В. В. Терёхин

Моделирование в системе MATLAB

Часть1. Основы работы в MATLAB

Практическое пособие

Новокузнецк 2005

Министерство Образования
Российской Федерации

Кемеровский государственный университет

В. В. Терёхин

Моделирование в системе MATLAB

Часть1. Основы работы в MATLAB

Практическое пособие

Новокузнецк 2005

ББК 32.973
УДК - 681.142.2
Т-

Печатается по решению редакционно-издательского совета Кемеровского государственного университета.

Рецензенты:
……
……..

Терёхин В.В.
Т- Моделирование в системе MATLAB:Учебное пособие /Кемеровский государственный университет. – Новокузнецк: Кузбассвузиздат, 2004. -376с.

ISBN ……

Автор скомпоновал основные сведения для работы в MATLAB, необходимые для выполнения учебных задач по моделированию при изучении различных дисциплин в области математики, информатики, теории управления и т.п. В данной работе использованы материалы книг Потёмкина В.Г. (Введение в MATLAB), Гультяева А..К. (MATLAB 5.3), Черных И.В. (SIMULINK) и Лазарева Ю.Ф. (MatLAB 5.x). Книга рассчитана на студентов 2-5 курсов по специальности «Прикладная математика и информатика», «Автоматизированные системы обработки информации и управления», «Прикладная информатика в экономике».

К …….. **ББК 32.81**

ISBN ……

Содержание

Введение

Для сотен тысяч специалистов в различных отраслях промышленности, занятых инженерными и научными исследованиями, система **MATLAB** обеспечила превосходную среду для организации вычислений. Поэтому знакомство с основами организации системы **MATLAB** может быть полезно как специалистам, приступающим к освоению этой системы, так и студентам университетов и вузов по самым различным специальностям.

В пособии описаны операционная среда системы **MATLAB**, инструментальное средство **Notebook**, элементы программирования и отладки программ, а также все типы данных и объектно-ориентированный подход, связанный с введением новых классов объектов. Эта информация является основой для эффективной работы в системе **MATLAB,** которая применяется в таких курсах как алгоритмы и численные методы, линейная алгебра, прикладная математика, теория управления, цифровая обработка сигналов и изображений, курсах по специальностям. Система **MATLAB** и это пособие совместно с другими пособиями, могут быть эффективно использованы для того, чтобы раскрыть важность объединения фундаментального знания с современными информационными технологиями, конечной целью которого является формирование прикладного математика.

Зарождение системы MATLAB относится к концу 70-х годов, когда первая версия этой системы была использована в Университете Нью Мехико и Станфордском университете для преподавания курсов теории матриц, линейной алгебры и численного анализа. В это время активно разрабатывались пакеты прикладных программ по линейной алгебре LINPACK и EISPACK на языке FORTRAN, и авторы системы MATLAB искали способы использовать эти пакеты, не программируя на языке FORTRAN.

Сейчас возможности системы значительно превосходят возможности первоначальной версии матричной лаборатории Matrix Laboratory. Нынешний MATLAB - это высокоэффективный язык инженерных и научных вычислений. Он поддерживает математические вычисления, визуализацию научной графики и программирование с использованием легко осваиваемого операционного окружения, когда задачи и их решения могут быть представлены в нотации, близкой к математической. Наиболее известные области применения системы MATLAB:

- математика и вычисления;

- разработка алгоритмов;
- вычислительный эксперимент, имитационное моделирование, макетирование;
- 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, которое понадобилось бы для программирования на скалярных языках типа C или FORTRAN.

Версия MATLAB 6.1 - это последнее достижение разработчиков; она содержит существенные изменения и улучшения в каждом разделе, начиная от встроенных математических функций и новых конструкций программирования и заканчивая новыми структурами данных, объектно-ориентированным подходом, новыми средствами визуализации и графическим интерфейсом пользователя.

Одно из назначений математики - служить языком общения между учеными и инженерами. Матрицы, дифференциальные уравнения, массивы данных, графики - это общие объекты и конструкции, используемые как в прикладной математике, так и в системе MATLAB. Именно эта фундаментальная основа обеспечивает системе MATLAB непревзойденную мощь и доступность. Стоит прислушаться к следующему афористичному мнению: "Причина, по которой MATLAB столь полезен для обработки сигналов, состоит в том, что он не проектировался специально для этой цели, а создавался для математиков".

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 M-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 (ППП), которые представляют собой коллекции M-файлов для решения определенной задачи или проблемы.

В действительности ППП - это нечто большее, чем просто набор полезных функций. Часто это результат работы многих исследо-

вателей по всему миру, которые объединяются в зависимости от области применения - теория управления, обработка сигналов, идентификация и т. п. Именно поэтому пакеты прикладных программ - MATLAB Application Toolboxes, входящие в состав семейства продуктов MATLAB, позволяют находиться на уровне самых современных мировых достижений.

1 Операционная среда системы Matlab

Операционная среда системы MATLAB - это множество интерфейсов, которые поддерживают связь этой системы с внешним миром. Это - диалог с пользователем через командную строку или графический интерфейс, просмотр рабочей области и путей доступа, редактор и отладчик М-файлов, работа с файлами и оболчкой DOS, экспорт и импорт данных, интерактивный доступ к справочной информации, динамическое взаимодействие с внешними системами Microsoft Word, Excel Microsoft Word, Excel и др.. Реализуются эти интерфейсы через командное окно, инструментальную панель, системы просмотра рабочей области и путей доступа, редактор/отладчик М-файлов, специальные меню и т.п.

1.1 Командное окно. Инструментальная панель

Командное окно

Командное окно системы MATLAB показано на рисунке 1.1. Здесь же показано ниспадающее меню **File**. Оно содержит следующие опции:

Опция	Подопции	Назначение
New	**M-file Figure**	Открыть в редакторе/отладчике новый файл

		Открыть графическое окно
Open		Открыть в редакторе/отладчике указанный файл
Open Selection		Открыть в редакторе/отладчике файл, выделенный в произвольной строке командного окна
Run Script		Вызов окна для запуска **Script**-файла
Load Workspace		Вызов окна загрузки **MAT**-файла
Save Workspace As		Вызов окна сохранения **MAT**-файла
Show Workspace		Вызов средства просмотра рабочей области **Workspace Browser**
Set Path		Вызов средства просмотра путей доступа **Path Browser**
Preferences		Выбор характеристик
Print Setup		Установка опций принтера
Print		Установка опций вывода на печать
Print Selection		Печать выделенного фрагмента

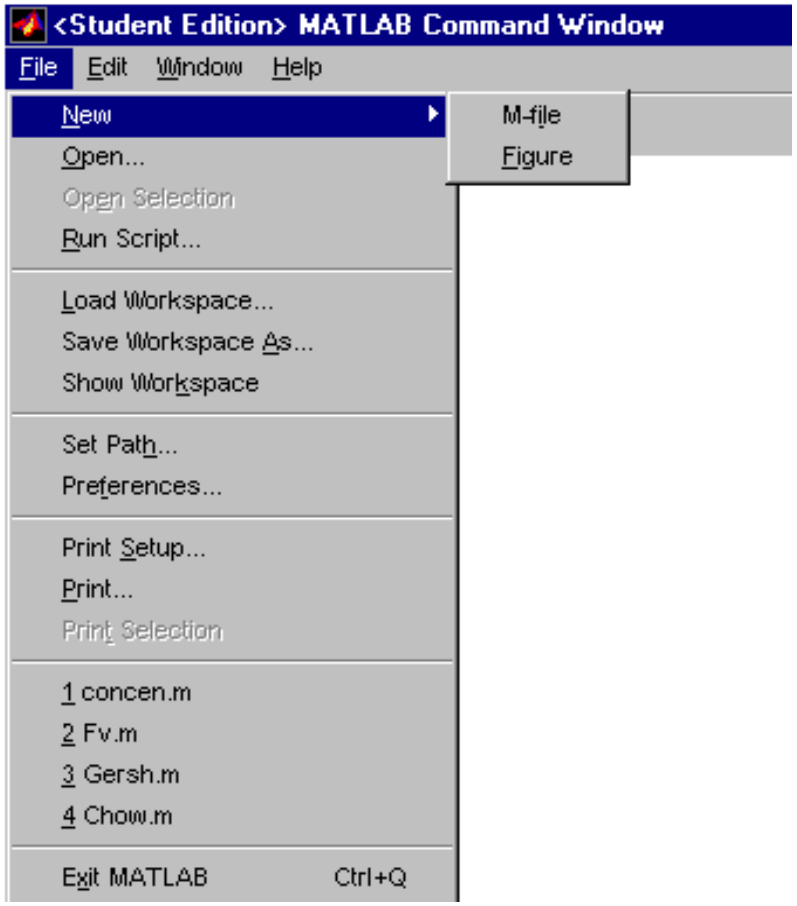


Рисунок 1.1

Особого рассмотрения заслуживает опция **Preferences** (Выбор характеристик), которая включает 3 вкладки. В первую очередь, рассмотрим вкладку **General** (Общее). В ней можно увидеть 3 поля и 3 маркера, имеющие следующие назначения:

Поле или маркер	**Назначение**
Numeric Format	Выбор формата представления чисел и межстрочного пробела. По умолчанию, формат **Short,** пробел **Loose**

Editor Preference	Выбор текстового редактора По умолчанию, встроенный редактор **Built in Editor**
Help Directory	Каталог справки **Help**
Echo on	Показывать на экране команды исполняемого **Script**-файла сценария/Не показывать
Show Toolbar	Показывать на экране инструментальную панель/Не показывать
Enable Graphical Debugging	Поддерживать отладку графики/Не поддерживать

Далее рассмотрим вкладку **CommandWindowFont** (Шрифт для командного окна). На ней можно увидеть 6 полей и 1 маркер, имеющие следующие назначения:

Поле или маркер	***Назначение***
Font	Шрифт для вывода текста в командном окне
Style	Тип шрифта: **Light** - светлый **Regular** - нормальный **Bold** - жирный
Size	Размер шрифта: 10 12 15
Background Color	Цвет фона: **Silver** - серебристый **Red** - красный **Lime** - лимонный **Yellow** - желтый **Blue** - синий **Fuscia** - светло-фиолетовый **Aqua** - голубой **White** - белый
Color	Цвет символа: **Black** - черный **Maroon** - каштановый **Grren** - зеленый **Olive** - оливковый **Navy** - темно-синий **Purple** - темно-фиолетовый **Teal** - зелено-голубой **Gray** - серый
Sample	Образец фона и шрифта

Display Fixed Pitch Fonts Only	Показать только шрифты с фиксированным шагом/ Показать все шрифты

Наконец, рассмотрим вкладку **CopyingOptions** (Опции копирования). В ней можно увидеть 3 поля, имеющие следующие назначения:

Поле или маркер	Назначение
Clipboard Format	Формат копирования в буфер обмена: **Windows Metafile Windows Bitmap**
Honor figure size properties	Маркер воспроизедения размеров рисунка. Выбор этой опции позволяет копировать рисунок с учетом свойства "**Paper Position**" (для формата **Windows Bitmap** не действует)
White Background	Маркер белого/черного фона (для формата **Windows Bitmap** не действует)

Инструментальная панель

Инструментальная панель командного окна системы MATLAB позволяет обеспечить простой доступ к операциям над М-файлами (смотри рисунок 1.2).

