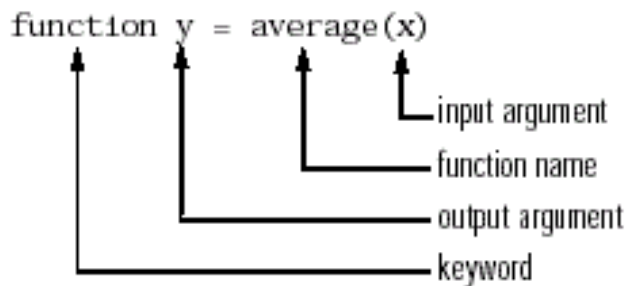
Основные части синтаксиса М-функций

Функции в общем случае состоят из следующих частей:

- Строка определения функции (The Function Definition Line)
- Строка помощи H1 (The H1 Line)
- Текст помощи (Help Text)
- Тело функции (The Function Body)
- Комментарии (Comments)

Строка определения функции

Строка определения функции информирует систему MATLAB, что М-файл содержит функцию, и задает последовательность входных и выходных переменных. Для функции **average** эта строка имеет следующий вид:



где **input argument** – входной аргумент;
function name – имя функции;
output argument – выходной аргумент;
keyword – зарезервированное слово;

Все функции MATLAB-а имеют линию определения функции, соответствующую данной схеме. Если функции имеют несколько выходных переменных, нужно заключить список этих переменных в квадратные скобки. Входные переменные, даже если их несколько, всегда заключаются в обычные скобки. Вот пример более сложной функции

function [x,y,z] = sphere(theta, phi, rho)

Если функция не имеет выходных переменных, оставьте выход пустым, например,

function printresults(x)

или используйте пустые квадратные скобки

function [] = printresults(x)

Переменные, которые вы передаете функции не обязательно должны иметь то же имя, что и в линии определения функции.

Строка помощи H1

Строка H1, названная так потому что она является первой строкой текста помощи (Help text), является линией комментария, которая следует непосредственно за строкой определения функции. Поскольку она состоит из текста комментария, строка H1 начинается с символа процента (%). Для функции **average** эта строка имеет вид

**% AVERAGE Mean of vector elements.
 (СРЕДНЕЕ ЗНАЧЕНИЕ Вычисление среднего значения векторов)**

Эта строка является первой строкой текста, который появляется при вводе пользователем в командной строке команды

help function_name
 (help имя_функции)

Далее, функция **lookfor** производит поиск и выводит в командное окно только строку H1. Так как данная строка обеспечивает важную обобщающую информацию о М-файле, очень важно сделать ее по возможности более описательной.

Текст помощи

Вы можете создать возможность оперативной помощи (справки) для вашей М-функции, путем ввода одной или большего числа строк комментария, начинающихся непосредственно за строкой H1. Текст помощи для функции **average** имеет вид

```
% AVERAGE(X), where X is a vector, is the mean of vector elements.  
% Nonvector input results in an error.  
(% СРЕДНЕЕ(X), где X является вектором, вычисляет среднее значение  
%элементов вектора. Не векторный вход приводит к ошибке).
```

Когда вы вводите **help function_name**, MATLAB выводит линии комментариев, которые находятся между строкой определения функции и первой строкой не комментариев (выполняемой или пустой строкой). MATLAB игнорирует любые линии комментариев, которые появляются за данным блоком текста помощи. Например, напечатав **help sin** получим

```
SIN      Sine.  
SIN(X) is the sine of the elements of X.  
(SIN(X) является синусом элементов массива X)
```

Тело функции

Тело функции содержит все коды системы MATLAB, которые осуществляют вычисления и определяют значения выходных переменных. Выражения в теле функции состоят из обращений к другим функциям, программных конструкций типа команд циклов, ввода и вывода, вычислений, операторов присваивания, комментариев и пустых строк. Например, тело функции **average** содержит нескольких простых программных выражений:

```
[m,n] = size(x);  
if ~(m == 1) | (n == 1) | (m == 1 & n == 1) % Flow control  
    error('Input must be a vector') % Error message display  
end  
y = sum(x)/length(x); % Computation and assignment
```

Комментарии

Как было указано ранее, строки комментариев начинаются с символа процента (%). Строки комментариев могут быть в любом месте М-файла, а также вы можете добавить комментарии к концу строки кодов программы. Например,

```
% Add up all the vector elements.  
y = sum(x)    % Use the sum function.  
( % Суммирование всех элементов вектора.  
y = sum(x)    % Используйте функцию sum)
```

Первая строка комментариев, следующая непосредственно за строкой определения функции рассматривается как строка H1 данной функции. Строка H1 и любые строки комментариев, непосредственно следующие за H1, составляют запись оперативной помощи для данного файла. В дополнение к строкам комментариев, вы можете вводить пустые строки в любом месте М-файла. Пустые строки игнорируются. С другой стороны, пустая строка может обозначать конец текста помощи.

Имена функций

Имена функций в MATLAB-е имеют те же ограничения, что и имена переменных. MATLAB использует первые 32 символа имени. Имена функций должны начинаться с буквы; остальные знаки могут быть любой комбинацией букв, цифр и символов подчеркивания. Некоторые операционные системы могут вводить свои ограничения на длину имен функций.

Название текстового файла, который содержит функцию MATLAB-а, состоит из имени функции с добавленным расширением **.m**. Например, **average.m**. Если имя файла и имя функции в ее строке определения отличаются, то внутреннее имя игнорируется.

Однако, несмотря на то что имя функции, заданное в ее строке определения, не обязательно должно совпадать с именем файла, настоятельно рекомендуется чтобы вы использовали одно и то же имя для файла и функции.

Как работает функция

Вы можете вызвать М-функцию как из командной строки MATLAB-а, так и из другого М-файла. Убедитесь, что вы включили все необходимые аргументы, заключив входные аргументы в обычные скобки, а выходные – в квадратные.

Определение имени функции

Когда MATLAB сталкивается с новым именем, он осуществляет следующую последовательность шагов:

1. Проверяет, не является ли имя *переменной*.
2. Проверяет, не является ли имя *подпрограммой* (*subfunction*), то есть функцией MATLAB-а, которая находится в пределах того же М-файла, что и вызываемая функция.
3. Проверяет, не является ли имя *частной функцией* (*private function*), то есть функцией, которая находится в специальной директории под названием *Private* (*private directory*), то есть директории доступной только для М-файлов в пределах той же директории где она сама находится.
4. Проверяет, находится ли данная на пути доступа MATLAB-а. MATLAB обращается к первому встреченному файлу с заданным именем. Если вы дублируете имена функций, MATLAB обращается к первой встреченной на основе приведенной выше процедуры.

Что происходит при вызове функцию

Когда вы вызываете М-файл из командной строки или же из пределов другой М-функции, MATLAB осуществляет синтаксический анализ функции и преобразует ее в псевдокод, который запоминается в памяти. Это исключает необходимость повторного анализа функции при каждом последующем ее вызове в пределах данного сеанса работы. Псевдокод сохраняется в памяти до тех пор пока вы не удалите его с помощью команды **clear**, или пока вы не выйдете из MATLAB-а.

В приведенной ниже таблице даны основные варианты применения команды **clear** для удаления любых функций из рабочего пространства MATLAB-а.

Синтаксис	Описание
clear function_name	Удаляет заданную функцию из рабочего пространства
clear functions	Удаляет все скомпилированные М-функции
clear all	Удаляет все переменные и функции

Создание Р-кодов файлов

Вы можете запомнить предварительно скомпилированные функции или сценарии, называемые псевдокодами (P-code) файлов, для использования их в последующих сеансах работы. Например, команда

pcode average

компилирует функцию **average.m** и запоминает полученный псевдокод в файле называемом **average.p**. Это позволяет MATLAB-и исключить операцию компилирования при первом вызове функции в каждом сеансе работы. В принципе, MATLAB осуществляет компиляцию весьма быстро, так что запоминание функции в виде псевдокода редко дает большой выигрыш в быстродействии. Единственная ситуация где псевдокод действительно дает ощутимый выигрыш во времени, связана с применением сложных **Графических Интерфейсов Пользователя (GUI)** в различных приложениях. В этом случае множество М-файлов должны быть скомпилировано прежде чем GUI станет видимым. Другая ситуация, где использование псевдокода является оправданным, имеет место при необходимости сохранить права собственности, то есть когда вы хотите исключить возможность применения вашего алгоритма другими лицами.

Как MATLAB передает аргументы функции

С точки зрения программиста создается впечатление, что MATLAB передает функции все аргументы в виде их значений. В действительности, однако, MATLAB передает значения только тех аргументов, которые изменяются данной функцией. Если функция не изменяет соответствующий аргумент, а просто использует его при вычислениях, MATLAB передает аргумент в виде ссылки на него (на его расположение в памяти) с целью оптимизации использования памяти.

Рабочие пространства функций

Каждая М-функция имеет в памяти свое рабочее пространство, отдельное от основного рабочего пространства MATLAB-а, в котором она работает. Это пространство называется рабочим пространством функции, причем разные функции имеют разные рабочие пространства. При использовании MATLAB-а, вы имеете доступ только к тем переменным, которые находятся в вызываемом контексте, будь это основное рабочее пространство или рабочее пространство какой-то функции. Переменные, которые вы передаете функции, должны быть расположены в пространстве вызова, и, в свою очередь, функция возвращает выходные аргументы в то же самое рабочее пространство вызова. Вы можете, однако, определить переменные как *глобальные*, что дает возможность доступа к ним из разных рабочих пространств.

Проверка числа аргументов функции

Функции **nargin** и **nargout** позволяют вам определить число входных и выходных аргументов функции. Вы можете использовать эти функции с условными операторами для выполнения различных задач в зависимости от числа аргументов. Например,

```
function c = testarg1(a,b)  
if (nargin == 1)  
    c = a.^2;  
elseif (nargin == 2)  
    c = a + b;  
end
```

При одном входном аргументе, данная функция вычисляет квадрат входной величины. Если заданы два входных аргумента, функция осуществляет их сложение.

Передача переменного числа аргументов

Функции **varargin** и **varargout** дают возможность передачи функции любого переменного числа аргументов или возвращать переменное число выходные аргументов. При использовании функции **varargin** MATLAB объединяет все заданные входные аргументы в массив ячеек. Если вы используете функцию **varargout**, то ваша программа должна обеспечить объединение выходных переменных в массив ячеек, с тем чтобы MATLAB имел возможность вернуть их в пространство вызова. Ниже дан пример функции, которая принимает любое число двумерных векторов, и наносит на графике линию, соединяющую соответствующие точки.

```
function testvar (varargin)
for i = 1:length (varargin)
    x(i) = varargin{i}(1);
    y(i) = varargin{i}(2);
end
xmin = min(0,min(x));
ymin = min(0,min(y));
axis([xmin fix(max(x)) + 3 ymin fix(max(y)) + 3])
plot(x,y)
```

Функция **testvar** работает с различным числом входных переменных; например, вы можете ввести два различных набора данных

```
testvar ([2 3], [1 5], [4 8], [6 5], [4 2], [2 3])
testvar ([-1 0], [3 -5], [4 2], [1 1])
```

Распаковка содержимого функции varargin

Поскольку функция **varargin** содержит все входные аргументы в виде массива ячеек, для извлечения данных необходимо использовать соответствующую индексацию. Например,

```
y(i) = varargin{i} (2);
```

Индексация ячеек имеет два набора компонент – первый набор указывает ячейку и заключен в фигурные скобки, а второй набор относится к содержимому ячейки и заключен в обычные скобки. В приведенном выше операторе выражение **{i}** обозначает обращение к **i**-ой ячейке в **varargin**, а выражение **(2)** представляет второй элемент содержимого выбранной ячейки.

Упаковка выходных переменных в функцию varargout

Когда вы хотите использовать произвольное число выходных аргументов, вы должны предусмотреть процедуру упаковки выходных переменных в массив ячеек **varargout**. При этом, для определения конкретного числа вызываемых выходных аргументов используйте функцию **nargout**. Например, приведенный ниже пример принимает входной массив в виде двух столбцов, где первый столбец характеризует набор данных по оси **x**, а второй столбец – соответствующий набор данных по оси **y**. Данные наборы разбиваются на отдельные пары векторов **[xi yi]**, которые вы можете передать описанной выше функции **testvar**.

```
function [varargout] = testvar2 (arrayin)
for i = 1:nargout
    varargout {i} = arrayin (i, :)
end
```

Оператор присваивания в цикле **for** использует синтаксис индексации массивов ячеек. Вот пример применения функции **testvar2**:

```
a = {1 2; 3 4; 5 6; 7 8; 9 0};  
[p1, p2, p3, p4, p5] = testvar2(a);
```

Место функций **varargin** и **varargout** в списке аргументов

Функции **varargin** или **varargout** должны быть последними в списке аргументов, при этом они могут быть расположены после любого числа входных или выходных переменных. Это значит, что в строке определения функции следует сперва указать требуемые входные или выходные аргументы. Например, следующие строки определения функций показывают правильное применение **varargin** и **varargout**.

```
function [out1,out2] = example1(a,b,varargin)  
function [i,j,varargout] = example2(x1,y1,x2,y2,flag)
```

Локальные и глобальные переменные

Каждая исполняемая функция MATLAB-а, определенная некоторым М-файлом, имеет свои собственные локальные переменные расположенные в своем рабочем пространстве, которые отделены от локальных переменных других функций и переменных в основном рабочем пространстве. Однако, если несколько функций и, возможно, основное рабочее пространство, объявляют некоторую конкретную переменную *глобальной*, то все эти функции и основное рабочее пространство будут иметь доступ к данной переменной. Любое изменение глобальной переменной, произведенное в пространстве какой-либо одной функции, немедленно воспринимается всеми остальными функциями, где эта переменная объявлена глобальной. Допустим, вы хотите изучить эффект изменения коэффициентов взаимосвязей α и β , в дифференциальном уравнении Лотки-Вольтера (Lotka-Volterra), известного как модель хищника-жертвы.

$$\begin{aligned}dy_1/dt &= y_1 - \alpha y_1 y_2 \\ dy_2/dt &= y_2 - \beta y_1 y_2\end{aligned}$$

Создадим М-файл **lotka.m**.

```
function yp = lotka(t,y)  
global ALPHA BETA  
yp = [y(1) - ALPHA*y(1)*y(2); -y(2) + BETA*y(1)*y(2)];
```

Затем введем последовательно в командное окно следующие выражения

```
global ALPHA BETA  
ALPHA = 0.01  
BETA = 0.02  
[t,y] = ode23('lotka',0,10,[1; 1]);  
plot(t,y)
```

Объявление переменных **ALPHA** и **BETA** глобальными в командной строке позволяет менять соответствующие значения внутри функции заданной файлом **lotka.m**. Интерактивное изменение данных переменных в командном окне приводит к получению новых решений без каких-либо редактирований текста файла.

Для работы в ваших приложениях с глобальными переменными следует:

- Объявить соответствующую переменную глобальной в каждой функции, где предусмотрено ее использование. Для обеспечения доступа к глобальной переменной из командного окна нужно объявить данную переменную глобальной также и в командной строке.
- В каждой функции объявите переменную глобальной до первого появления ее имени в тексте файла. Обычно рекомендуется объявлять переменные глобальными в начале М-файла.

Глобальные переменные в MATLAB-е обычно имеют более длинные имена и иногда записываются заглавными буквами. Это не является настоятельным требованием, но упрощает чтение файлов и уменьшает риск случайного изменения глобальной переменной.

Перманентные переменные (Persistent Variables)

Переменная может быть объявлена *перманентной* (постоянной) – при этом она не меняет своего значения между ее последовательными вызовами. Перманентные переменные могут быть использованы только в пределах определенной функции. Эти переменные остаются в памяти до удаления М-файла из памяти или его изменения. Во многих отношениях перманентные переменные аналогичны глобальным, за тем исключением, что их имя не находится в глобальном рабочем пространстве, а их значение сбрасывается при изменении М-файла или удалении из памяти.

Для работы с перманентными переменными в MATLAB-е предусмотрены три функции:

Функция	Описание
mlock	Исключает возможность удаления М-файла из памяти
munlock	Возвращает М-файлу возможность его удаления из памяти
mislocked	Указывает, может ли М-файл быть удален из памяти

Специальные переменные

Несколько функций возвращают важные специальные значения, которые вы можете использовать в ваших М-файлах.

Функция	Возвращаемое значение
ans	Последний ответ (переменная). Если вы не присваиваете выходной переменной или вычисляемому выражению какое-либо имя, MATLAB автоматически запоминает результат в переменной ans .
eps	Относительная точность вычислений с плавающей запятой. Это допуск, который MATLAB использует при вычислениях.
realmax	Наибольшее число с плавающей запятой.
realmin	Наименьшее число с плавающей запятой.
pi	3.1415926535897...
i, j	Мнимая единица.
inf	Бесконечность. Вычисления вида n/0 где n – любое ненулевое реально число, дает в результате inf .
NaN	Не численное значение (Not-a-Number). Выражения вида 0/0 и inf/inf дают в результате NaN , так же как и арифметические операции содержащие NaN . Выражения типа n/0 , где n явля-

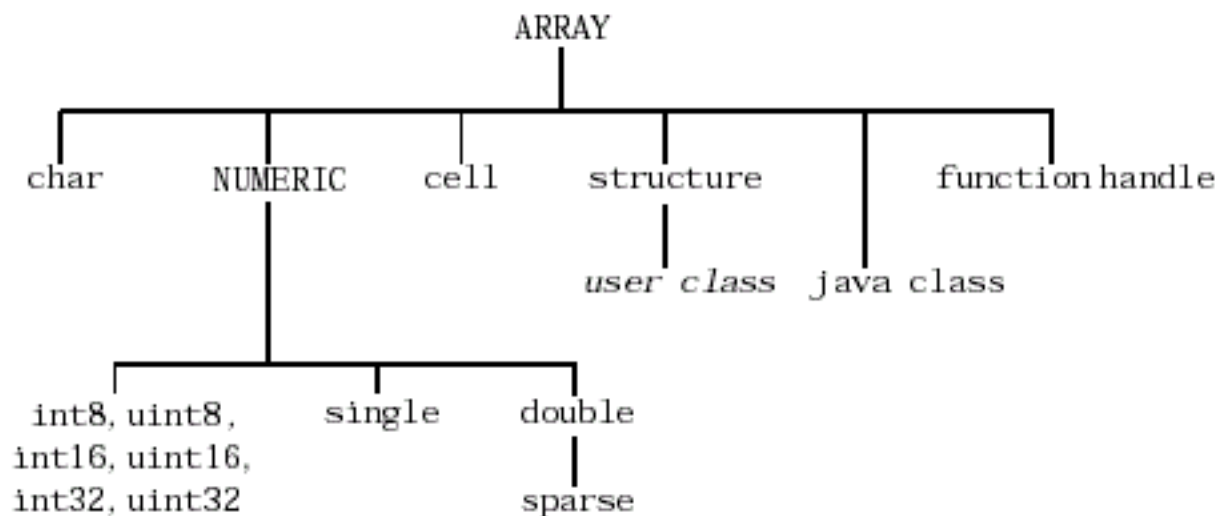
	ется комплексным числом, также возвращают NaN.
computer	Тип компьютера.
version	Строка, содержащая версию MATLAB-а.

Вот несколько примеров, где используются эти переменные.

```
x = 2*pi;
A = [3 + 2i 7 - 8i];
tol = 3*eps;
```

Типы данных

Всего в MATLAB –е имеется 14 базовых типов (или классов) данных. Каждый из этих типов данных является формой массива. Этот массив может иметь минимальный размер 0x0 и может иметь произвольную размерность по любой координате. Двумерные варианты таких массивов называются *матрицами*. Все 14 базовых класса типов данных показаны на приведенной ниже диаграмме. Дополнительно, тип данных, определенных пользователем, показанный ниже как *user class* (класс пользователя), является подмножеством данных типа *структуры*.



Тип данных **char** содержит символы данные в коде Unicode. Строка символов является просто массивом символов размера 1xn. array of characters. Вы можете использовать тип данных **char** для хранения массивов строк, при условии, что все строки массива имеют одинаковую длину (это является следствием того, что все массивы MATLAB-а должны быть прямоугольными). Для хранения массива строк разной длины нужно использовать массив ячеек.

Числовые типы данных включают целые числа со знаком и без знака, числа в формате плавающей запятой одинарной и двойной точности, и разреженные массивы (sparse arrays) двойной точности.

Сказанное ниже сохраняется в силе для всех типов числовых данных в MATLAB-е:

- ❑ Все вычисления в MATLAB-е выполняются с двойной точностью.
- ❑ Целые числа и числа одинарной точности обеспечивают более эффективное использование памяти по сравнению с числами двойной точности.

- ❑ Все типы данных поддерживают базовые операции над массивами, такие как использование индексов и измерение размеров массива.
- ❑ Для выполнения математических операций над целыми числами или массивами с одинарной точностью представления, вы должны превратить их в массивы с двойной точностью при помощи функции **double**.

Операторы

Операторы системы MATLAB делятся на три категории:

- ❑ Арифметические операторы, осуществляющие численные вычисления.
- ❑ Операции отношения, которые осуществляют численное сравнение операндов.
- ❑ Логические операторы, включающие AND (логическое И), OR (логическое ИЛИ), и NOT (логическое отрицание НЕ).

Арифметические операторы

MATLAB обеспечивает следующие арифметические операторы

Операторы	Описание
+	Сложение
-	Вычитание
.*	Умножение
./	Правое деление
.\	Левое деление
+	Унарный плюс (изменение знака объекта)
-	Унарный минус
:	Оператор двоеточия
.^	Степень
.'	Транспонирование
.'	Комплексно-сопряженное транспонирование
*	Матричное умножение
/	Матричное правое деление
\	Матричное левое деление
^	Степень матрицы

Арифметические операторы и массивы

За исключением некоторых матричных операторов, арифметические операторы MATLAB-а работают с соответствующими элементами массивов одинаковой размерности. Для векторов и прямоугольных массивов, оба операнда должны иметь одинаковый размер, или же один из них должен быть скаляром. Если один операнд является скаляром, а второй - нет, MATLAB применяет данный скаляр ко *всем* элементам второго операнда; данное свойство известно как *скалярное расширение (scalar expansion)*.

Следующий пример иллюстрирует свойство скалярного расширения при вычислении произведения скалярного операнда и матрицы

A = magic(3)

A =

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

Введем

3 * A

что дает

```
ans =
    24    3   18
     9   15   21
    12   27    6
```

Операторы отношения

MATLAB обеспечивает следующие операторы отношения

Операторы	Описание
<	Меньше чем
<=	Меньше чем или равно
>	Больше чем
>=	Больше чем или равно
==	Равно
~=	Не равно

Операторы отношения и массивы

Операторы отношения в MATLAB-е сравнивают соответствующие элементы двух массивов с одинаковыми размерностями. Эти операторы всегда действуют поэлементно. В приведенном ниже примере, результирующая матрица показывает, где элемент матрицы **A** равен соответствующему элементу матрицы **B**.

```
A = [2 7 6; 9 0 5; 3 0.5 6];
B = [8 7 0; 3 2 5; 4 -1 7];
```

```
A == B
ans =
     0     1     0
     0     0     1
     0     0     0
```

Для векторов и прямоугольных массивов, оба операнда должны иметь одинаковый размер или один из них должен быть скаляром. В случае когда один операнд является скаляром, а второй – нет, MATLAB проверяет данный скаляр с каждым элементом другого операнда. Те положения, где заданное отношение является истинным, принимают значение 1. Положения, где отношение является ложным, принимают значение 0.

Операторы отношения и пустые массивы

Операторы отношения работают и с массивами, у которых какая-либо размерность равна нулю (что приводит к пустому массиву), если оба массива имеют одинаковый размер или же один из них является скаляром. Однако, выражения вида

```
A == []
```

приводят к ошибке, если только массив **A** не имеет размеры 0x0 или 1x1. Для проверки является ли данный массив пустым, следует использовать специальную функцию **isempty(A)**.

Логические операторы

MATLAB обеспечивает следующие логические операторы

Оператор	Описание
&	AND (логическое И)
	OR (логическое ИЛИ)
~	NOT (логическое НЕ)

Внимание ! В дополнение к этим логическим операторам, в директории **ops** имеются несколько функций, предназначенных для побитовых (поразрядных) логических операций.