New File | Open File | Cut | Copy | Paste | Undo | Workspace Browser | Path Browser | Help

Рисунок 1.2

Эти операции включают:

- создание нового М-файла **(New File)**;
- открытие существующего М-файла (**Open File**);
- удаление фрагмента (**Cut**);
- копирование фрагмента (**Copy**);
- вставка фрагмента (**Paste**);
- восстановление только выполненной операции (**Undo**);
- просмотр области (**Workspace Browser**);

- просмотр путей доступа (**Path Browser**);
- текущая рабочей помощь (**Help**).

1.2 Редактор/отладчик М-файлов

В состав системы MATLAB входит редактор/отладчик М-файлов **M-file Editor/Debugger**, который может быть вызван из командной строки командой **edit** или **edit** <имя М-файла>. Редактор/отладчик поддерживает следующие операции:

- создание нового М-файла (**New File**);
- открытие существующего М-файла (**Open File**);
- сохранение М-файла на диске (**Save to Disk**);
- удаление фрагмента (**Cut**);
- копирование фрагмента (**Copy)**;
- вставка фрагмента (**Paste**);
- текущая помощь (**Help**);
- продолжить выполнение (**Continue**);
- установить/удалить контрольную точку (**Set/Clear Breakpoint**);
- удалить все контрольные точки (**Clear All Breakpoints**);
- выполнить один шаг отладки (**Single Step**);
- войти в М-модуль (**Step In**);
- завершить отладку (**Quit Debugging**).

1.3 Рабочая область

Рабочая область системы MATLAB - это область памяти, в которой размещены переменные системы. Содержимое этой области можно просмотреть из командной строки с помощью команд **who** и **whos**. Команда **who** выводит только имена переменных, а команда **whos** - информацию о размерах массивов и типе переменной.

Рассмотрим в качестве примера 5 массивов различного типа:

- **A** - трехмерный массив чисел удвоенной точности;
- **B** - массив разреженной структуры;
- **C** - массив ячеек;
- **S** - массив символов;
- **patient** - массив записей.

По команде **whos** получим

Name	**Size**	**Bytes**	**Class**
A	**4x3x2**	**192**	**double array**
B	4x4	212	**sparse array**
C	4x3x2	2400	**cell array**
S	4x16	128	**char array**
patient	1x2	840	**struct array**

Grand total is 194 elements using 3772 bytes

Специальное средство просмотра **Workspace Browser** обеспечивает представление команды **whos** в виде графического интерфейса. Для того чтобы открыть **Workspace Browser** надо либо выбрать опцию **Show Workspace** из меню **File**, либо воспользоваться кнопкой **Workspace Browser** инструментальной панели.

В результате этих операций на терминал будет выедено следующее окно (рисунок 1.3)

В этом окне можно выполнить следующие операции:

- удалить переменную, если выделить ее и нажать кнопку **Delete**;
- закрыть окно с помощью кнопки **Close**.

Кроме того, можно изменять размеры колонок посредством перемещения их границ с помощью мыши. Можно выполнить переименование переменной, если сначала выделить ее, затем однократно щелкнуть левой клавишей мыши (заметим, что двойной щелчок никакого действия не оказывает). После короткой задержки появляется поле, в котором можно указать новое имя; и наконец, следует нажать клавишу **Enter**, чтобы подтвердить завершение операции.

Загрузка и сохранение рабочей области

Команды **save** и **load** позволяют в любой момент времени сохранить содержимое рабочей области или загрузить новые данные в процессе сеанса работы. С помощью этих команд можно также осуществлять экспорт и импорт ASCII-файлов.

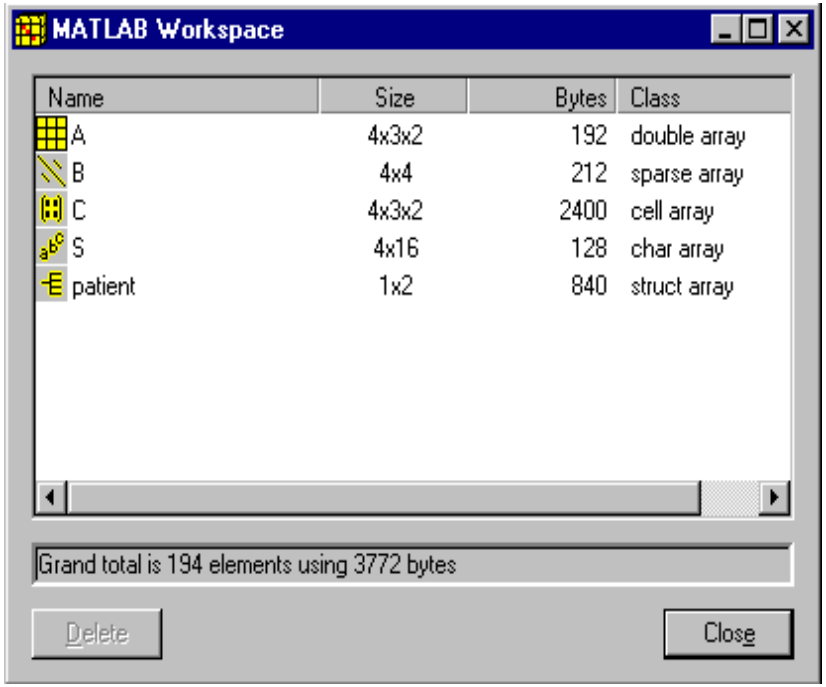


Рисунок 1.3

Сохранение переменных рабочей области. Команда **save** позволяет сохранить содержимое рабочей области в двоичном MAT-файле,который можнов дальнейшем вызваль командой **load**. Команда **save** также доступна в качестве опции **Save Workspace** меню **File.**

Спецификация формата файла. Для того чтобы управлять форматами файлов, следует в команде **save** в дополнение к имени файла и списку переменных использовать следующие флаги:

Флаг	***Пояснение***
-mat	Двоичный MAT-файл (по умолчанию)
-ascii	ASCII-формат (8 цифр)
-ascii -double	ASCII-формат (16 цифр)
-ascii -double -tabs	Формат с разделителями и метками табуляции
-v4	Формат версии MATLAB 4
-append	Добавить данные к сущесвующему MAT-файлу

При использовании флага **v4** можно сохранить только те данные, которые совместимы с данными, используемыми в версии MATLAB 4; это означает, что сохранить такие типы данных как массивы записей, ячеек, многомерные массивы или объекты нельзя. Когда содержимое рабочей области сохраняется в ASCII-формате, то рекомендуется единовременно сохранять только одну переменную. Если сохраняется более одной переменной, то система MATLAB создаст файл ASCII-файл, который нельзя будет в дальнейшем загрузить в MATLAB, используя команду **load**.

Загрузка рабочей области. Команда l**oad** позволяет загрузить MAT-файл, который был ранее сохранен с помощью команды **save**. При загрузке MAT-файла новые значения одноименных переменных будут записаны взамен старых. Если MAT-файл имеет расширение, отличающееся от .**mat**, то необходимо использовать флаг -**mat**; в противном случае MATLAB будет считать форматом файла ASCII-формат.

Загрузка файлов данных в ASCII-формате. Команда **load** позволяет выполнять импорт файлов данных в ASCII-формате; она преобразует содержимое файла в переменную с именем файла только без расширения. Например, применение команды **load tides.dat** создает в рабочей области системы MATLAB переменную с имененм **tides**. Если исходный файл в ASCII-формате имеет **m lines** строк с **n** значениями в каждой строке, то результаом будет массив чисел размера **m?n**.

Использование имен в формате строк. Если имена файлов и переменных представляют собой строковые переменные, то можно, используя свойство дуальности команды и функции, рассматривать команды **load** и **save** как функции. В этом случае входные переменные должны следовать в том же порядке, как они следовали в командной строке. Например, последовательность операторов

```
save('myfile', 'VAR1', 'VAR2')
A = 'myfile';
load(A)
```

это то же самое, что и последовательность команд

```
save myfile VAR1 VAR2
load myfile
```

Для сохранения или загрузки последовательности файлов, имена которых имеют общий корень и дополнительный целочисленный суффикс, необходимо использовать структуру цикла. Например,

следующая конструкция позволяет сохранить квадраты чисел от 1 до 10 в файлах с именами **data1, …, data10**:

```
file = 'data';
for i = 1:10
j = i.^2;
save([file int2str(i)],'j');
end
```

Использование группового символа. Команды **load** и **save** допускают использование группового символа (*****) в качестве замены ряда символов в шаблоне имени переменной.

Например, команда **save rundate x*** сохраняет все переменные, имена которых начинаются с символа **x** в файле с именем **rundata.mat**.

Точно также команда **load testdata ex1*95** загружает все переменные, имена которых начинаются с символов **'ex1'** и заканчиваются символами **'95'**, независимо от того, какие символы размещены между ними.

1.4 Список путей доступа

Для поиска М-файлов система MATLAB использует механизм путей доступа, поскольку М-файлы записываются в каталоги или папки файловой системы. Например, при поиске файла с именем **foo** MATLAB выполняет следующие действия:

- просматривает, не является ли **foo** именем переменной;
- просматривает, не является ли **foo** встроенной функцией;
- ищет в текущем каталоге М-файл с именем **foo.m**;
- ищет М-файл с именем **foo.m** во всех каталогах списка путей доступа.

Реально применяемые правила поиска являются более сложными из-за ограничений, которые связаны с использованием подфункций, частных функций и объектно-ориентированных механизмов. Однако приведенный выше приведенный порядок поиска точно отражает механизм поиска М-файлов, с которыми обычно работает пользователь. Работа со списком путей доступа. В процессе сеанса работы можно вывести на терминал или внести изменения в список путей доступа, используя следующие функции:

- **path** выводит на экран список путей доступа;

- **path(s)** заменяет существующий список списком **s;**
- **addpath /home/lib** и **path(path, '/home/lib')**-- добавляют новый каталог в список путей доступа;
- **rmpath /home/lib--** удаляет путь **/home/lib** из списка.

Список путей доступа, используемый по умолчанию, определен в файле **pathdef.m**, который размещен в каталоге **local**; этот файл выполняется при каждом запуске системы MATLAB. Кроме работы из командной строки существует средство просмотра путей доступа **Path Browser**, которое поддерживает удобный графический интерфейс для просмотра и изменения списка путей. Однако более предпочтительно вносить непосредственные изменения в М-файл **pathdef.m**, используя какой-либо текстовый редактор, в том числе и редактор/отладчик системы MATLAB.

Текущий каталог. Система MATLAB использует понятие текущего каталога при работе с М- и MAT-файлами во время сеанса работы. Начальный текущий каталог определен в файле запуска, который ассоциирован с ярлыком запуска системы MATLAB, расположенном на рабочем столе. Щелчок правой кнопки мыши, установленной на этом ярлыке, и выбор опции **Properties** позволяет изменить начальный каталог, используемый по умолчанию.

Для вывода текущего каталога на экран терминала предназначена команда **cd**. Для изменения текущего каталога следует использовать команду **cd** <новый путь доступа>.

Просмотр списка файлов. Мы уже видели, как команда **path** позволяет отобразить список путей доступа. В свою очередь, команда **what** позволяет увидеть список файлов, расположенных в заданном или текущем каталогах. Команда **what** без параметров выводит на экран список файлов текущего каталога, а команда **what** <полный или частичный путь доступа> выводит на экран список файлов заданного каталога.

Для распечатки содержимого М-файла предназначена команда **type** <имя файла>; для редактирования М-файла используется команда **edit** <имя файла>.

Средство просмотра путей доступа. На платформе PC имеется средство визуального просмотра путей доступа **Path Browser**, которое позволяет просматривать, модифицировать пути доступа и видеть списки всех файлов системы MATLAB. Для того чтобы открыть средство просмотра **Path Browser** следует использовать либо опцию **Set Path** из меню **File**, либо кнопку инструментальной панели **Path Browser**.

В окне средств просмотра путей доступа **Path Browser** имеются:

- поле **Current Directory** с кнопкой **Browse**, предназначенное для изменения текущего каталога;
- поле **Path** содержит список путей доступа;
- поле **Files in** <имя каталога>, выделенного в поле **Path** содержит список файлов и внутренних каталогов **типа private, @**;
- кнопки:

Add to Path	Добавить каталог в начало пути
Remove from Path	Удалить каталог из пути
Undo	Отменить предыдущую операцию
Refresh	Обновить, используя текущие установки
Save Settings	Сохранить установки в файле **pathdef.m**
Restore Defaults	Восстановить установки, принятые по умолчанию
Close	Закрыть **Path Browser**

Для перемещения каталога в другую позицию в **Path** следует захватить его левой кнопкой мыши и переместить в нужную позицию.

Если изменение списка путей доступа выполняется в командном окне, то для отражения этих изменений в средстве просмотра **Path Browser** необходимо использовать кнопку **Refresh**.

Все изменения, которые вносятся в список путей доступа, действуют только в течение сеанса работы; для того чтобы внести их в файл **pathdef.m** для постоянного использования необходимо воспользоваться кнопкой **Save Settings**.

1.5 Работа с файлами и оболочкой DOS. Импорт и экспорт данных

Работа с файлами и оболочкой DOS

Команды **cd, dir, delete, type** позволяют из командной строки системы MATLAB выполнить ряд команд DOS, связанных с управлением файлами. Приведенная ниже таблица отражает связь команд системы MATLAB с командами DOS:

MATLAB	MS-DOS
cd	**chdir**
dir	**dir**
delete	**del или erase**
type	**type**

Большинство этих команд позволяет указывать пути доступа, имена дисководов, использовать групповые символы.

Запуск внешних программ

Признаком перехода к выполнению команд DOS является знак "!", который указывает, что следующая за ним команда - это команда DOS. Это исключительно полезно при вызове утилит и выполнении внешних других программ без выхода из системы MATLAB.

Импорт и экспорт данных

Существует много приемов для перемещения данных между системой MATLAB и другими приложениями. В большинстве случаев при работе с данными системы MATLAB можно просто использовать команды чтения и записи файлов. Для более сложных наборов данных можно создать собственные программы для чтения и записи на языках **C** или **Fortran**. Импортирование данных в систему MATLAB. Существует несколько способов для передачи данных из других приложений в систему MATLAB. Выбор способа зависит от объема и формата данных.

- Ввод данных в виде списка. Если количество данных невелико, то их можно просто напечатать, помещая в квадратные скобки. Этот метод неудобен при большом количестве данных, поскольку их невозможно редактировать.

- Формирование данных в М-файле.Используя текстовый редактор, можно сформировать М-файл, в колором данные представлены как список элементов, это тот же первый способ, но он имеет то преимущество,что позоляет с помощью редактора корректировать данные. Достаточно после исправления перезапустить М-файл, чтобы ввести исправленные данные.
- Загрузка данных из ASCII-файла. ASCII-файлы накапливают данные в 7-разрядном коде без контроля по четности. Каждая строка содержит одинаковое количество значений, разделенных пробелами, и завершается символом возврата каретки. Эти файлы можно редактировать, используя обычный текстовый редактор. Их можно читать непосредственно в системе MATLAB, используя функцию **load**. При этом создается переменная, имя которой совпадает с именем файла. Можно воспользоваться функций **dlmread**, чтобы указать другой тип разделителя.
- Чтение данных с использованием функций ввода/вывода. Применение функций ввода/вывода, а также функций fopen и **fread**, полезно при загрузке файлов данных из других приложений, использующих специальные форматы данных.
- Использование специальных средств для чтения файлов. Для чтения файлов, записанных в специальных форматах, в системе MATLAB имеются следующие специализированные функции:

Функция	***Назначение***
dlmread	Чтение ASCII-файлов
wk1read	Чтение электронных таблиц в формате WK1
imread	Чтение изображения из графического файла
auread	Чтение звукового файла с расширением .au (формат фирмы SUN Microsystems)
wavread	Чтение звукового файла с расширением .wav (формат фирмы Microsoft)

- Создание MEX-файла. Наилучший способ создания программ для чтения данных - это использовать уже имеющиеся программы на языках **C** или **Fortran** для чтения данных из дру-

гих приложений. Однако этот метод, называемый смешанным программированием, требует написания специальных программ-связок, оформляемых в виде MEX-файлов.

- Разработка программы на языках **Fortran** или **C**. Программисты, использующие языки **Fortran** или **C**, могут написать специальные программы для преобразования данных в формат MAT-файла системы MATLAB. В этом случае преобразованные данные могут быть загружены в систему MATLAB с помощью обычной команды **load**.

Экспортирование данных из системы MATLAB

Существует несколько способов для передачи данных из системы MATLAB в другие приложения:

- Использование команды **diary**. Для массивов небольших размеров можно использовать команду **diary**, чтобы создать файл дневника, который включает команды MATLAB, используемые в течение сеанса работы, а также позволяет на экране просмотреть необходимые данные. Записи дневника могут быть полезны для вложения в документы или отчеты. В дальнейшем можно использовать текстовый редактор для редактирования дневника.
- Сохранение данных в формате ASCII. Команда **save** с опцией **-ascii** позволяет записать данные в этом формате, причем, используя команду **dlmwrite**, можно задать другой тип разделителя.
- Использование специальных средств для записи файлов. Для записи файлов в специальных форматах, определяемых приложениями, в системе MATLAB имеются следующие специализированные функции:

Функция	***Назначение***
dlmwrite	Запись данных в ASCII-файл
wk1write	Запись данных в электронную таблицу в формате **WK1**
imwrite	Запись изображения в графический файл
auwrite	Запись данных в звуковой файл с расширением **.au** (формат фирмы SUN Microsystems)

wavwrite	Запись данных в звуковой файл с расширением **.wav** (формат фирмы Microsoft)

- Создание MEX-файла. Наилучший способ создания программ для записи данных - это использовать уже имеющиеся программы на языках **C** или **Fortran** для записи данных в другие приложения. Однако этот метод, называемый смешанным программированием, требует написания специальных программ-связок, оформляемых в виде MEX-файлов.
- Разработка программы на языках **Fortran** или **C**. Программисты, использующие языки **Fortran** или **C**, могут написать специальные программы для преобразования данных из формата MAT-файла системы MATLAB в формат приложения. В этом случае данные могут быть выгружены из системы MATLAB с помощью обычной команды **save**.
- Текстовые файлы с разделителями. Функции **dlmread** и **dlmwrite** позволяют читать и записывать данные, отделенные разделителем, используя ASCII-файл. В качестве разделителя может быть использован любой символ, который отделяет одно значение от другого.

Например, рассмотрим файл с именем **ph.dat**, который содержит данные, разделенные точкой с запятой:

7.2; 8.5; 6.2; 6.6
5.4;9.2;8.1;7.2

Для того чтобы прочитать содержимое этого файла в массив с именем **A**, надо использовать следующий оператор

A = dlmread('ph.dat', ';');

Второй аргумент функции **dlmread** указывает тип разделителя. В дополнение к разделителю, который вы используете, функция **dlmread** также считает разделителями имеющиеся пробелы. Поэтому функция **dlmread**, приведенная выше, будет работать правильно, если даже содержимое файла **ph.dat** будет таким:

7.2; 8.5; 6.2; 6.6
5.4; 9.2 ;8.1; 7.2

Предупреждение.

Первый аргумент М-функции **dlmread** - это имя файла, а не идентификатор файла. Поэтому не надо предварительно открывать файл с помощью функции **fopen**, а следует сразу применять функции **dlmread** и **dlmwrite**. Продемонстрируем, как функция **dlmwrite** выполняет запись текста с разделителями во внешний файл с именем **myfile**, используя разделитель **";"**:

```
A =
1 2 3
4 5 6
dlmwrite('myfile',A,';')
1; 2; 3
4; 5; 6
```

Обмен файлами данных для различных платформ

Иногда оказывается необходимо работать с версиями системы MATLAB для разных вычислительных платформ или передавать разработанные приложения на другие системы. Приложения, создаваемые в системе MATLAB могут включать М-файлы, представляющие собой М-функции или М-сценарии, а также MAT-файлы, содержащие двоичные данные. Оба типа файлов могут быть непосредственно использованы на различных платформах:

- М-файлы являются ASCII-файлами, содержащими обычный текст. Они не зависимы от типа используемого компьютера. В то же время для различных платформ символами окончания строки могут быть как символ CR, так и символ LF. Интерпретатор системы MATLAB допускает любые комбинации.
- MAT-файлы являются двоичными файлами и зависят от типа используемого компьютера. Тем не менее, они могут переноситься с одного типа компьютера на другой, поскольку содержат признак используемого компьютера в заголовке файла. Система MATLAB проверяет этот признак, когда загружает файл и, если оказывается, что файл создан на компьютере другой платформы, выполняет необходимое преобразование.

Чтобы использовать MATLAB на компьютерах различных платформ, необходимы программы обмена данными для двоичного и ASCII-формата. При использовании этих программ надо быть уверенными, что MAT-файлы передаются как двоичные файлы, М-файлы - как ASCII-файлы. Ошибка в установке соответствующих режимов обычно разрушает данные.

Команда diary

Эта команда позволяет сформировать дневник сеанса работы, включая графический вывод. Дневник записывается в специальный файл на жестком диске. После сеанса работы этот файл можно просмотреть с помощью любого текстового редактора. Например, чтобы создать в текущем каталоге файл дневника с именем **febr01.out** следует использовать команду **diary febr01.out.**

Для того чтобы в процессе ведения дневника прервать запись, достаточно воспользоваться командой **diary off**, а для возобновления командой **diary on**.

M-файл Startup

Файл **matlabrc.m**, который размещен в каталоге **local**, зарезервирован для использования программистами фирмы **MathWorks**, а на многопользовательских системах для использования менеджером системы.

Файл **startup.m** предназначен для пользователя. В нем можно установить, задаваемые по умолчанию пути доступа, дескриптоы графики, а также переменные рабочей области. Например, в файл **startup.m** можно ввести строку, которая добавит каталог **/home/me/mytools** к установленному по умолчанию списку путей доступа **addpath /home/me/mytools**.

1.6 Использование памяти

Система MATLAB требует для хранения каждой матрицы непрерывной области памяти. В частности, образы и анимация могут потреблять очень большие объемы памяти. В дополнение к памяти для хранения матрицы, карта пикселей, используемая для образов, требует памяти, пропорциональной площади изображения. Так например, изображение 500*500 цветных пикселей требут 2 МБ оперативной памяти. Если требуется 10 изображений такого размера, то уже необходимо 20 МБ, что является очень большим объемом. Чтобы уменьшить объем памяти, требуемый для этих операций, надо ограничить размер выводимых изображений.