Каждый логический оператор имеет специфичный набор правил, которые определяют результат логического выражения:

- Выражения использующие оператор И (&), истинны, если истинны оба операнда. При численных элементах, выражение является истинным, если оба операнда ненулевые. Следующий пример показывает операцию логического И для двух векторов

```
u = [1 0 2 3 0 5];  
v = [5 6 1 0 0 7];
```

```
u & v  
ans =  
0 0 1 0 0 1
```

- Выражения, использующие оператор ИЛИ (|), являются истинными если один из операндов является истинным. Выражения с ИЛИ являются ложными только если ложными являются оба операнда. При численных элементах, выражение является ложным, если только оба операнда равны нулю. Для приведенных выше векторов **u** и **v** имеем

```
u | v  
ans =  
1 1 1 1 0 1
```

- Выражения, использующие оператор ~ выполняют логическое отрицание. Это дает ложный результат, если операнд является истинным и истинный, если операнд является ложным. При численных элементах, любой ненулевой операнд становится нулевым (логическим нулем), а любой нулевой элемент становится равным (логической) единице. Рассмотрим операцию логического отрицания вектора **u**

```
~u  
ans =  
0 1 0 0 1 0
```

Использованием логических операторов с массивами

Логические операторы MATLAB-а сравнивают соответствующие элементы массивов одинаковой размерности. Для векторов или прямоугольных массивов, оба операнда должны иметь одинаковый размер, или один из них должен быть скаляром. Если один из элементов является скаляром, а второй – нет, то здесь также имеет место описанное выше свойство скалярного расширения.

Логические функции

В дополнение к логическим операторам, MATLAB имеет ряд логических функций.

Функция	Описание	Примеры
xor	Выполняет операцию исключающего ИЛИ над своими операндами. При числовых элементах, функция возвращает 1 если один из операндов ненулевой, а второй - нулевой	a = 1; b = 1; xor(a,b) ans = 0
all	Возвращает 1, если все элементы ее аргумента являются истинными или не равны нулю; в противном случае результат равен логическому нулю. Над матрицами функция all работает вдоль столбцов	A = [0 1 2; 3 5 0] A = 0 1 2 3 5 0 all(A) ans = 0 1 0
any	Возвращает единицу, если любой из аргументов является истинным или ненулевым; в противном случае возвращает 0. Как и all , any работает вдоль столбцов матриц.	v = [5 0 8]; any(v) ans = 1

Ряд других функций MATLAB-а выполняет логические операции. Например, функция **isnan** возвращает 1 для **NaN**; функция **isinf** возвращает 1 для **Inf**. Более подробный список можно найти в директории **ops**.

Логические выражения использующие функцию **find**

Функция **find** определяет индексы числового массива, удовлетворяющие заданному логическому условию. Эта функция удобна для создания логических масок (шаблонов) и матриц индексов. В наиболее общей форме, функция **find** возвращает единственный вектор индексов. Этот вектор может быть использован для индексации массивов любого размера или формы. Например, в приведенном ниже примере функция **find** позволяет легко заменить все элементы матрицы **A** больше 8 на число 100:

```

A = magic(4)

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

i = find (A > 8);
A(i) = 100

A =
    100     2     3    100
     5    100    100     8
    100     7     6    100
     4    100    100     1

```

Вы можете также использовать функцию **find** для получения обеих индексов строк и столбцов прямоугольных матриц, удовлетворяющих заданному логическому условию (более подробно эта функция описана в справочниках).

Приоритеты операторов

Вы можете строить выражения, использующие любую комбинацию арифметических и логических операторов, а также операторов отношения. Уровни приоритетов этих операторов определяют порядок, в котором MATLAB обрабатывает выражение. В пределах каждого уровня приоритета, операторы имеют одинаковый приоритет и оцениваются (обрабатываются) слева направо. Последовательность приоритетов для операторов MATLAB-а приведены ниже, упорядоченные в порядке убывания приоритетов, то есть от высшего приоритета к низшему.

1. Обычные скобки ().
2. Транспонирование (.'), степень (.^), комплексно-сопряженное транспонирование('), степень матрицы(^).
3. Унарный плюс (+), унарный минус (-), логическое отрицание (~).
4. Умножение (.*), правое деление (./), левое деление(.\), матричное умножение (*), матричное правое деление (/), матричное левое деление (\).
5. Сложение (+), вычитание (-).
6. Оператор двоеточия (:).
7. Меньше чем (<), меньше чем или равно (<=), больше чем (>), больше чем или равно (>=), равно (==), не равно (~=).
8. Логическое И (&).
9. Логическое ИЛИ (|).

Изменение приоритетов операторов

Имеющаяся последовательность приоритетов может быть изменена путем использования обычных скобок, как это показано в следующем примере.

```
A = [3  9  5];  
B = [2  1  5];  
  
C = A ./ B.^2  
C =  
0.7500  9.0000  0.2000  
C = (A ./ B) .^2  
C =  
2.2500  81.0000  1.0000
```

Выражения могут также содержать переменные, заданные посредством индексов

```
b = sqrt (A(2)) + 2*B(1)
```

```
b =  
7
```

Команды управления данными (Flow Control)

В MATLAB-е имеются 8 базовых команд для управления потоками данных:

- **if**, совместно с **else** и **elseif**, осуществляет обработку группы выражений, основываясь на некотором логическом условии.
- **switch**, совместно с **case** и **otherwise**, обрабатывает различные группы выражений, основываясь на значении некоторого логического условия.
- **while** осуществляет обработки группы выражений неопределенное число раз, основываясь на некотором логическом условии.
- **for** осуществляет обработку группы выражений определенное (заданное) число раз.
- **continue** передает управление к следующей итерации в циклах **for** или **while**, пропуская все оставшиеся выражения в теле цикла.
- **break** прекращает обработку выражений и выходит из циклов, созданных командами **for** или **while**.
- **try...catch** изменяет последовательность выполнения команд, если во время выполнения программы получено сообщение об ошибке.
- **return** приводит к прекращению выполнения данной программы и к возврату в вызывающую функцию.

Все конструкции программ, основанные на логических условиях, используют команду **end** для указания конца соответствующего блока.

Внимание! Во многих случаях вы можете ускорить выполнение программ MATLAB-а, путем замены циклов с командами **for** и **while** векторными выражениями (см. ниже).

Команды **if**, **else**, and **elseif**

Команда **if** оценивает логическое выражение и обрабатывает группу операторов, основываясь на значении указанного выражения. В своей простейшей форме синтаксис команды имеет вид

```
if (логическое выражение)logical_expression  
операторы  
end
```

Если логическое выражение истинно, то есть равно 1, MATLAB выполняет все операторы между строками, содержащими команды **if** и **end**. После этого он продолжает выполнять команды, находящиеся за строкой с **end**. Если логическое выражение ложно, то есть дает логический 0, MATLAB перескакивает через все выражения между строками с **if** и **end**, и продолжает свою работу со строки, следующей за командой **end**. Например,

```
if rem (a,2) == 0  
    disp('a is even')  
    b = a/2;  
end
```

Данный блок проверяет, является ли входной аргумент четным числом и, если да, то выводит в командную строку соответствующее сообщение и делит число **a** пополам. В противном случае, данный блок не выполняется. Между строками с **if** и **end** вы можете включить произвольное число операторов, содержащих, в свою очередь, любые команды и циклы. Если логическое выражение приводит к не скалярной величине, то для выполнения блока все элементы аргумента должны быть ненулевыми. Например, допустим **X** является матрицей. Тогда выражение

```

if X
    операторы
end

```

эквивалентно следующему

```

if all(X(:))
    операторы
end

```

При использовании с **if**, команды **else** и **elseif** дают следующие дополнительные возможности создания программ:

- Команда **else** не имеет логического условия. Операторы, связанные с данной командой выполняются, если предшествующее условие команды **if** (и, возможно, **elseif**) является ложным.
- Команда **elseif** имеет логическое условие, которое оценивается, если предшествующее условие команды **if** (и, возможно, **elseif**), является ложным. Если логическое условие данной команды **elseif** является истинным, то выполняются соответствующие операторы, следующие за данной командой. Вы можете иметь произвольное число команд **elseif** в пределах одного блока с **if**.

```

if n < 0      % Если n отрицательно, дать сообщение об ошибке
    disp('Input must be positive');
elseif rem(n, 2) == 0 % Если n положительно и четно, разделить на 2.
    A = n/2;
else
    A = (n+1)/2; % Если n положительно и нечетно, прибавить 1 и
                % разделить на два.
end

```

Команда **if** и пустые массивы

Если логическое условие, связанное с **if**, приводит к нулевому массиву, то оно оценивается как ложное. Например, если **A** является пустым массивом, то следующий блок

```

if A
    S1
else
    S0
end

```

выполняет оператор **S0**.

Команда **switch**

Команда **switch** осуществляет обработку определенных операторов, исходя из значения переменной или выражения. Ее базовая форма имеет вид

```

switch выражение (скаляр или строка символов)
case значение 1
    операторы % Выполняются если выражение == значение 1
case значение 2
    операторы % Выполняются если выражение == значение 2
.

```

```

.
.
otherwise
    операторы % Выполняются если выражение не соответствует не одному
               % значению, связанному с командами case
end

```

Данный блок состоит из:

- Слова **switch** за которым следует выражение которое нужно оценить..
- Любого числа блоков с командами **case**. Эти блоки состоят из слова **case**, за которым на той же строке следует возможное значение выражения за словом **switch**. Последующие строки содержат операторы, которые необходимо выполнить при указанном значении выражения в первой строке за словом **switch**. Эти строки могут быть любыми допустимыми выражениями, включая другие циклы **switch**. Выполнение группы операторов, связанных с данной командой **case** прекращается, когда MATLAB встречает следующую команду **case** или слово **otherwise**. Отметим, что всегда выполняется только *первый* подходящий блок с **case**.
- Не обязательной группы операторов, начинающихся словом **otherwise**; эта группа обрабатывается, если значение *выражения* не было перехвачено каким-либо предшествующим блоком с **case**. Обработка группы операторов за словом **otherwise** прекращается на команде **end**.
- Заключительной команды **end**.

Блок с командой **switch** работает путем сравнения входного выражения, которое может быть численным скаляром или строкой символов, с каждым значением ключей **case**. В случае численных выражений выполняется какой-либо (первый) блок, если справедливо логическое равенство *значение == выражение*. При выражениях в виде строки символов, блок выполняется, если истинно выражение **strcmp(значение,выражение)** (команда **strcmp** осуществляет логическое сравнение строк символов) .

Приведенный ниже код дает простой пример использования команды **switch**. Он проверяет переменную **input_num** и сравнивает ее с заданными числами. Если значения **input_num** равны -1, 0, или 1, команды **case** производят вывод значений на экран в виде текста. Если переменная **input_num** не равна не одному из указанных значений, выполнение переходит к строке **otherwise** и программа выводит на экран текст '**other value**' («*другое значение*»).

```

switch input_num
case -1
    disp('negative one');
case 0
    disp('zero');
case 1
    disp('positive one');
otherwise
    disp('other value');
end

```

Внимание ! В отличие от соответствующих операторов языка C, оператор **switch** в MATLAB-е «доходит» только до первого оператора **case**, который удовлетворяет заданному входному выражению; остальные операторы **case** при этом не проверяются.

Команда **switch** позволяет оперировать с несколькими условиями при одном операторе **case**, путем заключения данных условий в фигурные скобки, то есть при их записи в виде массива ячеек. Соответствующий пример приводится ниже

```

switch var
case 1
    disp('1')
case {2,3,4}
    disp('2 or 3 or 4')
case 5
    disp('5')
otherwise
    disp('something else')
end

```

Команда **while**

Циклы с командой **while** обрабатывают оператор или группу операторов, находящихся в теле цикла, до тех пор, пока истинно проверяемой логическое условие при данной команде. Иными словами, операторы внутри цикла могут выполняться любое неопределенное заранее число раз. Ее синтаксис в общем случае имеет вид

```

while expression
    statements
end

```

Если логическое выражение *expression* имеет матричный вид, то для продолжения выполнения цикла все его элементы должны быть истинными, то есть равны логической единице. Чтобы привести матричное условие к скалярному, следует воспользоваться функциями **all** и **any**. Например, следующий цикл находит первое целое число **n**, для которого факториал **n!** является 100-значным числом

```

n = 1;
while prod(1:n) < 1e100
    n = n + 1;
end

```

Для выхода из петли **while** в любой момент нужно применить в теле цикла команду **break**. При этом, естественно, следует сформировать соответствующее логическое условие выхода из цикла.

Команда **while** и пустые массивы

Если условие при **while** сводится к пустому массиву, то оно соответствует ложному выражению, то есть последовательность команд

```

while A
    S1;
end

```

никогда не выполнит оператор **S1** если **A** есть пустой массив.

Команда **for**

Цикл с командой **for** обрабатывает оператор или группу операторов заранее заданное число раз. Ее синтаксис имеет вид

```
for index = start:increment:end
    statements
end
```

где *index* – является изменяемым целочисленным индексом с начальным и конечным значениями *start* и *end* и приращением *increment*. По умолчанию, приращение равно 1, но вы можете задать любое приращение, включая и отрицательное. При положительных индексах, выполнение прекращается когда значение индекса превышает конечное значение *end*; при отрицательных приращениях выполнение прекращается когда индекс становится меньше конечного значения.

Например, следующий цикл выполняется пять раз.

```
for i = 2:6
    x(i) = 2*x (i-1);
end
```

Вы можете использовать вложенные циклы с командой **for**:

```
for i = 1:m
    for j = 1:n
        A (i, j) = 1/(i + j - 1) ;
    end
end
```

Внимание ! Вы можете зачастую ускорить выполнение кодов в MATLAB-е путем замены циклов с **for** и **while** векторизованными кодами (см. ниже).

Использование в качестве индексов массивов

Индексы команды **for** могут быть массивом значений. Например, рассмотрим массив *A* размера *m* x *n* . Выражение

```
for i = A
    statements
end
```

приравнивает индекс *i* вектору *A*(: , *k*). При первой итерации значение *k* равно 1; при второй итерации *k* равно 2 , и так до тех пор, пока *k* не становится равным *n*. Иными словами, цикл повторяется *n* раз, где *n* есть число столбцов в *A*. При этом, при каждой итерации индекс *i* является вектором, содержащим один из столбцов матрицы *A*.

Команда **continue**

Команда **continue** передает управление следующей итерации в циклах **for** или **while**, что приводит к перескакиванию через все оставшиеся операторы в теле цикла. В случае вложенных циклов, команда **continue** передает управление к следующей итерации внешних по отношению к данному циклу команд **for** или **while** , то есть пропускаются только оставшиеся операторы данного внутреннего цикла. Для применения данной команды нужно сформировать соответствующее логическое условие.

Команда **break**

Команда **break** прекращает выполнение циклов, образованных командами **for** или **while**. Когда в теле цикла встречается данная команда, дальнейшее выполнение происходит начиная со следующей строки вне данного цикла. Во вложенных циклах, команда **break** приводит к выходу только из внутреннего цикла.

Команды **try ... catch**

Общая форма последовательности команд **try ... catch** имеет вид

```
try
    statement,
    ...,
    statement,
catch,
    statement,
    ...,
    statement,
end
```

В данной последовательности выполняются операторы между командами **try** и **catch** до тех пор, пока не произойдет какая-либо ошибка вычислений. Тогда управление передается («перехватывается») последовательностью операторов между командами **catch** и **end**. Для того чтобы узнать причину, приведшую к ошибке, можно воспользоваться командой **lasterr** (**последняя ошибка**). Если ошибка случается между командами **catch** и **end**, то MATLAB прекращает вычисления, если только между предыдущими командами **catch** и **end** не установлена другая последовательность команд **try ... catch**.

Команда **return**

Команда **return** прекращает выполнение текущей последовательности команд и возвращает управление в вызывающую функцию (то есть «родительскую» функцию, из которой была вызвана текущая функция) или же возвращает управление к клавиатуре. Команда **return** также приводит к окончанию режима **keyboard** (последняя команда, когда встречается в теле программы, передает управление клавиатуре; она часто используется при отладке программы). При обычных условиях работы вызванная функция передает управление в вызывающую функцию когда первая выполнена полностью, то есть до конца. Команда **return** может быть включена в тело вызываемой функции для того чтобы прекратить ее исполнение при выполнении определенных условий.

Подфункции

М-функции могут содержать коды более чем одной функции. Первая функция в файле является *главной функцией* (*primary function*), которая имеет имя самого М-файла. Дополнительные функции в пределах данного файла являются *подфункциями* (*subfunctions*), которые «видны» (то есть могут быть вызваны) только для главной функции или других подфункций того же файла. Каждая подфункция начинается со своей линии определения. Все подфункции следуют непосредственно друг за другом. Порядок следования различных подфункций не имеет никакого значения, при этом главная функция должна быть обязательно записана первой. Например,

```

function [avg,med] = newstats(u) % Главная функция
% NEWSTATS Находит среднее значение и медиану при помощи внутренних
% подфункций.
n = length(u);
avg = mean(u,n);
med = median(u,n);
function a = mean(v,n) % Подфункция
% Находит среднее значение.
a = sum(v)/n;
function m = median(v,n) % Подфункция
% Находит медиану.
w = sort(v);
if rem(n,2) == 1
    m = w((n+1)/2);
else
    m = (w(n/2)+w(n/2+1))/2;
end

```

Подфункции **mean** и **median** вычисляют среднее значение и медиану входных данных. Главная функция **newstats** находит длину входного вектора данных и вызывает подфункции, передавая им данные длины **n**. Функции в пределах одного и того же файла не имеют общего доступа к каким-либо переменным, если только вы не объявите эти переменные *глобальными* или же не передадите их в виде аргументов в соответствующие подфункции, как это реализовано в приведенном выше примере. Кроме того, все средства помощи («help») имеют доступ только к главной функции в М-файле. Когда вы вызываете какую-либо функцию из М-файла, MATLAB прежде всего проверяет, не является ли данная функция подфункцией. Затем проверяется, не является ли данная функция *частной функцией* (*private function*), описанной в следующем разделе, и далее ищутся стандартные М-файлы на путях доступа MATLAB. Поскольку подфункции проверяются первыми, вы можете «подавить» вызов любого существующего М-файла путем применения подфункций с тем же именем, как, например, мы имели в примере с функцией **mean**. Вместе с тем, в пределах одного М-файла все подфункции должны иметь разные индивидуальные имена.

Частные функции

Частные функции являются функциями, которые находятся в подкаталоге со специальным именем **private** (частный). Они видны, то есть могут быть вызваны, только из функций в пределах родительского каталога. Например, допустим, что каталог **newmath** находится на путях доступа MATLAB-а. Подкаталог в данной директории **newmath**, названный **private**, содержит только функции, которые могут быть вызваны из каталога **newmath**. Поскольку частные функции являются невидимыми вне пределов родительского каталога, они могут использовать имена, совпадающие с именами функций в других каталогах. Это свойство бывает полезным, если вы хотите создать свою версию какой-либо конкретной функции, сохраняя в то же время оригинальную функцию в другом каталоге. Так как MATLAB ищет среди частных функций прежде чем перейти к поиску в остальных каталогах, он всегда найдет частную функцию, например с именем **test.m**, раньше чем любую другую М-функцию, названную тем же именем **test.m**, но расположенную в других каталогах. Вы можете создавать свои частные директории просто путем создания стандартных подкаталогов, названных **private**. При этом не следует помещать директории **private** на ваших путях доступа.

Приложение 1. Тематические направления системы MATLAB

1. **matlab\general** - Команды общего назначения
2. **matlab\ops** - Операторы и специальные символы
3. **matlab\lang** - Конструкции языка и функции отладки
4. **matlab\elmat** - Элементарные матрицы и операции на ними
5. **matlab\elfun** - Элементарные математические функции
6. **matlab\specfun** - Специальные математические функции
7. **matlab\matfun** - Матричные функции и линейная алгебра
8. **matlab\datafun** - Анализ данных и преобразование Фурье
9. **matlab\audio** - Поддержка звуковых сигналов
10. **matlab\polyfun** - Полиномы и интерполяция
11. **matlab\funfun** - Нелинейные численные методы и решатели обыкновенных дифференциальных уравнений (**ODE solvers**)
12. **matlab\sparfun** - Разреженные матрицы
13. **matlab\graph2d** - Двумерная графика
14. **matlab\graph3d** - Трехмерная графика
15. **matlab\specgraph** - Специальная графика
16. **matlab\graphics** - Дескрипторная графика (**Handle Graphics**)
17. **matlab\uitools** - Инструменты графического интерфейса пользователя (**Graphical User Interface (GUI)**)
18. **matlab\strfun** - Функции обработки символьных строк
19. **matlab\iofun** - Функции ввода/вывода
20. **matlab\timefun** - Функции времени и даты
21. **matlab\datatypes** - Типы данных и структуры
22. **matlab\verctrl** - Выбор версии
23. **matlab\winfun** - Интерфейс с операционной системой Windows (**DDE/ActiveX**)
24. **matlab\demos** - Демонстрации и примеры возможностей системы **MATLAB**
25. **toolbox\local** - Выбор характеристик (**Preferences**)
26. **simulink\simulink** - Система моделирования **Simulink**
27. **simulink\blocks** - Библиотека блоков (моделей) **Simulink**
28. **simulink\simdemos** - Демонстрация и примеры системы **Simulink 4**
29. **simdemos\ aerospace** - **Simulink**: Демонстрация и примеры космических моделей
30. **simdemos\automotive** - **Simulink**: Демонстрация моделей САУ
31. **simdemos\simfeatures** - **Simulink**: Демонстрация основных свойств
32. **simdemos\simgeneral** - **Simulink**: Демонстрация примеров общих моделей
33. **simdemos\simnew** - **Simulink**: Демонстрация новых свойств
34. **simulink\dee** - Редактор дифференциальных уравнений
35. **stateflow\stateflow** - Система моделирования событий **Stateflow**
36. **stateflow\sfdemos** - Демонстрации возможностей и примеры **Stateflow**
37. **stateflow\coder** - Кодирование устройств системы **Stateflow**
38. **rtw\rtw** - Моделирование систем реального времени **Real-Time Workshop**
39. **rtw\rtwdemos** - Демонстрация возможностей системы **Real-Time Workshop**
40. **cdma\cdma** - Библиотека моделей системы **Simulink** для систем коммуникации (**CDMA Reference Blockset**)
41. **cdma\cdmamasks** - Функции поддержки моделей **CDMA**
42. **cdma\cdmamex** - **S-функции (S-Functions)** системы **CDMA**
43. **cdma\cdmademos** - Демонстрации и примеры системы **CDMA**
44. **commblocks\commblocks** - Библиотека моделей устройств связи и коммуникации (**Communications Blockset**)