Разрешение проблем выделения памяти. Если отсутствует фрагмент памяти, достаточный для размещения матрицы, то возникает ошибка **out of memory**, хотя общий объем свободной памяти может быть большим. Это связано с фрагментированием памяти в процессе ее выделения. Чтобы ликвидировать фрагментацию, следует восполь-

зоваться командой **pack;** другой способ - разместить массивы больших размеров в оперативной памяти заранее в начале сеанса работы.

Управление динамической памятью. Система MATLAB использует для выделения динамической памяти стандартные функции **malloc** и **free** языка **C**. Эти утилиты поддерживают пул памяти, которая распределяется операционной системой в относительно медленном темпе; в свою очередь, для системы MATLAB эта память выделяется намного быстрее. Если пул недостаточен, то утилит **malloc** запрашивает операционную систему относительно выделения другого фрагмента оперативной памяти, чтобы пополнить пул. По мере выделения памяти пул может становиться очень большим. Чтобы поддержать быстродействие, утилиты **malloc** и **free** не возвращают использованную память операционной системе. Эти подпрограммы исходят из предположения, что если большой объем памяти потребовался один раз, то в нем возникнет необходимость снова. Побочный эффект этого алгоритма состоит в том, что если MATLAB использовал некоторый объем памяти один раз, то она более не доступна другим программам, даже если MATLAB не использует это. Память пула возвращается операционной системе только по завершении работы системы MATLAB.

1.7 Интерактивный доступ к справочной информации и документации. Команды Help, lookfor. Меню Help

Существуют следующие способы получить информацию о функциях системы MATLAB в процессе сеанса работы:

- команда **help**;
- команда **lookfor**;
- меню **Help;**
- просмотр и вывод на печать страниц документации;
- обращение к **Web-**серверу фирмы The MathWorks.

Команда Help

Основной и наиболее быстрый способ выяснить синтаксис и особенности применения М-функции - это использовать команду **help** <имя М-функции>. Соответствующая информация появляется непо-

средственно в командном окне. Например, команда **help magic** выведет в командное окно следующую информацию на английском языке:

help magic

MAGIC Magic square.

MAGIC(N) is an N-by-N matrix constructed from the integers

1 through N^2 with equal row, column, and diagonal sums.

Produces valid magic squares for N = 1,3,4,5,...

или

MAGIC Магический квадрат.

MAGIC(N) - это матрица размера N?N, построенная из целых чисел

от 1 до N^2 так, что суммы элементов по строкам, столбцам

и диагоналям совпадают.

Формирует правильные магические квадраты для N = 1,3,4,5,...

Следует обратить внимание, что текст интерактивной справки использует верхний регистр написания имен функций и переменных, чтобы выделить их из остальной части текста. Однако при вводе имен функций в командной строке всегда используются символы нижнего регистра, а поскольку система MATLAB чувствительна к выбору регистра, а действительные имена функций записываются строчными буквами.

Все функции системы MATLAB, а их более 800, организованы в логические группы, и структура каталогов основана на этой организации. Например, все функции линейной алгебры находятся в каталоге **matfun.** Можно распечатать все функции этого каталога с короткими пояснениями, если использовать команду **help matfun**:

help matfun

Matrix functions - numerical linear algebra.

Matrix analysis.

norm - Matrix or vector norm.

normest - Estimate the matrix 2-norm.

Linear equations.

\ and / - Linear equation solution; use "help slash".

inv - Matrix inverse.

Eigenvalues and singular values.
eig - Eigenvalues and eigenvectors.
svd - Singular value decomposition.
Matrix functions.
expm - Matrix exponential.
logm - Matrix logarithm.
Factorization utilities
qrdelete - Delete column from QR factorization.
qrinsert - Insert column in QR factorization.

Команда **help** сама по себе выводит на экран список каталогов **help** :

HELP topics:
matlab\general - General purpose commands.
matlab\ops - Operators and special characters.
matlab\lang - Programming language constructs.
matlab\elmat - Elementary matrices and matrix manipulation.

Команда lookfor

Эта команда позволяет выполнить поиск М-функции по ключевому слову; при этом анализируется первая строка комментария, и она же выводится на экран, если в ней встретилось ключевое слово. Например, в системе MATLAB нет М-функции с именем **inverse** и поэтому на команду **help inverse** ответом будет - **inverse.m not found**. Однако команда **lookfor inverse** найдет не менее дюжины совпадений, и это будет зависеть от того, какие ППП подключены к системе MATLAB.

lookfor inverse

INVHILB Inverse Hilbert matrix.
ACOS Inverse cosine.
ACOSH Inverse hyperbolic cosine.
IFFTN N-dimensional inverse discrete Fourier transform.
IPERMUTE Inverse permute array dimensions.
ICCEPS Inverse complex cepstrum.
IDCT Inverse discrete cosine transform.
idctold.m: %IDCT Inverse discrete cosine transform.

Добавление опции **-all** команде **lookfor в** форме **lookfor** шаблон **-all** позволяет просматривать все строки комментария к М-функции, а не только первую строку.

Меню Help

Это меню командного окна системы MATLAB позволяет активизировать следующие окна:

Help Window	Окно справки
Help Tips	Окно справки для получения подсказки
Help Desk (HTML)	Доступ к справочным системам на жестком диске или CD-ROM
Examples and Demos	Окно демонстрационной подсистемы
About MATLAB	Инфрмация об установленной версии
Subscibe (HTML)	Подписка на услуги фирмы **The MathWorks, Inc**.

Окно справки Help Window

Это окно может быть вызвано несколькими способами: как опция **Help Window** меню **Help**, нажатием кнопки ? инструментальной панели, либо с помощью команды **helpwin** (рисунок 1.4). На этом рисунке показано начальное окно MATLAB **Help topics**; выделяя любую из строк списка и дважды щелкая левой кнопкой мыши, можно переходить к спискам соответствующих разделов. Такое действие аналогично команде **helpwin** <имя раздела>. Ниспадающий список **See also** в этом случае не активен. Кнопки **Back, Forward**, **Home** позволяют переходить от одного активизированного окна к другому, либо возвратиться к начальному окну.

В правом верхнем углу окна расположены 3 кнопки **Go to Help Desk, Tips, Close**.

Кнопка **Tips** выводит окно подсказок (рисунок 1.5). В этом окне ниспадающее меню активизировано и позволяет выполнить ряд

дополнительных справочных команд **More help info help (HTML), lokfor, which, demo, general**.

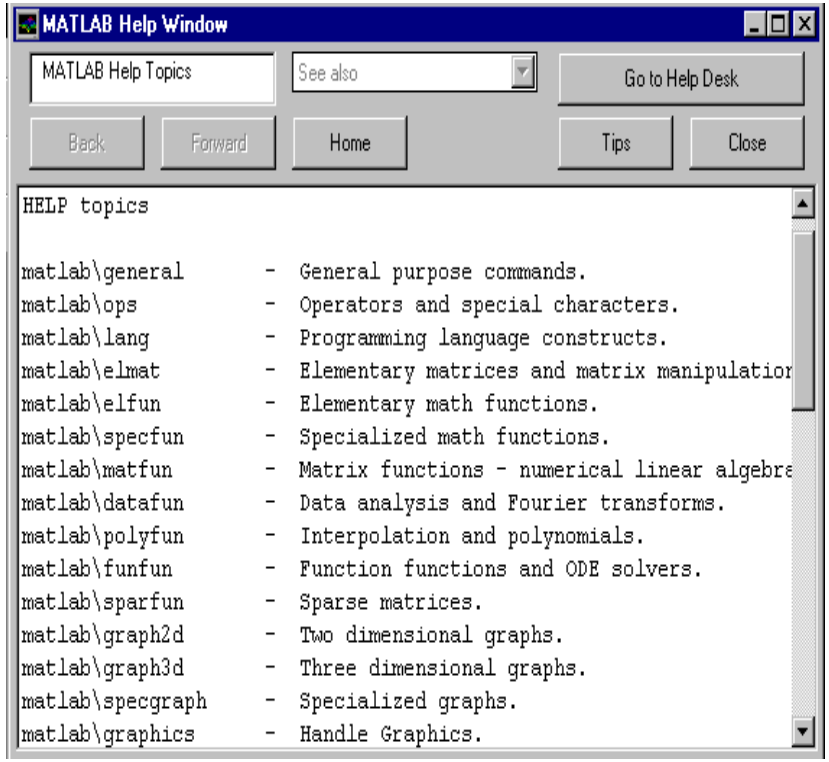


Рисунок 1.4

Опция Help Desk. Эта опция позволяет получить доступ к большому объему справочной информации и к документации по системе, размещаемой либо на жестких дисках, либо на диске CD-ROM в рамках используемого персонального компьютера. Многие из документов используют язык гипертекстовых ссылок **HTML (HyperText Markup Language)** и доступны для просмотра с помощью таких средств как **Netscape Navigator** или **Microsoft Explorer**. Эта опция может быть также инициирована с помощью команды **helpdesk**. Все

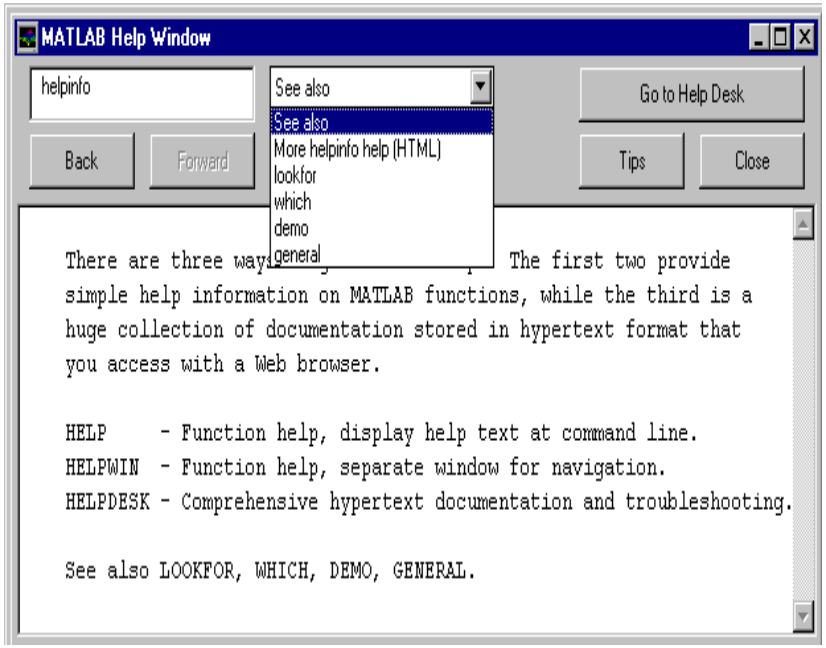


Рисунок 1.5

операторы и функции системы MATLAB описаны в формате **HTML** и содержат больше подробностей и примеров, чем справки по команде **help.** Доступны **HTML** -версии разных документов, включая описания, руководства пользователя по системе и пакетам прикладных программ. Реализованная поисковая система позволяет выполнить необходимые команды.