45. **commblks\commmasks** - Функции поддержки моделей **Communications Blockset**
46. **commblks\commmex** - S-функции библиотеки **Communications Blockset**
47. **commblks\commblksdemos** - Демонстрации библиотеки **Communications Blockset**
48. **comm\comm** - Системы связи и коммуникации (**Communications Toolbox**)
49. **comm\commdemos** - Демонстрация возможностей пакета **Communications Toolbox**
50. **control\control** - Системы управления (**Control System Toolbox**)
51. **control\ctrl demos** - Демонстрации возможностей **Control System Toolbox**
52. **dspblks\dspblks** - Библиотека для цифровой обработки сигналов (**DSP Blockset**)
53. **dspblks\dspmasks** - Функции поддержки моделей **DSP Blockset**
54. **dspblks\dspmex** - S-функции и MEX-файлы библиотеки **DSP Blockset**
55. **dspblks\dspdemos** - Демонстрации и примеры библиотеки **DSP Blockset**
56. **daq\daq** - Сбор данных в режиме реального времени (**Data Acquisition Toolbox**)
57. **daq\daqdemos** - Демонстрации и примеры пакета **Data Acquisition Toolbox**
58. **database\database** - Работа с базами данных (**Database Toolbox**)
59. **database\dbdemos** - Демонстрации возможностей пакета **Database Toolbox**
60. **datafeed\datafeed** - Пакет сбора финансовой информации (**Datafeed Toolbox**)
61. **toolbox\dials** - Интерфейс поддержки моделей **Simulink (Dials & Gauges Blockset)**
62. **filterdesign\filterdesign** - Проектирование фильтров (**Filter Design Toolbox**)
63. **filterdesign\filt desdemos** – Демонстрации пакета **Filter Design Toolbox**
64. **finderiv\finderiv** - Расширение возможностей пакета **Financial Toolbox (Financial Derivatives Toolbox)**
65. **ftseries\ftseries** - Временной анализ финансового рынка (**Financial Time Series Toolbox**)
66. **finance\finance** - Финансы (**Financial Toolbox**)
67. **finance\calendar** - Функции времени и даты пакета **Financial Toolbox**
68. **finance\findemos** - Демонстрации возможностей пакета **Financial Toolbox**
69. **toolbox\fixpoint** - Библиотека для моделирования с конечной разрядностью (**Fixed-Point Blockset**)
70. **fixpoint\fxpdemos** - Демонстрации библиотеки **Fixed-Point Blockset**
71. **fuzzy\fuzzy** - Размытые множества (**Fuzzy Logic Toolbox**)
72. **fuzzy\fuzdemos** - Демонстрации возможностей пакета **Fuzzy Logic Toolbox**
73. **images\images** - Обработка изображений (**Image Processing Toolbox**)
74. **images\indemos** - Демонстрации и примеры пакета **Image Processing Toolbox**
75. **instrument\instrument** - Связь (поддержка интерфейса с аппаратурой пользователя) (**Instrument Control Toolbox**)
76. **lmi\lmictrl** Приложения пакета **LMI Control Toolbox** в управлении
77. **lmi\lmilab** - Синтез систем управления на основе линейных матричных неравенств (**LMI Control Toolbox**)
78. **toolbox\compiler** - Компилятор системы **MATLAB (MATLAB Compiler)**
79. **map\map** - Картография (**Mapping Toolbox**)
80. **mpc\mpccmds** - Системы управления с эталонной моделью (**Model Predictive Control Toolbox**)
81. **mpc\mpcdemos** - Демонстрации пакета **Model Predictive Control Toolbox**
82. **mutools\commands** - μ -анализ и синтез систем управления (**Mu-Analysis and Synthesis Toolbox**)
83. **nnet\nnet** - Нейронные сети (**Neural Network Toolbox**)
84. **nnet\nndemos** - Демонстрации возможностей пакета **Neural Network**
85. **toolbox\ncd** - Проектирование нелинейных систем управления (**Nonlinear Control Design Blockset**)
86. **toolbox\optim** - Оптимизация (**Optimization Toolbox**)
87. **toolbox\pde** - Дифференциальные уравнения в частных производных и метод

- конечных элементов (**Partial Differential Equation Toolbox**)
- 88. **powersys\powersys** - Моделирование энергетических систем (**Power System Blockset**)
 - 89. **powersys\powerdemo** - Демонстрации моделей **Power System Blockset**
 - 90. **targets\ecoder** - Создание пользовательских программ на основе моделей **Simulink (Real-Time Workshop Embedded Coder)**
 - 91. **toolbox\robust** - Синтез робастных систем управления (**Robust Control Toolbox**)
 - 92. **signal\signal** - Обработка сигналов (**Signal Processing Toolbox**)
 - 93. **signal\sigdemos** - Демонстрации возможностей пакета **Signal Processing Toolbox**
 - 94. **toolbox\splines** - Сплайн-аппроксимация (**Spline Toolbox**)
 - 95. **toolbox\stats** - Статистика (**Statistics Toolbox**)
 - 96. **toolbox\symbolic** - Символьная математика (**Symbolic Math Toolbox**)
 - 97. **ident\ident** - Идентификация параметров систем управления (**System Identification Toolbox**)
 - 98. **wavelet\wavelet** - Импульсная декомпозиция сигналов и изображений (**Wavelet Toolbox**)
 - 99. **wavelet\wavedemo** - Демонстрация возможностей пакета **Wavelet Toolbox**
 - 100. **xpc\xpc** - Пакет макетирования и контроля систем реального времени (**xPC Target**)
 - 101. **xpc\xpcdemos** - Демонстрации возможностей пакета **xPC Target**

Примечание. Имя каждого тематического направления определяет соответствующую директорию (каталог) системы MATLAB.

Приложение 2. Команды общего назначения (General purpose commands)

Директория - **matlab\general**

Общая информация (General information)

- 1. **help** - Оперативная справка, выводит текст в командную строку.
- 2. **helpwin** - Оперативная справка, выводит информацию в специальном окне (**help browser**).
- 3. **helpdesk** - Исчерпывающая информация в рабочем окне помощи.
- 4. **support** - Открывает Web-страницу технической помощи фирмы MathWorks.
- 5. **demo** - Выполняет демонстрационные ролики.
- 6. **java** - Использование средств Java из MATLAB-a.
- 7. **ver** - Информация о версиях MATLAB-a, SIMULINK-a и прикладных пакетов.
- 8. **whatsnew** - Вызов информации о новых свойствах версий.

Управление рабочим пространством (Managing the workspace)

- 9. **who** - Выводит в командное окно список текущих переменных.
- 10. **whos** - Выводит в командное окно список текущих переменных в длинном формате, с указанием размеров, числа байтов и классов.
- 11. **workspace** - Вызывает Окно Просмотра Рабочего Пространства (**Workspace**)

- Browser), т.е. специальный Графический Интерфейс Пользователя (GUI) для выполнения действий с переменными рабочего пространства.
- 12. **clear** - Удаляет переменные и функции из памяти.
 - 13. **pack** - Дефрагментация рабочей области памяти.
 - 14. **load** - Загрузка переменных в рабочее пространство из диска.
 - 15. **save** - Сохранение переменных рабочего пространства на диск.
 - 16. **quit** - Прекращение сеанса работы системы MATLAB.

Управляющие команды и функции (Managing commands and functions)

- 17. **what** - Выводит список файлов MATLAB-а в текущей директории в командное окно.
- 18. **type** - Печатает содержимое заданного М.-файла в командном окне.
- 19. **edit** - Вызывает окно Редактора/Отладчика.
- 20. **open** - Открывает файлы по расширению.
- 21. **which** - Локализирует (выводит в командное окно) путь доступа к функциям и файлам.
- 22. **pcode** - Создает Р-файл псевдокода с выполнением грамматического анализа.
- 23. **inmem** - Выводит список функций в памяти.
- 24. **mex** - Компилирует MEX-файлы.

Управление путями доступа (Managing the search path)

- 25. **path** - Вывод/изменение путей доступа.
- 26. **addpath** - Прибавляет директорию к путям доступа.
- 27. **rmpath** - Удаляет директорию из путей доступа .
- 28. **pathtool** - Модифицирует пути доступа.
- 29. **rehash** - Управление кеш-памятью.
- 30. **import** - Импортирование пакетов Java в текущую область.

Управление командным окном (Controlling the command window)

- 31. **echo** - Вывод в командное окно исполняемых в М.-файлах команд.
- 32. **more** - Управление постраничным выводом информации на экран.
- 33. **diary** - Сохранение записи (дневника) сеанса работы системы MATLAB .
- 34. **format** - Контроль формата вывода данных на экран.
- 35. **beep** - Производит звуковой сигнал (beep).

Команды операционной системы (Operating system commands)

- 36. **cd** - Изменить текущей директории.
- 37. **copyfile** - Копировать файла.
- 38. **pwd** - Показать (напечатать) текущую рабочую директорию.
- 39. **dir** - Вывод на экран листинга каталога.
- 40. **delete** - Удалить файл.
- 41. **getenv** - Получение значения переменной из внешней операционной среды.

- 42. **mkdir** - Создать директорию.
- 43. **!** - Выполнить команды операционной системы.
- 44. **dos** - Выполнить команду DOS и вернуть результат.
- 45. **unix** - Выполнить команду UNIX и вернуть результат.
- 46. **vms** - Выполнить команду VMS DCL и вернуть результат.
- 47. **web** - Вызвать Web browser.
- 48. **computer** - Выдать тип компьютера.
- 49. **isunix** - Истинно, если установлена UNIX-версия MATLAB-а.
- 50. **ispc** - Истинно, если установлена PC (Windows)-версия MATLAB-а.

Отладка М-файлов (Debugging M-files)

- 51. **debug** - Вывести список команд отладки и редактирования файлов.
- 52. **dbstop** - Установить точку останова (контрольную точку).
- 53. **dbclear** - Удалить контрольную точку.
- 54. **dbcont** - Продолжить выполнение.
- 55. **dbdown** - Переход между рабочими пространствами редактируемых функций сверху вниз.
- 56. **dbstack** - Вывести в командное окно стек вызываемых функций.
- 57. **dbstatus** - Вывести список всех контрольных точек.
- 58. **dbstep** - Выполнить одну или более строк.
- 59. **dbtype** - Напечатать в командном окне редактируемый файл с указанием номеров строк.
- 60. **dbup** - Переход между рабочими пространствами редактируемых функций снизу вверх.
- 61. **dbquit** - Выход из режима отладки.
- 62. **dbmex** - Отладка MEX-файлов (только для системы UNIX).

Профилировщик М-файлов (Profiling M-files)

- 63. **profile** - Измерить и вывести на экран временные затраты при выполнении файла
- 64. **profreport** - Сформировать отчет о профилировании файла.

Функции для определения местонахождения зависимых функций М-файла (Tools to locate dependent functions of an M-file).

- 65. **depfun** - Определить местонахождение функций, от которых зависит М-файл.
- 66. **depdir** - Определить местонахождение директории, где расположены функции, от которых зависит М-файл.
- 67. **inmem** - Вывести в командное окно список функций в памяти.

Приложение 3. Операторы и специальные символы

Директория - `matlab\general`

Арифметические операторы (Arithmetic operators)

1. plus	- Plus	+	- Сложение.
2. uplus	- Unary plus	+	- Унарное сложение.
3. minus	- Minus	-	- Вычитание.
4. uminus	- Unary minus	-	- Унарное вычитание.
5. mtimes	- Matrix multiply	*	- Умножение матриц.
6. times	- Array multiply	.*	- Умножение массивов.
7. mpower	- Matrix power	^	- Возведение в степень матриц .
8. power	- Array power	.^	- Возведение в степень массивов.
9. ldivide	- Left matrix divide	\	- Левое деление матриц.
10. mrdivide	- Right matrix divide	/	- Правое деление матриц.
11. ldivide	- Left array divide	.\	- Левое деление массивов.
12. rdivide	- Right array divide	./	- Правое деление массивов.
13. kron	- Kronecker tensor product		- Кронекеровское произведение.

Операторы отношения (Relational operators)

14. eq	- Equal	==
15. ne	- Not equal	~=
16. lt	- Less than	<
17. gt	- Greater than	>
18. le	- Less than or equal	<=
19. ge	- Greater than or equal	>=

Логические операторы (Logical operators)

20. and	- Logical AND	&	- Логическое «Да».
21. or	- Logical OR		- Логическое «Или».
22. not	- Logical NOT	~	- Логическое «Нет».
23. xor	- Logical EXCLUSIVE OR		- Логическое исключающее «Или».
24. any	- Истинно, если хоть один из элементов вектора не равен нулю.		
25. all	- Истинно, если все элементы вектора не равны нулю.		

Специальные символы (Special characters.)

26. colon	- :	- Двоеточие.
27. paren	- ()	- Круглые скобки и индексация.
28. paren	- []	- Квадратные скобки.
29. paren	- { }	- Фигурные скобки и индексация.
30. punct	- @	- Создание дескриптора функций.
31. punct	- .	- Десятичная точка.
32. punct	- .	- Доступ к полю структуры.
33. punct	- ..	- Родительская директория.

34. punct	-	...	- Продолжение.
35. punct	-	,	- Разделитель.
36. punct	-	;	- Точка с запятой.
37. punct	-	%	- Комментарий.
38. punct	-	!	- Вызов команды операционной системы.
39. punct	-	=	- Присвоение значений.
40. punct	-	'	- Кавычка.
41. transpose	-	.'	- Поэлементное транспонирование.
42. ctranspose	-	'	- Транспонирование и комплексное сопряжение.
43. horzcat	-	[,]	- Горизонтальное объединение объектов.
44. vertcat	-	[;]	- Вертикальное объединение объектов.
45. subsasgn	-	(), { }, .	- Индексное присвоение.
46. subsref	-	(), { }, .	- Индексная ссылка.
47. subsindex	-		- Индексный дескриптор

Операторы побитовой обработки (Bitwise operators)

48. bitand	- Побитовое логическое умножение (Bit-wise AND).
49. bitcmp	- Побитовое n-разрядное дополнение.
50. bitor	- Побитовое логическое «Или» (Bit-wise OR).
51. bitmax	- Максимальное целое число.
52. bitxor	- Побитовое логическое исключающее «Или» (Bit-wise XOR).
53. bitset	- Установить значение бита.
54. bitget	- Получить значение бита.
55. bitshift	- Сложение битов по модулю 2.

Операторы обработки множеств (Set operators)

56. union	- Объединение множеств.
57. unique	- Удаление из множеств одинаковых элементов.
58. intersect	- Пересечение множеств.
59. setdiff	- Разность множеств.
60. setxor	- Операция исключающего «Или» над множествами.
61. ismember	- Выявление одинаковых элементов.

Приложение 4. Элементарные математические функции (Elementary math functions).

Директория **matlab\elfun**

Тригонометрические функции (Trigonometric)

1. sin	- Синус.
2. sinh	- Гиперболический синус.
3. asin	- Обратный синус.
4. asinh	- Обратный гиперболический синус.

- 5. **cos** - Косинус.
- 6. **cosh** - Гиперболический косинус.
- 7. **acos** - Обратный косинус.
- 8. **acosh** - Обратный гиперболический косинус.
- 9. **tan** - Тангенс.
- 10. **tanh** - Гиперболический тангенс.
- 11. **atan** - Обратный тангенс.
- 12. **atan2** - 4-х квадрантный обратный тангенс.
- 13. **atanh** - Обратный гиперболический тангенс.
- 14. **sec** - Секанс.
- 15. **sech** - Гиперболический секанс.
- 16. **asec** - Обратный секанс.
- 17. **asech** - Обратный гиперболический секанс.
- 18. **csc** - Косеканс.
- 19. **csch** - Гиперболический косеканс.
- 20. **acsc** - Обратный косеканс.
- 21. **acsch** - Обратный гиперболический косеканс.
- 22. **cot** - Котангенс.
- 23. **coth** - Гиперболический котангенс.
- 24. **acot** - Обратный котангенс.
- 25. **acoth** - Обратный гиперболический котангенс.

Экспоненциальные функции (Exponential)

- 26. **exp** - Экспоненциальная функция.
- 27. **log** - Натуральный логарифм.
- 28. **log10** - Логарифм по основанию 10.
- 29. **log2** - Логарифм по основанию 2 .
- 30. **pow2** - Экспонента по основанию 2.
- 31. **sqrt** - Квадратный корень.
- 32. **nextpow2** - Ближайшая степень по основанию 2.

Комплексные числа (Complex)

- 33. **abs** - Абсолютное значение числа.
- 34. **angle** - Аргумент комплексного числа.
- 35. **complex** - Конструирование комплексных данных из действительных и мнимых частей.
- 36. **conj** - Комплексное сопряжение.
- 37. **imag** - Мнимая часть комплексного числа.
- 38. **real** - Действительная часть комплексного числа.
- 39. **unwrap** - Корректировка фазового угла.
- 40. **isreal** - Истинно, если массив содержит реальные числа.
- 41. **cplxpair** - Сортировка чисел в комплексно-сопряженные пары.

Округление и остатки (Rounding and remainder)

- 42. **fix** - Округление в сторону нуля.

- 43. **floor** - Округление в сторону минус бесконечность.
- 44. **ceil** - Округление в сторону плюс бесконечность
- 45. **round** - Округление в сторону ближайшего целого числа.
- 46. **mod** - Остаток со знаком после деления (Modulus or signed remainder after division).
- 47. **rem** - Остаток после деления.
- 48. **sign** - Функция знака (Signum).

Приложение 5. Элементарные матрицы и операции над ними **(Elementary matrices and matrix manipulation)**

Директория **matlab\elmat**

Элементарные матрицы и векторы (Elementary matrices)

- 1. **zeros** - Формирование массива нулей (матрицы из нулей).
- 2. **ones** - Формирование массива единиц.
- 3. **eye** - Единичная матрица.
- 4. **repmat** - Формирование многомерного массива из блоков.
- 5. **rand** - Равномерно распределенные случайные числа.
- 6. **randn** - Нормально распределенные случайные числа.
- 7. **linspace** - Формирование массива равноотстоящих чисел.
- 8. **logspace** - Формирование узлов логарифмической сетки.
- 9. **freqspace** - Формирование массива частот для частотных откликов.
- 10. **meshgrid** - Формирование узлов двумерной и трехмерной сеток (для построения трехмерных графиков).
- 11. **:** - Формирование векторов с равноотстоящими значениями и индексирование матриц.

Основная информация о массивах (Basic array information)

- 12. **size** - Размер массива (матрицы).
- 13. **length** - Длина вектора.
- 14. **ndims** - Число размерностей массива.
- 15. **disp** - Вывод в командное окно матрицы или текста.
- 16. **isempty** - Истинно для пустых массивов.
- 17. **isequal** - Истинно для одинаковых массивов.
- 18. **isnumeric** - Истинно для числовых массивов.
- 19. **islogical** - Истинно для логических массивов.
- 20. **logical** - Преобразование числовых массивов в логические.

Преобразования матриц (Matrix manipulation)

- 21. **reshape** - Преобразование размеров многомерного массива.
- 22. **diag** - Диагональные матрицы и диагонали матриц.
- 23. **blkdiag** - Блочно-диагональное объединение матриц.

- 24. **tril** - Извлечение нижней треугольной части матрицы.
- 25. **triu** - Извлечение верхней треугольной части матрицы.
- 26. **fliplr** - Зеркальное отображение слева направо.
- 27. **flipud** - Зеркальное отображение сверху вниз.
- 28. **flipdim** - Зеркальное отображение вдоль заданной размерности (обобщенное транспонирование).
- 29. **rot90** - Поворот матриц на 90 градусов.
- 30. **find** - Определение индексов ненулевых элементов массива.
- 31. **end** - Последний индекс по указанной размерности.
- 32. **sub2ind** - Преобразование многомерной нумерации в последовательную.
- 33. **ind2sub** - Преобразование последовательной нумерации в многомерную.

Специальные символы, переменные и константы (Special variables and constants)

- 34. **ans** - Результат выполнения последней операции.
- 35. **eps** - Точность машинного представления чисел с плавающей запятой.
- 36. **realmax** - Наибольшее положительное число с плавающей запятой.
- 37. **realmin** - Наименьшее положительное число с плавающей запятой.
- 38. **pi** - 3.1415926535897....
- 39. **i, j** - Мнимая единица.
- 40. **inf** - Бесконечность.
- 41. **NaN** - Нечисловое значение.
- 42. **isnan** - Истинно для нечисловых значений.
- 43. **isinf** - Истинно для бесконечных значений.
- 44. **isfinite** - Истинно для конечных значений.
- 45. **why** - Выдает ответ в виде набора случайных фраз.

Специальные типы матриц (Specialized matrices)

- 46. **compan** - Сопутствующая матрица.
- 47. **gallery** - Набор тестовых матриц.
- 48. **hadamard** - Матрица Адамара.
- 49. **hankel** - Матрица Ганкеля.
- 50. **hilb** - Матрица Гильберта.
- 51. **invhilb** - Обратная матрица Гильберта.
- 52. **magic** - Матрица «Волшебный квадрат».
- 53. **pascal** - Матрица Паскаля.
- 54. **rosser** - Матрица Рессера (тестовая матрица для классической симметричной проблемы собственных значений).
- 55. **toeplitz** - Матрица Теплица (Toeplitz matrix).
- 56. **vander** - Матрица Вандермонда (Vandermonde matrix).
- 57. **wilkinson** - Матрица Уилкинсона (тестовая матрица Уилкинсона (Wilkinson) для задачи на собственные значения).

Приложение 6. - Матричные функции и линейная алгебра **(Matrix functions - numerical linear algebra)**

Директория **matlab\matfun**

Матричный анализ (Matrix analysis)

1. **norm** - Нормы векторов и матриц.
2. **normest** - Оценка 2-нормы матриц.
3. **rank** - Ранг матрицы.
4. **det** - Детерминант матрицы
5. **trace** - След матрицы (сумма диагональных элементов).
6. **null** - Нуль-пространство (ядро) матрицы.
7. **orth** - Ортонормальный базис матрицы.
8. **rref** - Треугольная форма матрицы (Reduced row echelon form).
9. **subspace** - Угол между двумя подпространствами.

Линейные уравнения (Linear equations)

10. **\ and /** - Решение линейных уравнений (см. Приложение 3. Арифметические операторы)
11. **inv** - Обратная матрица.
12. **rcond** - Обратная величина числа обусловленности матрицы ,найденная при помощи вычислителя пакета LAPACK (LAPACK reciprocal condition estimator).
13. **cond** - Число обусловленности по отношению к обращению матриц.
14. **condest** - Оценка числа обусловленности 1-нормы матрицы.
15. **normest1** - Оценка 1-нормы матрицы.
16. **chol** - Разложение Холецкого (Cholesky factorization).
17. **cholinc** - Неполное разложение Холецкого (Incomplete Cholesky factorization).
18. **lu** - LU-разложение (LU factorization).
19. **luinc** - Неполное LU-разложение (Incomplete LU factorization).
20. **qr** - Ортогонально-треугольная декомпозиция.
21. **lsqnonneg** - Метод наименьших квадратов с неотрицательными ограничениями.
22. **pinv** - Псевдообратная матрица.
23. **lscov** - Метод наименьших квадратов в присутствии шумов.

Собственные значения и сингулярные числа (Eigenvalues and singular values)

24. **eig** - Собственные значения и собственные векторы.
25. **svd** - Сингулярное разложение матрицы.
26. **gsvd** - Обобщенное сингулярное разложение матрицы.
27. **eigs** - Вычисление нескольких собственных значений (с наибольшими модулями).
28. **svds** - Вычисление нескольких сингулярных чисел.
29. **poly** - Характеристический полином матрицы.
30. **polyeig** - Вычисление собственных значений матричного полинома (Polynomial eigenvalue problem).
31. **condeig** - Число обусловленности относительно собственных значений матрицы.
32. **hess** - Приведение к форме Хессенберга (Hessenberg form).

- 33. **qz** - QZ-факторизация (приведение пары матриц к обобщенной форме Шура).
- 34. **schur** - Приведение к форме Шура (Schur decomposition).

Вычисление функций от матриц (Matrix functions).

- 35. **expm** - Вычисление матричной экспоненты.
- 36. **logm** - Вычисление логарифма матрицы.
- 37. **sqrtn** - Вычисление квадратного корня матрицы.
- 38. **funm** - Вычисление произвольной функции от матрицы.
- 39. **expm1** - Матричная экспонента с использованием разложения Паде.
- 40. **expm2** - Матричная экспонента с использованием разложения в ряд Тейлора.
- 41. **expm3** - Матричная экспонента с использованием собственных значений и собственных векторов.

Утилиты для процедур факторизации матриц (Factorization utilities)

- 42. **qrdelete** - Удалить столбец в QR -разложении.
- 43. **qrintert** - Вставить столбец в QR-разложение.
- 44. **rsf2csf** - Преобразование действительной блочно-диагональной формы к комплексной диагональной форме.
- 45. **cdf2rdf** - Преобразование комплексной блочно-диагональной формы к действительной диагональной форме.
- 46. **balance** - Масштабирование матрицы для повышения точности вычисления собственных значений.
- 47. **planerot** - Преобразование Гивенса (плоское вращение Гивенса).
- 48. **cholupdate** - Разложение Холецкого модифицированной матрицы.
- 49. **qrupdate** - QR –разложение модифицированной матрицы.

Приложение 7. Полиномы и интерполяция (Interpolation and polynomials)

Директория **matlab\polyfun**

Интерполяция данных (Data interpolation)

- 1. **pchip** - Интерполяция кусочным кубическим полиномом Эрмита.
- 2. **interp1** - Одномерная табличная интерполяция.
- 3. **interp1q** - Быстрая одномерная табличная интерполяция.
- 4. **interpft** - Одномерная интерполяция с использованием быстрого преобразования Фурье.
- 5. **interp2** - Двумерная табличная интерполяция.
- 6. **interp3** - Трехмерная табличная интерполяция.
- 7. **interp** - Многомерная табличная интерполяция.
- 8. **griddata** - Двумерная интерполяция на неравномерной сетке.
- 9. **griddata3** - Трехмерная интерполяция на неравномерной сетке.
- 10. **griddatan** - Многомерная интерполяция на неравномерной сетке.