Опция About MATLAB. Эта опция выводит на экран заставку системы с указанием версии и принадлежности пользователю (рисунок 1.6).

Просмотр и распечатка документации. Версии справочной документации доступны для просмотра и распечатки в формате **PDF (Portable Document Format)** с помощью средства **Adobe's Acrobat**. Оно позволяет просматривать текст в формате печатной страницы, с полным набором шрифтов, графики и изображений, с полным ощущением чтения книги. Одновременно это и наилучший способ получения копий нужных страниц.

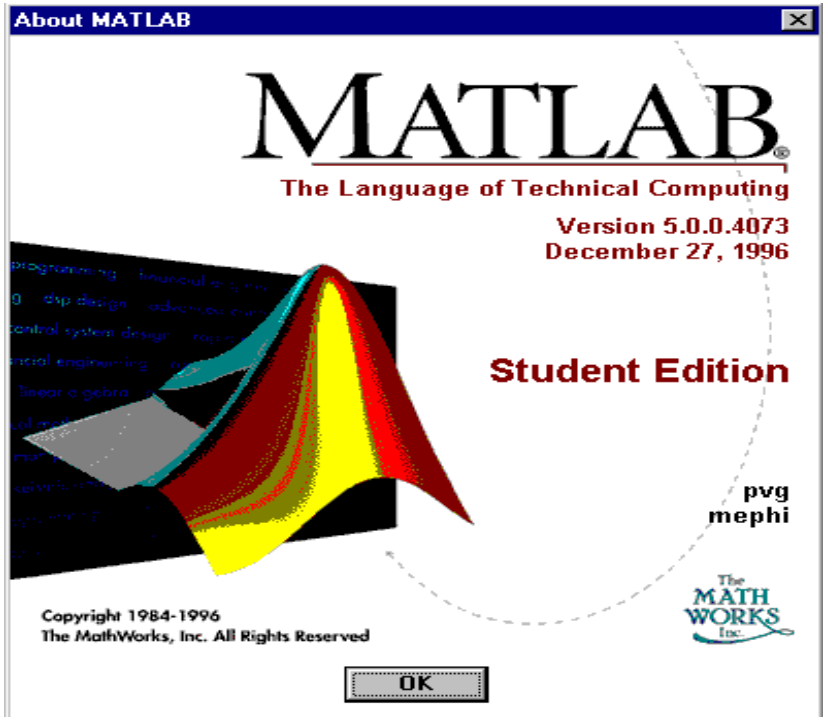


Рисунок 1.6

Web-сервер фирмы MathWorks. Если Ваш компьютер подсоединен к сети Интернет, то можно реализовать через механизм **Help Desk** соединение с WWW-сервером фирмы-производителя и выйти на страницу системы MATLAB. Можно также воспользоваться электронной почтой, чтобы задать вопросы, внести предложения или сообщить об обнаруженных ошибках. Можно воспользоваться поисковой системой WWW-сервера, чтобы сделать запрос к постоянно обновляемой базе данных технической поддержки. Возможности WWW-сервера постоянно и расширяются и вы можете более подробно ознакомиться с его информационными возможностями, непосредственно связавшись с его зеркалом в Европе по адресу www.europe.mathworks.com .

2 Инструментальное средство Notebook

В состав программного обеспечения системы MATLAB включено инструментальное средство **Notebook** (ИС **Notebook**) для создания записных книжок, содержимое которых может быть прочитано, вычислено и модифицировано в среде редактора **Microsoft Word**.

Понятие М-книги. Документ, созданный в среде ИС **Notebook**, называется М-книгой. Она включает текст, команды системы MATLAB и результаты их выполнения. Ее можно представлять себе либо как запись результатов интерактивного сеанса работы, сопровождаемую текстом, либо как документ, в который помещены исполняемые команды системы MATLAB и их результаты.

Шаблон М-книги. При создании или редактировании М-книги редактор **Word** использует специальный шаблон **M-book**. Этот шаблон позволяет получить доступ к системе MATLAB из документа редактора **Word** и управлять его форматированием. Когда создается или открывается М-книга, шаблон:

- запускает систему MATLAB, если она не была стартована, и поддерживает динамический обмен данными между **Word** и MATLAB на основе **DDE** (**Dynamic Data Exchange**) интерфейса;
- содержит макрокоманды, которые позволяют системе MATLAB обрабатывать специальные типы ячеек, в которые записываются команды и операторы языка MATLAB и результаты их исполнения;
- поддерживает в редакторе **Word** меню **Notebook**;
- поддерживает стили для текста и ячеек.

2.1 Работа в среде ИС Notebook

Для того чтобы начать работу с ИС **Notebook**, необходимо прежде всего стартовать редактор **Word**. Следующий шаг состоит в том, чтобы либо открыть новую М-книгу, либо продолжить редактирование существующей.

Создание новой М-книги. Для этого следует выбрать опцию **New** из меню **File** редактора **Word**; затем в этом диалоговом окне выбрать шаблон М-**book.dot**. Редактор **Word** создает новый документ,

используя шаблон M-**book**, добавляет меню **Notebook** и стартует систему MATLAB.

Редактирование M-книги. Для того чтобы приступить к редактированию уже существующей M-книги, надо выполнить одно из следующих действий:

- стартовать редактор **Word** и открыть существующую M-книгу, используя опцию Open из меню **File;**
- стартовать редактор **Word** и выбрать открыть M-книгу из списка последних редактированных документов из нижней части ниспадающего меню **File**;
- дважды щелкнуть на документ M-book.

Редактор **Word** открывает документ, используя шаблон **M-book**, стартует систему MATLAB, если она не была активной и добавляет меню **Notebook**.

Преобразование документа Word в M-книгу. Для того чтобы конвертировать созданный ранее в редакторе **Word** документ в M-книгу, необходимо выполнить следующие шаги:

1. Открыть новую M-книгу.
2. Выбрать опцию **File** из меню **Insert**.
3. Выбрать файл, который должен быть преобразован.
4. Нажать клавишу **Enter**.

2.2 Написание M-книги

Написание M-книги связано с вводом текста, а также операторов и команд системы MATLAB.

Ввод текста. Эта операция аналогична вводу текста в произвольный документ, создаваемый в редакторе **Word.** Используя различные стили, можно управлять шрифтами и другими атрибутами представления текста; однако следует иметь в виду, что по умолчанию для текста принят стиль **Normal**.

Ввод операторов и команд системы MATLAB. Для записи команд и операторов системы MATLAB используются специальные ячейки ввода, которые либо включены в текст, либо состоят из одной или нескольких командных строк. Для создания входной ячейки следует:

1. Ввести команду в виде текста и не нажимая клавиши **Enter** оставить курсор в конце текста.
2. Выбрать команду **Define Input Cell** из меню **Notebook,** либо использовать комбинацию клавиш **Alt-D**.

ИС **Notebook** определяет команду как ячейку ввода, помещая ее в специальные скобки и форматируя специальным стилем. Все входные ячейки отмечаются жирными скобками серого цвета, которые существенно отличаются от скобок, используемых для обозначения матриц размером и шириной; для изображения символов - жирный шрифт темно-зеленого цвета.

Исполнение команд. Для того чтобы выполнить команду системы MATLAB, ранее не определенную в качестве входной ячейки, необходимо:

1. Ввести команду как текст, оставить курсор на этой строке в конце текста и не нажимать клавишу **Enter**.
2. Ввести команду в виде текста и не нажимая клавиши **Enter** оставить курсор в конце текста. Выбрать команду **Evaluate Cell** из меню **Notebook**, либо использовать комбинацию клавиш **Ctrl-Enter**.

ИС **Notebook** кроме ячеек ввода использует также ячейки вывода, чтобы сохранить вычисленные результаты. Ячейки вывода следуют непосредственно за ячейками ввода и помечаются специальными скобками; для вывода чисел и текста используются символы синего цвета; сообщения об ошибках выводятся символами красного цвета.

Пример.

1. Напечатаем в строке команду системы MATLAB: **a = magic(3)**.
2. Используем команду **Evaluate Cell** из меню **Notebook** или комбинацию клавиш **Ctrl-Enter**.

ИС **Notebook** отобразит команду как ячейку ввода и выведет результат в ячейку вывода:

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

Многострочные ячейки ввода. Если вводятся команды MATLAB, которые занимают несколько строк, то необходимо обязательно выделить все строки, чтобы либо определить их как ячейки ввода, либо вычислить их.

Автоматическая инициализация команд. Для автоматической инициализации команд при открытии М-книги необходимо определить команды как ячейки автостарта (**autoinit cells**). Это наиболее быстрый и простой способ формирования рабочей области. Ячейки

автостарта - это те же ячейки ввода со следующими дополнительными свойствами:

- ИС **Notebook** вычисляет ячейки автостарта при открытии М-книги;
- команды в ячейках автостарта изображаются символами темно-синего цвета.

Создание ячеек автостарта. Создать ячейки автостарта можно двумя способами:

1. Ввести команды в виде текста, а затем определить их как ячейку автостарта, используя команду **Define AutoInit Cell**.
2. Если команды определены как ячейка ввода, то ее можно конвертировать в ячейку автовызова, используя команду **Define AutoInit Cell** .

Пример.

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

Принудительное вычисление ячейки автостарта. Для принудительного вычисления ячейки автостарта следует:

1. Позиционировать курсор в ячейке автостарта.
2. Использовать команду **Evaluate Cell** или сочетание клавиш **Ctrl-Enter**.

2.3 Объединение команд в группы

ИС **Notebook** позволяет вводить последовательность команд системы MATLAB и работать с ней как с группой ячеек ввода. Группа ячеек - это многострочная ячейка ввода или автостарта, которая включает более одной команды.

В группу ячеек нельзя включать текст или ячейки вывода. Результаты вычисления группы ячеек хранятся в одной ячейке вывода, которая располагается непосредственно за группой ячеек ввода.

При создании группы ячеек ИС **Notebook** определяет ее как ячейку ввода, если только первая строка не является ячейкой автостарта; в последнем случае вся группа объявляется ячейкой автостарта.

Если группа ячеек включает команды, которые производят текстовый или числовой выход, а также команды графического вывода, то числовые данные и текст всегда выводятся первыми, независимо от действительной последовательности команд в группе.

Группы ячеек чрезвычайно полезны, когда несколько команд полностью задают графический образ. Если это сделано именно так, то формируется единственный график, который отражает все свойства, заданные в командах. Если же команда построения графика оформлена в виде отдельной ячейки, то вычисляемые ячейки генерируют множество графиков.

Создание группы ячеек. Для создания группы ячеек необходимо:

1. Выделить ячейки ввода, которые предполагается объединить в группу: если в составе выделенных оказались ячейки вывода, то они удаляются; если выделенный фрагмент включает текст, то он размещается после группы, за исключением того случая, когда текст предшествует первой ячейке ввода; если оказалась выделенной часть или вся ячейка вывода и не затронута ячейка ввода, то в группу включается соответствующая входная ячейка.
2. Применить команду **Group Cells** или комбинацию клавиш **Alt-G**.

ИС **Notebook** преобразует выделенные ячеки в общую группу и заменяет маркеры ячеек единственной парой скобок.

Вычисление группы ячеек. Для вычисления группы ячеек необходимо выполнить те же операции, что и для выполнения ячейки ввода, а именно:

- Позиционировать курсор в любом месте группы ячеек или в ячейке вывода.
- Использовать команду **Evaluate Cell** или комбинацию клавиш **Ctrl-Enter**.

При вычислении группы ячеек результат размещается в единственной ячейке вывода. По умолчанию, ячейка вывода располагается сразу за группой ячеек, как только начинаются вычисления. Если вычисления выполняются при наличии ячейки вывода, то результат помещается в нее.

Пример. Следующий пример показывает шаги, связанные с преобразованием ячеек ввода в группу ячеек и последующего ее вычисления.

Записать ячейки ввода:

```
t = 0:pi/10:2*pi;
[X, Y, Z] = cylinder(4*cos(t)+1);
mesh(X, Y, Z)
```

Выделить ячейки:

```
t = 0:pi/10:2*pi;
[X, Y, Z] = cylinder(4*cos(t)+1);
mesh(X, Y, Z)
```

Применить команду **Group Cells**, чтобы создать группу ячеек

```
t = 0:pi/5:2*pi;
[X, Y, Z] = cylinder(4*cos(t)+1);
mesh(X, Y, Z)
```

Чтобы вычислить группу ячеек, воспользуйтесь командой **Evaluate Cell** или комбинацией клавиш **Ctrl-Enter**. На экран будет выведен рисунок 2.1.

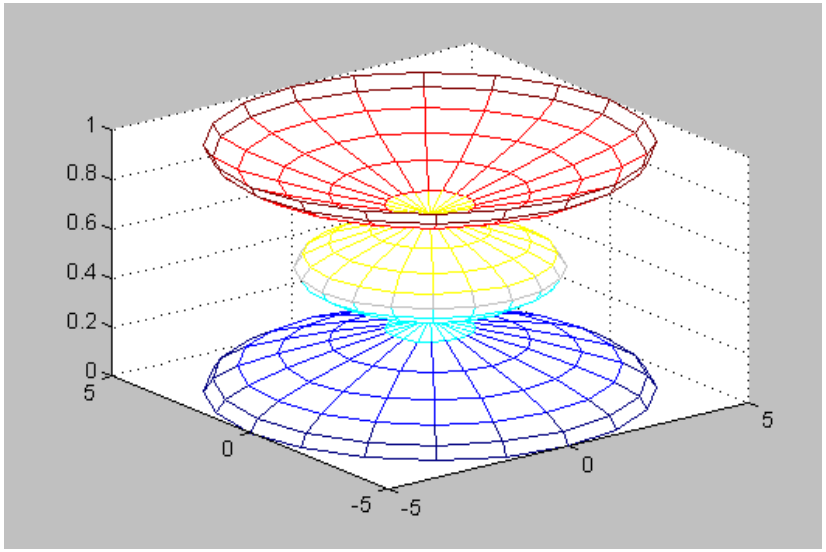


Рисунок 2.1

2.4 Использование операторов системы MATLAB внутри текста. Зоны вычислений. Преобразование ячейки в текст

ИС **Notebook** позволяет помещать команды и операторы MATLAB непосредственно в текст строки или параграфа. Для этого следует выполнить следующую последовательность операций:

1. Напечатать текст вместе с оператором или командой.
2. Выделить в тексте команду или оператор.
3. Применить команду **Evaluate Cell** или комбинацию клавиш **Ctrl-Enter**.

Рассмотрим эту возможность на следующем примере:

Команда **z = magic(3)** генерирует магический квадрат размера 3 x 3

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

Как показывает этот пример, ИС **Notebook** преобразует команду в ячейку ввода, посылает на исполнение в систему MATLAB и выводит результаты в ячейке вывода.

Зоны вычислений

Существует возможность разбивать М-книгу на автономные секции, называемые зонами вычислений. Зона вычислений - это непрерывный блок, который включает текст, ячейки ввода и вывода, связанные с описанием некоторой операции или проблемы. ИС **Notebook** определяет зону вычислений как секцию документа и помечает ее начало и конец, за исключением начала и конца документа. М-книга может содержать любое количество таких зон.

Преподаватели могут использовать эту возможность, чтобы подготовить, например, множество контрольных вопросов или задач. Сопоставляя каждой задаче зону вычислений, можно создавать и отлаживать их автономно. Стоит отметить, что переменные отдельной зоны не являются локальными переменными только этой зоны.

Задание зон вычислений. Как только написан текст и созданы ячейки ввода, можно определить зону вычислений, используя следующие шаги:

1. Выделить ячейки и текст, включаемые в зону вычислений.
2. Применить команду **Define Calc Zone**.

Если в документы уже существуют как ячейки ввода, так и ячейки вывода, то при определении зоны вычислений должны быть выделены и те и другие.

Вычисление зон. Для того чтобы исполнить команды зоны вычислений, необходимо:

1. Позиционировать курсор в зоне вычислений.

2. Применить команду **Evaluate Calc Zone** или комбинацию клавиш **Alt-Enter**.

ИС **Notebook** посылает каждую ячейку ввода из зоны вычислений в систему MATLAB для исполнения. По умолчанию, **Notebook** размещает ячейку вывода сразу после зоны вычислений. Если зона вычислений включает ячейку вывода, то результат помещается в эту ячейку, где бы ни было ее расположение в создаваемой М-книге.

Преобразование ячейки в текст

Для того чтобы преобразовать ячейку ввода (ячейку автовызова или группу ячеек) в текс, необходимо:

1. Позиционировать курсор в любом месте ячейки.
2. Применить команду **Undefine Cells** или комбинацию клавиш **Alt-U**.

Notebook преобразует ячейку в текст, применяя стиль, при этом ячейка вывода во внимание не принимается.

Вынесение окна MATLAB на передний план. Для размещения окна MATLAB на переднем плане следует использовать команду **Bring MATLAB to Front**.

Поддержание целостности рабочей области. Когда в одном сеансе работы с редактором **Word** обрабатывается более одной М-книги, выполняются следующие условия:

- все М-книги используют одну и ту же копию системы MATLAB (один процесс);
- все М-книги используют одну и ту же рабочую область.

Если несколько одинаковых имен для переменных используются в нескольких М-книгах, то возможно их взаимное влияние. Чтобы обеспечить целостность рабочей области для каждой М-книги, надо в первой ячейке автовызова для каждой М-книги определить команду **clear**.

2.5 Вычисление ячеек

М-книгу можно рассматривать как дневник сеанса работы с системой MATLAB, при этом в М-книге аккуратно отслеживаются все связи между использованными операторами. Однако если приходится изменять или удалять ячейку ввода в процессе написания М-книги, надо помнить, что **Notebook** не выполняет автоматического перевычисления ячеек, которые могут оказаться зависимыми от вне-

сенных изменений. В результате может оказаться нарушенной непротиворечивость данных.

При работе над книгой целесообразно периодически применять команду **Evaluate M-book**, чтобы гарантировать непротиворечивость используемых данных. Можно применять механизм зон вычислений, чтобы объединить связанные команды в отдельную секцию М-книги, а затем применить команду **Evaluate Calc Zone**, чтобы исполнить их.

В этом разделе приведены особенности вычисления отдельных ячеек, последовательности ячеек, вычислений в цикле и вычисления М-книг.

Вычисление ячеек ввода, ячеек автовызова и групп ячеек. Для того чтобы вычислить такие ячейки, необходимо:

1. Позиционировать курсор в любом месте ячейки ввода или соответствующей ячейке вывода.
2. Применить команду **Evaluate Cell** или комбинацию клавиш **Ctrl-Enter**.

Если ячейки вывода отсутствуют, то **Notebook** размещает их сразу после ячейки ввода; если ячейки вывода уже созданы, то новые результаты размещаются в них, где бы в книге они не находились.

Вычисление последовательности ячеек ввода. Для вычисления более чем одной команды или оператора системы MATLAB, размещенных в разных, но непрерывно следующих одна за другой ячеек ввода, необходимо:

1. Выделить последовательность ячеек и текст, который включает ячейки ввода.
2. Применить команду **Evaluate Cell** или комбинацию клавиш **Ctrl-Enter**.

Notebook вычисляет каждую ячейку ввода в выделенном фрагменте, создавая, если необходимо, ячейки вывода или размещая результат в существующих ячейках.

Вычисление М-книги в целом. Для этого следует использовать команду **Evaluate M-book** или комбинацию клавиш **Alt-R**. ИС **Notebook** начинает вычисление М-книги с самого начала, независимо от места расположения курсора, и вычисляет каждую ячейку. По мере вычисления **Notebook** включает новые ячейки вывода или размещает результаты в существующих ячейках.

Контроль вычисления ячеек. Для контроля результатов исполнения ячеек при вычислении М-книги рекомендуется использовать опцию **Stop evaluating on error**. Если опция включена, то при возникновении ошибки, дальнейшее вычисление прекращается; если нет, то вычисления выполняются полностью, независимо от имеющихся ошибок.

Вычисление зон. Для вычисления зон необходимо:

1. Позиционировать курсор в любом месте зоны.
2. Применить команду **Evaluate Calc Zone** или комбинацию клавиш **Alt-Enter**.

Notebook вычисляет зону, независимо от места расположения курсора, и создает, если необходимо, ячейки вывода или размещает результат в существующих ячейках.

Вычисление команд в цикле. Для того чтобы вычислить последовательность команд повторно, необходимо:

1. Выделить ячейки ввода, включая текст и ячейки вывода, размещенные между ними.
2. Применить команду **Evaluate Loop** или комбинацию клавиш **Alt-L**.

ИС **Notebook** выведет на экран следующую диалоговую панель (Рисунок 2.2):

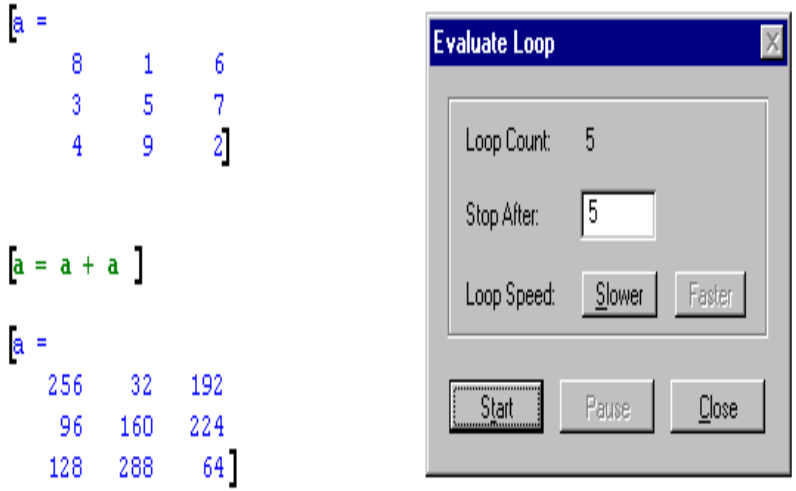


Рисунок 2.2

С помощью этой диалоговой панели можно реализовать следующие операции:

- задать в поле **Stop After** количество циклов вычисления команд или операторов;
- запускать вычисления, используя кнопку **Start**, которая при этом инвертируется в кнопку **Stop**;

- приостанавливать выполнение команд с помощью кнопки **Pause**, которая при этом преобразуется в кнопку **Continue**;
- изменять скорость вычислений с помощью кнопок **Faster и Slower**; прерывать выполнение цикла, используя кнопку **Stop**.