Интерполяция сплайнами (Spline interpolation)

- 11. **spline** - Кубическая интерполяция сплайнами.
- 12. **ppval** - Оценка кусочно-непрерывных полиномов.

Геометрический анализ (Geometric analysis)

- 13. **delaunay** - Построение триангуляционной сетки (Delaunay triangulation).
- 14. **delaunay3** - Трехмерная сотовая сетка Делануа (3-D Delaunay tessellation).
- 15. **delaunayn** - Многомерная сотовая сетка Делануа (N-D Delaunay tessellation).
- 16. **dsearch** - Поиск ближайшей точки в триангуляции Делануа.
- 17. **dsearchn** - Поиск ближайшей точки в многомерной сотовой сетке Делануа.
- 18. **tsearch** - Ближайшая точка двумерной триангуляции.
- 19. **tsearchn** - Ближайшая точка многомерной триангуляции.
- 20. **convhull** - Построение двумерной выпуклой оболочки.
- 21. **convhulln** - Построение многомерной выпуклой оболочки.
- 22. **voronoi** - Построение диаграммы Вороного.
- 23. **voronoin** - Построение многомерной диаграммы Вороного.
- 24. **inpolygon** - Истинно для точек области, ограниченной многоугольником.
- 25. **rectint** - Площади областей пересечения двух семейств прямоугольников.
- 26. **polyarea** - Площадь прямоугольника.

Полиномы (Polynomials)

- 27. **roots** - Нахождение корней полиномов.
- 28. **poly** - Вычисление характеристического полинома матрицы или определение полинома с заданными корнями.
- 29. **polyval** - Вычисление значений полиномов в заданных точках.
- 30. **polyvalm** - Вычисление значений матричного полинома.
- 31. **residue** - Разложение на простые дроби (вычисление вычетов).
- 32. **polyfit** - Аппроксимация данных полиномом.
- 33. **polyder** - Вычисление производных от полиномов.
- 34. **polyint** - Аналитическое интегрирование полиномов.
- 35. **conv** - Умножение полиномов.
- 36. **deconv** - Деление полиномов.

Приложение 8. Анализ данных и преобразование Фурье (Data analysis and Fourier transforms)

Директория `matlab\datafun`

Основные операции (Basic operations)

- 1. **max** - Определение максимальных элементов массива.
- 2. **min** - Определение минимальных элементов массива.
- 3. **mean** - Определение средних значений элементов массива.
- 4. **median** - Определение медиан (срединных значений).

- 5. **std** - Определение стандартных отклонений элементов массива.
- 6. **var** - Определение дисперсий элементов массива.
- 7. **sort** - Сортировка элементов массива.
- 8. **sortrows** - Сортировка строк матриц.
- 9. **sum** - Суммирование элементов массива.
- 10. **prod** - Произведение элементов массива.
- 11. **hist** - Построение гистограммы.
- 12. **histc** - Подсчет элементов гистограммы.
- 13. **trapz** - Численное интегрирование методом трапеций.
- 14. **cumsum** - Куммулятивная сумма элементов массива.
- 15. **cumprod** - Куммулятивное произведение элементов массива.
- 16. **cumtrapz** - Куммулятивное численное интегрирование методом трапеций.

Конечные разности (Finite differences)

- 17. **diff** - Вычисление конечных разностей и приближенное дифференцирование.
- 18. **gradient** - Приближенное вычисление градиента функций.
- 19. **del2** - Дискретная аппроксимация дифференциального оператора Лапласа.

Корреляционные соотношения (Correlation)

- 20. **corrcoef** - Вычисление коэффициентов корреляции.
- 21. **cov** - Вычисление ковариационной матрицы.
- 22. **subspace** - Вычисление угла между двумя подпространствами.

Фильтрация и свертка (Filtering and convolution)

- 23. **filter** - Одномерная цифровая фильтрация.
- 24. **filter2** - Двумерная цифровая фильтрация.
- 25. **conv** - Свертка и умножение полиномов.
- 26. **conv2** - Двумерная свертка.
- 27. **convn** - N-мерная (многомерная) свертка.
- 28. **deconv** - Обращение свертки и деление полиномов.
- 29. **detrend** - Удаление линейного тренда.

Преобразование Фурье (Fourier transforms)

- 30. **fft** - Дискретное преобразование Фурье.
- 31. **fft2** - Двумерное дискретное преобразование Фурье.
- 32. **fftn** - Многомерное дискретное преобразование Фурье.
- 33. **ifft** - Обратное дискретное преобразование Фурье.
- 34. **ifft2** - Двумерное обратное дискретное преобразование Фурье.
- 35. **fftn** - Многомерное обратное дискретное преобразование Фурье.
- 36. **fftshift** - Перенос нулевой частоты в середину спектра.
- 37. **ifftshift** - Аннулирование переноса нулевой частоты в середину спектра.

Приложение 9. Функции обработки символьных строк (Character strings)

Директория `matlab\strfun`

Общие (General)

1. **char** - Сформировать массив символов (строку).
2. **double** - Преобразовать символы строки в числовые коды.
3. **cellstr** - Преобразовать массив символов в массив ячеек для строк.
4. **blanks** - Сформировать строку пробелов.
5. **deblank** - Удалить пробелы в конце строки.
6. **eval** - Выполнение выражения, записанного в виде строки символов.

Проверка строк (String tests)

7. **ischar** - Истинно, если это массив символов.
8. **iscellstr** - Истинно, если это массив ячеек для строк.
9. **isletter** - Истинно, если это символ (буква) алфавита.
10. **isspace** - Истинно, если это пробел.

Операции над строками (String operations)

11. **strcat** - Горизонтальное объединение строк.
12. **strvcat** - Вертикальное объединение строк.
13. **strcmp** - Сравнить строки.
14. **strncmp** - Сравнить первые N символов строк.
15. **strcmpi** - Сравнить строки игнорируя регистр.
16. **strncmpi** - Сравнить первые N символов строк игнорируя регистр.
17. **findstr** - Найти заданную строку в составе другой строки.
18. **strjust** - Выравнивать массив символов.
19. **strmatch** - Найти все совпадения.
20. **strrep** - Заменить одну строку другой.
21. **strtok** - Найти часть строки, ограниченную разделителями (token).
22. **upper** - Перевести все символы строки в верхний регистр.
23. **lower** - Перевести все символы строки в нижний регистр.

Преобразования строк (String to number conversion)

24. **num2str** - Преобразование числа в строку.
25. **int2str** - Преобразование целых чисел в строку.
26. **mat2str** - Преобразование матрицы в строку.
27. **str2double** - Преобразование строки в число удвоенной точности.
28. **str2num** - Преобразование массива строк в числовой массив.
29. **sprintf** - Записать форматированные знаки в виде строки.
30. **sscanf** - Прочитать строку с учетом формата.

Преобразование систем счисления (Base number conversion)

- 31. **hex2num** - Преобразовать шестнадцатеричное число в число удвоенной точности.
- 32. **hex2dec** - Преобразовать шестнадцатеричное число в десятичное число.
- 33. **dec2hex** - Преобразовать десятичное число в шестнадцатеричное число.
- 34. **bin2dec** - Преобразовать двоичную строку в десятичное число.
- 35. **dec2bin** - Преобразовать десятичное число в двоичную строку.
- 36. **base2dec** - Преобразовать В-строку в десятичное число.
- 37. **dec2base** - Преобразовать десятичное число в В-строку.

Справочник по базовым функциям системы MATLAB

Функция PLOT

Назначение - Двумерный линейных график

Синтаксис - `plot(Y)`
`plot(X1, Y1, ...)`
`plot(X1, Y1, LineSpec, ...)`
`plot(..., 'PropertyName', PropertyValue, ...)`
`h = plot(...)`

Описание. Если прямоугольный числовой массив **Y** является действительным, то функция **plot(Y)** строит линейные графики столбцов массива **Y** в зависимости от индексов их элементов. Если **Y** содержит комплексные числа, то запись **plot(Y)** эквивалентна записи **plot(real(Y), imag(Y))**. Во всех остальных случаях, при использовании **plot** мнимые компоненты игнорируются.

Команда **plot(X1, Y1, ...)**, где количество пар массивов может быть произвольным, осуществляет следующие построения.

- Если оба массива одномерные, то строится линейный график функции, где одномерный массив **X1** соответствует значениям аргумента, а одномерный массив **Y1** – значениям функции.
- Если оба массива **X1** и **Y1** – двумерные (они должны иметь при этом одинаковую размерность), то строятся попарно линейные зависимости столбцов **Y1** от соответствующих столбцов **X1**.
- Если массив **Y1** двумерный, а массив **X1** одномерный, то строятся линейные графики всех столбцов или строк массива **Y1** в зависимости от элементов вектора **X1**. Выбор столбцов или строк массива **Y1** здесь определяется размерностью вектора **X1**, то есть это могут быть и столбцы и строки (если массив **Y1** - квадратный, то приоритет отдается столбцам)
- Если двумерным является массив **X1**, а массив **Y1** одномерный (то есть вектор), то строятся графики столбцов или строк массива **X1** в зависимости от элементов вектора **Y1** (см. также предыдущий вариант).

Функция **plot(X1, Y1, LineSpec,...)** осуществляет приведенные выше построения, причем в спецификации линий **LineSpec** можно указать типы линий, символы маркеров и их цвета. Вы можете комбинировать тройки **Xn, Yn, LineSpec** с парами **Xn, Yn**, т.е. задавать спецификации **LineSpec** только для некоторых пар массивов, например: **plot(X1, Y1, X2, Y2, LineSpec, X3, Y3)**.

Функция **plot(..., 'PropertyName', PropertyValue,...)** устанавливает свойство всех построенных линий, указанное строкой **'PropertyName'**, в соответствие с его значением, заданным в **PropertyValue** (См. раздел «Примеры» ниже).

h = plot(...) возвращает вектор-столбец всех дескрипторов (handles) построенных на графике линий, по одному дескриптору на каждую линию.

Замечания. Если вы не зададите цвета при построении более чем одной линии, то функция **plot** циклически выбирает последовательные цвета так, как они указаны в свойстве *ColorOrder* текущих осей. После того как все цвета, определенные в *ColorOrder*, будут использованы, функция **plot** использует циклически стили линий (line styles) так, как они определены в свойстве *LineStyleOrder* текущих осей. При этом, после перехода к каждому новому стилю линии, происходит циклический выбор всех определенных цветов. Отметим, что по умолчанию, система MATLAB восстанавливает свойства *ColorOrder* и *LineStyleOrder* при каждом новом вызове функции **plot**. Если вы хотите сохранить введенные вами в данные свойства изменения, то вы должны задать эти изменения как значения по умолчанию. Например, команда

```
set(0, 'DefaultAxesColorOrder', [0 0 0], 'DefaultAxesLineStyleOrder', '-|-.|--|:')
```

задает на корневом (экранном) уровне использование только черного цвета при построении линий (свойство *ColorOrder*), а также использование следующих стилей линий (свойство *LineStyleOrder*) : сплошная линия (-), штрих-пунктирная линия (-.), штриховая линия (--) и пунктирная линия (:).

Дополнительная информация

- Для получения более обширной информации по заданию стилей и цветов линии см. раздел **LineSpec**.

Примеры

1. Задание цвета и размера маркеров

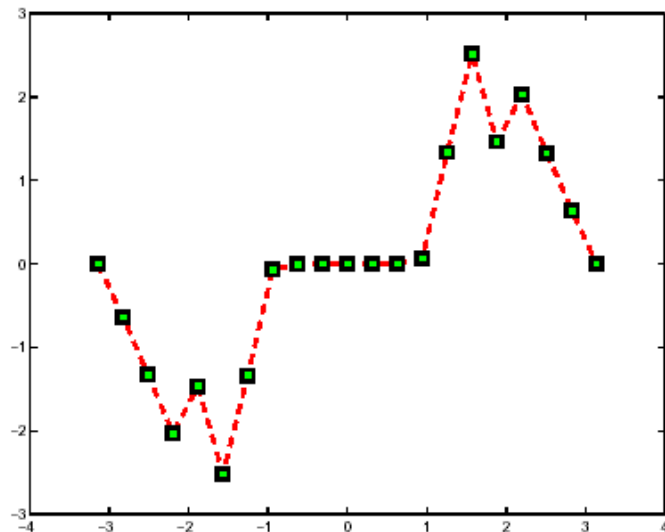
Помимо цвета и стиля, имеется также возможность задавать и другие графические характеристики линий (более подробное описание этих свойств дано в описании функции **line**):

- *LineWidth* – задание ширины линии в точках (points), где 1 точка равна 1/72 дюйма.
- *MarkerEdgeColor* – задание цвета маркера или цвета граней маркера для «заполненных» маркеров (кружков, квадратов, ромбов, пентаграмм (пятиугольников), гексаграмм (шестиугольников), и четырех типов треугольников).
- *MarkerFaceColor* – задание цвета поверхности заполненных маркеров.
- *MarkerSize* – задание размера маркера в единицах точки.

Например, выражения

```
x = -pi : pi/10 : pi;  
y = tan(sin(x)) - sin(tan(x));  
plot(x, y, '--rs', 'LineWidth', 2,...  
     'MarkerEdgeColor', 'k',...  
     'MarkerFaceColor', 'g',...  
     'MarkerSize', 10)
```

дают следующий график



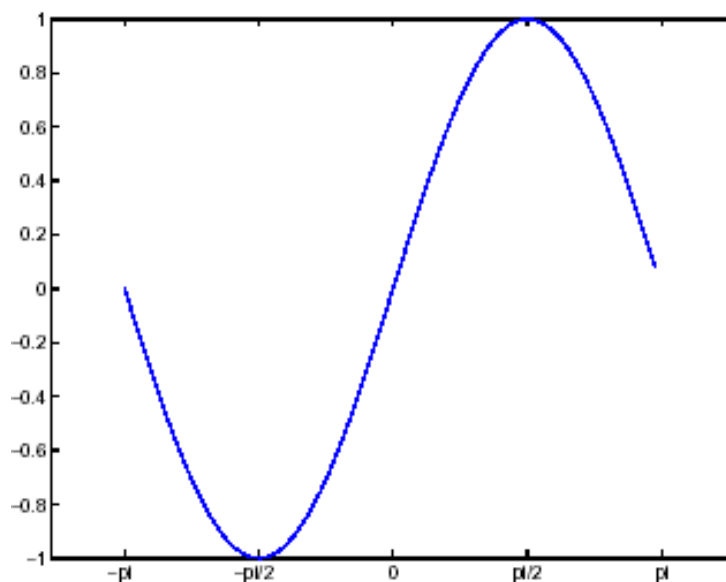
2. Задание положений меток (Tick-Mark) и указание надписей (Labeling)

Вы имеете возможность устанавливать положение меток на осях, а также вид надписей на каждой метке. Например, в следующем графике синусоидальной функции производится изменение маркировки оси x , с целью придания более значимых и информативных значений меток и их надписей:

```
x = -pi : 0.1 : pi;
y = sin(x);
plot(x, y)
```

```
set(gca, 'XTick', -pi : pi/2 : pi)
```

```
set(gca, 'XTickLabel', {'-pi', '-pi/2', '0', 'pi/2', 'pi'})
```



Добавление заголовков, надписей к осям и аннотаций

Добавим теперь надписи к осям и комментарий к точке $-\pi/4, \sin(-\pi/4)$.

Следующие три стандартные команды позволяют ввести надписи к осям x и y , а также общий заголовок к графику

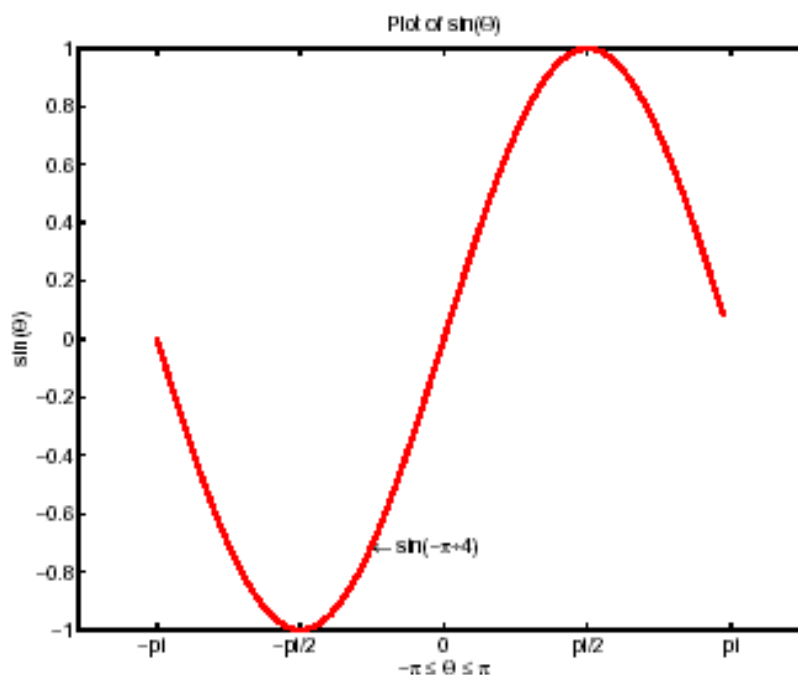
```
xlabel('-\pi \leq \Theta \leq \pi')
ylabel('sin(\Theta)')
title('Plot of sin(\Theta)')
```

а команда **text** дает возможность ввести текстовую информацию в любую выбранную точку координатных осей

```
text(-pi/4, sin(-pi/4), '\leftarrow sin(-\pi/4)', 'HorizontalAlignment', 'left')
```

Изменим также цвет линии на красный, найдя сперва дескриптор линии, созданной функцией **plot**, а затем установив требуемое свойство линии *Color* property. В том же выражении установим толщину линии *LineWidth* равной двум точкам

```
set(findobj(gca, 'Type', 'line', 'Color', [0 0 1]),...
'Color','red',...
'LineWidth', 2)
```



См. также функции:

axis, bar, grid, legend, line, LineSpec, loglog, plotyy, semilogx, semilogy, subplot, xlabel, xlim, ylabel, ylim, zlabel, zlim, stem

См. свойство **String** функции **text**, где дается список символов и описывается их применение.

LineSpec

Назначение. Спецификации задания свойств графического объекта **линия (Line)**.

Описание. В данном разделе описывается как можно задать свойства линий, используемых при построении графиков. Система MATLAB дает возможность задавать многие характеристики, включая:

- Стиль линии (Line style)
- Толщину линии (Line width)
- Цвет (Color)
- Тип маркера (Marker type)
- Размер маркера (Marker size)
- Цвета поверхности и граней маркера (для заполненных маркеров)

MATLAB предусматривает специальные символьные спецификаторы (описатели) для стилей линий, типов маркеров и цветов. В следующих таблицах дается перечисление этих спецификаторов.

Спецификаторы стилей линии

Спецификаторы	Стили линии
-	Сплошная линия (по умолчанию)
--	Штриховая линия
:	Пунктирная линия
-.	Штрих-пунктирная линия

Спецификаторы маркеров

Спецификаторы	Типы маркеров
+	Знак плюс
o	Кружочек
*	Звездочка
.	Точка
x	Крестик
s	Квадрат
d	Ромб
^	Треугольная стрелка вверх
v	Треугольная стрелка вниз
>	Треугольная стрелка направо
<	Треугольная стрелка налево
p	Пятиугольная звезда (пентаграмма)
h	Шестиугольная звезда (гексаграмма)

Спецификаторы цвета

Спецификатор	Цвет
r	Красный
g	Зеленый
b	Синий
c	Голубой (cyan)
m	Магента (magenta)
y	Желтый
k	Черный
w	Белый

Многие графические функции допускают аргумент **LineStyle**, который определяет три спецификатора для характеристики линии:

- Стиль линии
- Тип маркера
- Цвет

Например, функция **plot(x, y, '-.or')** строит график значений y от аргумента x , используя штрих-пунктирную линию ($-.$); размещает круглые маркеры (o) в точках данных, и окрашивает как линию, так и маркеры в красный цвет (r). Данные спецификаторы нужно задать (в любом порядке) как строка символов в кавычках, после записей массивов данных. Если вы зададите в **LineStyle** только маркеры, но не стиль линии (например, **plot(x,y,'d')**), MATLAB наносит только маркеры (без линий)

Связанные (родственные) свойства. При использовании функций **plot** и **plot3**, вы можете задавать также другие характеристики линий, используя следующие графические свойства:

- *LineWidth* – задает ширину линии (в точках, равных 1/72 дюйма)
- *MarkerEdgeColor* – задает цвет маркера или цвет граней для заполненных маркеров (кружок, квадрат, ромб, пентаграмма, гексаграмма, и четыре треугольника).
- *MarkerFaceColor* – задает цвет поверхности заполненного маркера
- *MarkerSize* – задает размер маркера в точках

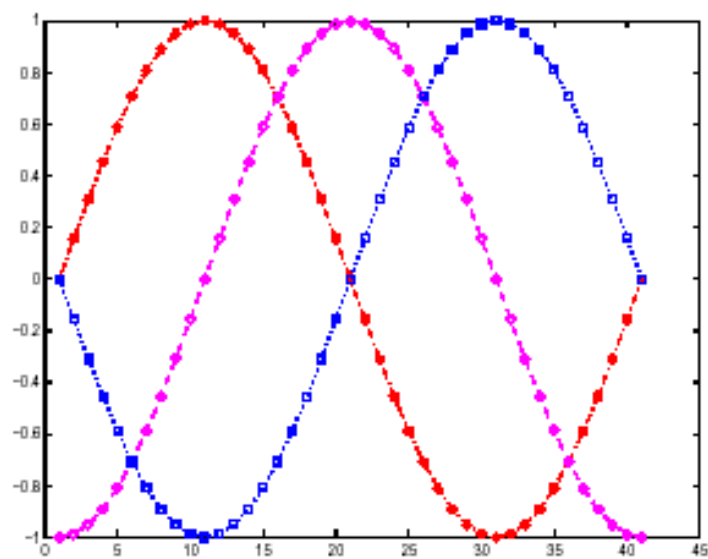
В дополнение, вы можете задавать графические свойства *LineStyle*, *Color*, и *Marker* вместо использования символьной строки. Это может быть полезным, например, если вы хотите задать цвет, которого нет в приведенном выше списке спецификаторов цвета, при помощи тройки значений RGB. Более подробная информация о возможностях выбора цвета дана в разделе **ColorSpec**.

Примеры

Построим синусоидальную функцию для трех различных пределов изменения аргумента, используя различные стили линий, цвета и маркеры.

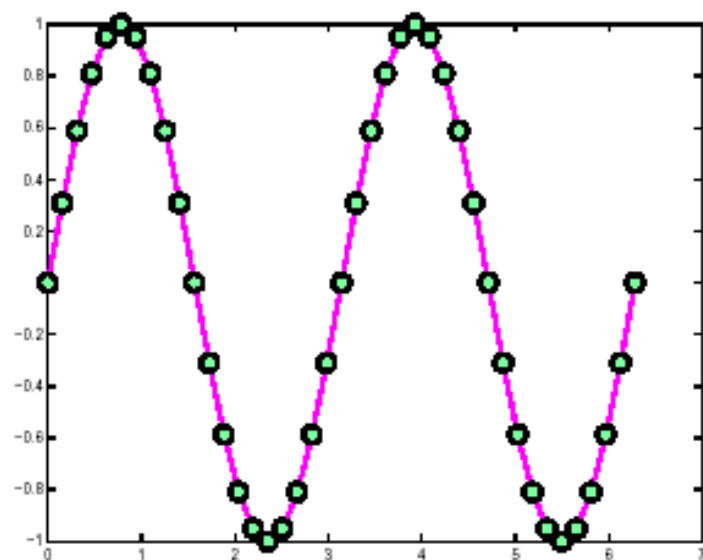
```
t = 0 : pi/20 : 2*pi;  
plot(t, sin(t), '-.r*')  
hold on  
plot(sin(t - pi/2), '--mo')  
plot(sin(t - pi), ':bs')
```

hold off



Построим еще один график, иллюстрирующий как можно задавать свойства линий.

```
plot(t, sin(2*t), '-mo',...  
      'LineWidth', 2,...  
      'MarkerEdgeColor', 'k',...  
      'MarkerFaceColor', [0.49 1 0.63],...  
      'MarkerSize', 12)
```



См. также функции:

line, plot, patch, set, surface, и свойство *LineStyleOrder* координатных осей