Пример. Для матрицы

```
a = magic(3)

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

выполнить слеующий оператор 3 раза

```
a = a+a
     a =
         64     8    48
          4    40    56
         32    72    16
```

2.6 Операции с результатами вычислений

В этом разделе обсуждаются возможные операции с результатами вычислений, выполняемыми в М-книге, в том числе операции над ячейками вывода, вывод числовой и графической информации, а также печать результатов.

Операции с ячейками вывода. Если вычисляется ячейка ввода, для которой не создано ячейки вывода, ИС **Notebook** размещает ячейку вывода сразу за ячейкой ввода; если ячейка вывода уже существует, то результат помещается в нее, где бы она не располагалась в М-книге.

Если вычисляется группа ячеек, то результат следует сразу за этой группой и занимает одну ячейку вывода.

Вывод результатов вычислений на терминал. При выводе результатов вычислений на терминал с помощью команды **Notebook Options** можно управлять форматом вывода числовых данных и размерами графического окна.

Преобразование ячеек вывода в текст. Для преобразования ячеек вывода в текст применяется команда **Undefine Cells**. Если выходом являются числовые данные или текст, то **Notebook** удаляет маркеры ячейки и преобразует содержимое в текстовый формат, используя стиль **Normal**. Если выходом является графика, то **Notebook**

удаляет маркеры ячейки вывода, сохраняя ее содержимое. При этом команда никак не действует на соответствующую ячейку ввода.

Для того чтобы преобразовать ячейку вывода в формат текста книги, необходимо выполнить следующие действия:

1. Выделить преобразуемую ячейку.
2. Применить команду **Undefine Cells** или комбинацию клавиш **Alt-U**.

Для того чтобы удалить ячейку вывода, используйте команду **Purge Output Cells** или комбинацию клавиш **Alt-P**.

Вывод на печать М-книги. Для этого используется команда **Print** из меню **File**. Редактор **Word** соблюдает следующие правила при выводе на печать ячеек М-книги:

- маркеры ячеек не выводятся на печать;
- ячейки ввода, автовызова и вывода (включая сообщения об ошибках) печатаются в соответствии с выбранными для них стилями. Если требуется черно-белая печать вместо цветной и оттенков серого, то следует внести соответствующие изменения в стили.

Изменение стилей в шаблоне М-книги. Шаблон М-книги, в котором предопределены стили для текста и ячеек, использует стиль **Normal** редактора **Word** для печати текста и стили, задаваемые ИС **Notebook**, для распечатки ячеек. Так если М-книга распечатывается на
цветном принтере, то ячейки ввода печатаются темно-зеленым цветом, ячейки вывода - синим, а ячейки автовызова - темно-синим цветом; сообщения об ошибках - красным цветом. Если М-книга распечатывается на черно-белом принтере, то эти цвета преобразуются в оттенки серого. Если требуется только черный цвет, то необходимо изменить цвета стилях **Input**, **Output, AutoInit** и **Error** .

Следующая таблица описывает стили, принятые по умолчанию в ИС **Notebook**. Если требуется изменить стиль, то эта таблица поможет в дальнейшем вернуться к первоначальным стилям.

Стиль	***Шрифт***	***Размер пункты***	***Выделение***	***Цвет***
Normal	**Times New Roman**	10		Черный
AutoInit	**Courier New**	10	**Bold**	Темно-синий
Error	**Courier New**	10	**Bold**	Красный
Input	**Courier**	10	**Bold**	Темно-

	New			зеленый
Output	**Courier New**	10		Синий

Предупреждение. Если вы изменяете стиль, то редактор Word применяет его ко всем символам М-книги, которые используют его, и предлагает опцию изменить шаблон. Будьте внимательны, внося изменения в шаблон, поскольку это означает, что эти изменения отразятся на всех вновь создаваемых М-книгах.

2.7 Управление форматом вывода чисел

Управление форматом вывода чисел реализуется с помощью опции **Numeric Format** диалогового окна **Notebook Options** (на рисунок 2.3 показана установка формата чисел). Для доступа к диалоговому окну необходимо

Рисунок 2.3

применить команду **Notebook Options**.

Опция **Numeric Format** реализована в виде ниспадающего списка, который соответствует всем возможностям команды **format** системы MATLAB. На рисунке 2.4 показан список доступных форматов.

Кроме того, можно управлять использованием пробела между ячейками ввода и ячейками вывода: формат **Loose** добавляет пробел между ячейками, а формат **Compact** - нет.

После выбора всех опций формата надо подтвердить установки нажатием клавиши **Enter**. Эти установки не будут задействованы до тех пор, пока не будет перевычислена соответствующая ячейка ввода.

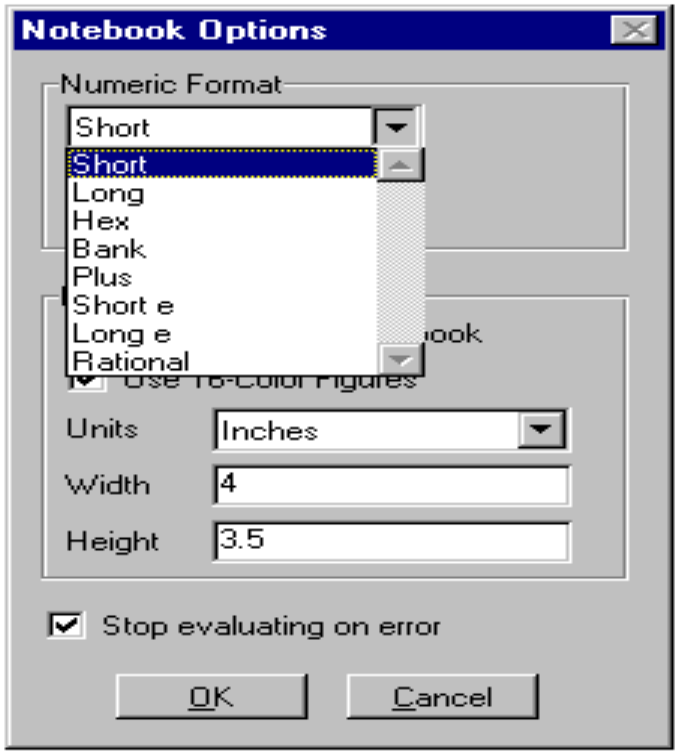


Рисунок 2.4

2.8 Управление графическим выводом

Среди опций команды **Notebook Options** присутствуют опции **Figure Options**, которые позволяют управлять включением рисунка в М-книгу, его размером, а также способом вывода на печать (палитра из 16 или 256 цветов).

Включение рисунка в текст М-книги. По умолчанию, создаваемые рисунки помещаются в М-книгу, однако можно отменить опцию **Embed Figures in M-book** и в этом случае рисунок будет виден только в графическом окне системы MATLAB (рисунок 2.5) (Надпись около флажка **«Embed Figures in M-book»** означает «Отменить флажок для отображения графики в отдельном окне»).

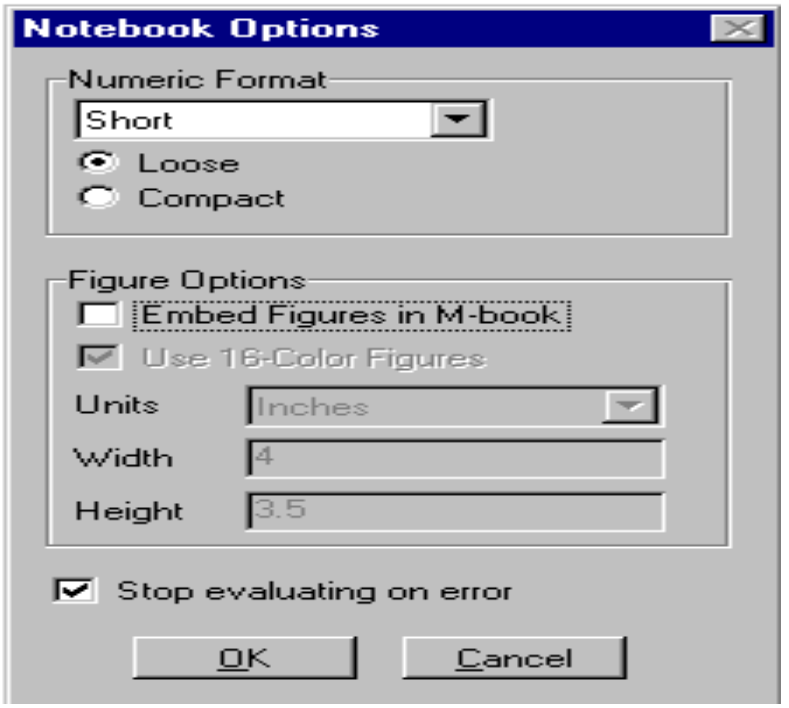


Рисунок 2.5

Рисунки, размещаемые в тексте, не включают объектов графического интерфейса пользователя **GUI**, генерируемых функциями **uicontrol** и **uimenu.** ИС **Notebook** проверяет, включать рисунок в М-

книгу или нет, проверяя свойство фигуры **Visible**; если его состояние **off,** то рисунок включается в книгу, если **on**, то рисунок отображается только в графическом окне.

Управление выводом графики. Команда **Toggle Graph Output for Cell** разрешает или подавляет вывод рисунка для данной ячейки ввода. Для ее применения надо разместить курсор в поле ячейки ввода и выбрать эту команду из меню **Notebook Options**. В этом случае в строке команды графического вывода появится уведомление (**no graph**), а вслед за ячейкой ввода появится пустая строка, чтобы обозначить, что графический вывод подавлен. Для того чтобы разрешить графический вывод, надо разместить курсор в поле ячейки ввода и вновь воспользоваться командой **Toggle Graph Output for Cell**. При этом уведомление (**no graph**) исчезнет. Эта команда подавляет опцию **Embed Figures in M-book**, если она была установлена.

Управление размером рисунка. Размер рисунка устанавливается с помощью команды **Notebook Options**; указывая в полях **Width** и **Height** ширину и высоту рисунка и нажимая клавишу **Enter**, можно зафиксировать необходимый размер поля. Эти изменения вступают в действие только после пересчета соответствующей ячейки ввода.

Можно также изменять размеры фигуры, манипулируя ее графическими дескрипторами. Выделите фигуру, нажимая на левую кнопку мыши в ее произвольном месте; **Word** выделит контур этого рисунка, который выполняет роль графического дескриптора. Перемещение углового дескриптора корректирует положение смежных сторон, а перемещение срединного - расположение самой стороны. Если после таких изменений рисунок строится заново, то он восстанавливает свои первоначальные размеры.

Удаление пустого пространства вокруг графика. Существует возможность вырезать области пустого пространства вокруг рисунка. Для этого, удерживая клавишу **SHIFT**, следует перемещать дескриптор установки размеров так, чтобы удалить ненужные области.

Добавление пространства вокруг графика. Для этого следует, удерживая клавишу **SHIFT**, перемещать дескриптор установки размеров от центра рисунка, регулируя размеры контура.

Вывод графики на печать. Если на печать выводится рисунок, который включает поверхности или многоугольники, то по умолчанию используется палитра из 16 цветов. Если требуется палитра из 256 цветов, то надо убедиться, что опция **Use 16-Color Figures** выключена (рисунок 2.6)(Надпись около флажка «Use 16-Color Figures» означает «Флажок выбора 16- или 256- цветной палитры при печати рисунка »).

Замечание. Эта опция будет применяться только к вновь генерируемому графику. Таким образом, чтобы использовать нужную палитру, надо ее сначала установить, а уже затем сгенерировать график.

2.9 Команды ИС Notebook

Ниже в порядке их следования описаны команды, представленные в меню **Notebook** редактора **Microsoft Word** и используемые ИС **Notebook** при работе с М-книгой (Рисунок 2.7).

Define Input Cell (меню Notebook)

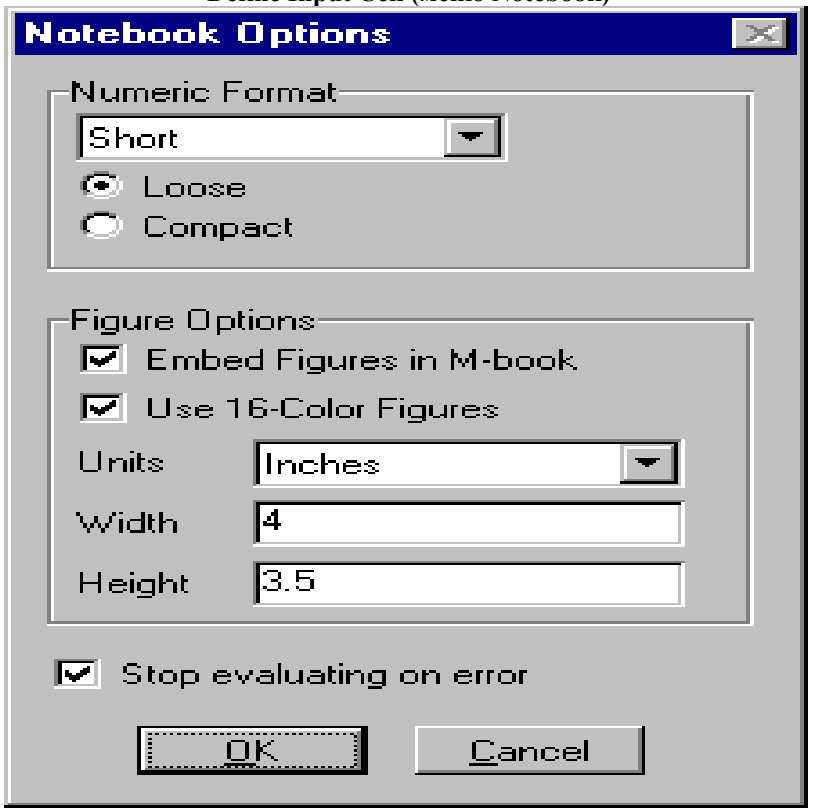


Рисунок 2.6

Команда **Define Input Cell** создает ячейку ввода, включая в нее текущий параграф, выделенный текст или ячейку автовызова. Если вы,
воспользуетесь этой командой, когда курсор находится в тексте параграфа, то весь параграф будет преобразован в ячейку ввода. Если выделен некоторый фрагмент текста, то именно он и будет преобразован в ячейку ввода. Если курсор находится в поле ячейки автовызова, то по этой команде она будет преобразована в ячейку ввода. Формат ячейки ввода использует стиль **Input**, который включает жирный шрифт **Courier New** кегля 10 темно-зеленого цвета.

Define AutoInit Cell (меню Notebook)

Команда **Define AutoInit Cell** создает ячейку автовызова, включая в нее текущий параграф, выделенный текст или ячейку ввода.

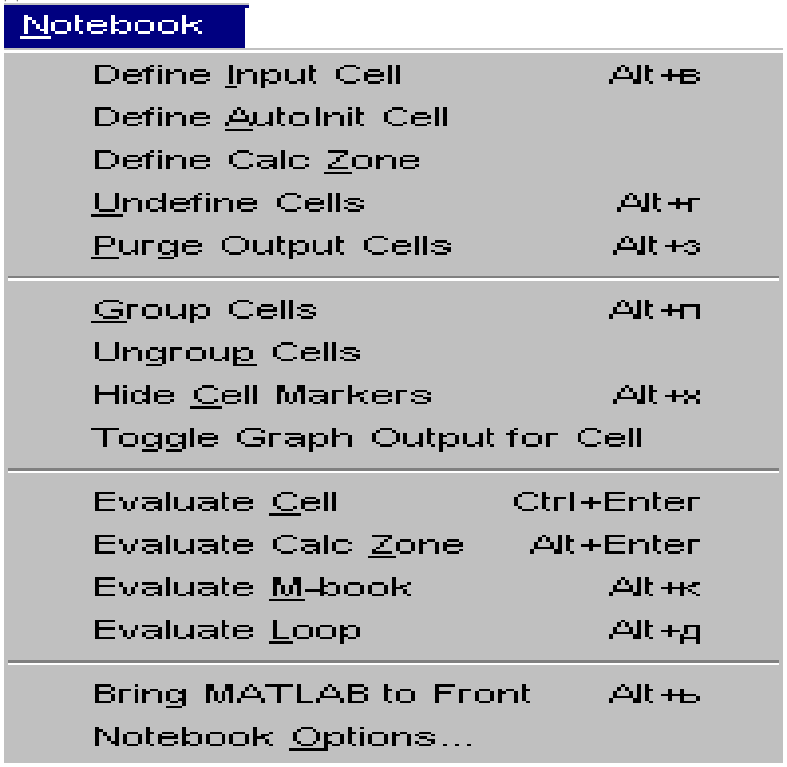


Рисунок 2.7

В дальнейшем ячейка автовызова будет вычисляться автоматически при открытии М-книги. Если вы воспользуетесь этой командой, когда курсор находится в тексте параграфа, то весь параграф будет преобразован в ячейку автовызова. Если выделен некоторый фрагмент текста, то именно он и будет преобразован в ячейку автовызова. Если курсор находится в поле ячейки ввода, то по этой команде она будет преобразована в ячейку автовызова. Формат ячейки автовызова использует стиль **AutoInit**, который включает жирный шрифт **Courier New** кегля 10 темно-синего цвета.

Define Calc Zone (меню Notebook)

Команда **Define Calc Zone** определяет выделенный текст, ячейки ввода и вывода в качестве зоны вычисления. Эта зона объединяет текст, ячейки ввода и вывода, которые вместе описывают некоторую операцию или задачу. **ИС Notebook** определяет зону вычисления как секцию документа, выделяя ее маркерами, однако маркеры не выводятся, секция располагается в начале и в конце документа.

Undefine Cells (меню Notebook)

Команда **Undefine Cells** преобразовывает выделенные ячейки в текст. Если никакие ячейки не выделены, но курсор находится в некоторой ячейке, то **Notebook** преобразует только эту ячейку, при этом удаляются маркеры ячейки, а ее формат соответствует стилю **Normal.** При преобразовании ячейки ввода автоматически преобразуется и ячейка вывода , но не наоборот. Если преобразуется ячейка вывода, содержащая графику, то графический вывод в М-книге сохраняется, но он более не будет ассоциирован с ячейкой ввода.

Purge Output Cells (меню Notebook)

Команда **Purge Output Cells** удаляет все выделенные ячейки вывода.

Group Cells(меню Notebook)

Команда **Group Cells** преобразует выделенные ячейки ввода в единственную многострочную ячейку ввода, которая называется группой ячеек. Если выделенный фрагмент включает текст, то **Notebook** помещает его после группы ячеек. Однако, если текст предшествует первой ячейке ввода в группе, то он остается на месте. Если выделенный фрагмент включает ячейки вывода, то **Notebook** удаляет их. Если перед использованием команды выделены все или часть ячеек вывода, то **Notebook** включает соответствующие ячейки ввода в группу ячеек. Если первая строка в группе ячейки является ячейкой автовызова, то вся группа ячеек воспринимается как после-

довательность ячеек автовызова; в противном случае, группа равносильна последовательности ячеек ввода.

Группу ячеек можно преобразовать в ячейку автовызова командой **Define AutoInit Cell**. Структура группы наиболее полезна, когда, например, последовательность команд или операторов формирует один график.

Вычислить группу ячеек можно с помощью команды **Evaluate Cell**, при этом все результаты вычислений будут размещаться в единственной ячейке вывода.

Ungroup Cells (меню Notebook)

Команда **Ungroup Cells** преобразовывает текущую группу ячеек в последовательность ячеек ввода или ячеек автовызова. Если группа ячеек является ячейкой ввода, то она преобразовывается в ячейку ввода; если группа ячеек - ячейка автовызова, то она преобразовывается в ячейку автовызова. При этом ячейка вывода, ассоциированная с группой ячеек, удаляется.

Группа ячеек являются текущей, если выполняются следующие условия:

- курсор находится в поле этой группы;
- курсор находится в конце строки, в которой расположен закрывающий маркер для группы ячеек;
- курсор находится в ячейке вывода, связанной с данной группой ячеек; если группа ячеек выделена.

Hide (Show) Cell Markers (меню Notebook)

Команда **Hide Cell Markers** позволяет сделать невидимыми маркеры ячеек по всей М-книге. Когда команда выбрана, она изменяется на **Show Cell Markers**. Эти команды никак не влияют на распечатку ячеек, поскольку **Notebook** вообще не печатает маркеры, независимо от того, видимы они на экране или нет.

Toggle Graph Output for Cell (меню Notebook)

Команда **Toggle Graph Output for Cell** позволяет запретить или разрешить выод графика для данной ячейки ввода. Если ячейка ввода или автовызова создает рисунок, а вы желаете подавить его вывод, то следует поместить курсор в поле ячейки ввода и выбрать эту команду. Сообщение (**no graph**) будет помещено после маркера ячейки ввода, чтобы указать, что вывод графика для той ячейки подавлен. Чтобы разрешить вывод графика для этой ячейки, следует поместить курсор в поле ячейки ввода и выбрать эту же команду еще раз. Помета (**no graph**) будет удалена. Команда **Toggle Graph Output for Cell** от-

меняет опцию **Embed Figures in M-book**, если эта опция была установлена в диалоговой панели **Notebook Options**.

Evaluate Cell(меню Notebook)

Команда **Evaluate Cell** посылает текущую ячейку ввода или группу ячеек в среду системы MATLAB для обработки. Результаты вычисления или сообщения об ошибках размещаются в ячейках вывода. Ячейка ввода содержит команду или оператор системы MATLAB. Группа ячеек - это одна многострочная ячейка ввода, которая содержит более одной команды или оператора системы MATLAB.

Когда вычисляется ячейка ввода, для которой не существует ячейки вывода, то ИС **Notebook** размещает такую ячейку сразу за ячейкой ввода; если ячейка вывода уже существует, то результаты размещаются в этой ячейке, где бы внутри М-книги она не находилась. Если вычисляется группа ячеек, то все результаты размещаются в единственной ячейке вывода.

Ячейка ввода или группа ячеек являются текущими, если выполняются следующие условия:

- курсор находится в поле этих ячеек;
- курсор находится в конце строки, в которой расположен закрывающий маркер ячейки ввода или группы ячеек;
- курсор находится в ячейке вывода, связанной с данной ячейкой ввода или группой ячеек;
- если ячейка ввода или группа ячеек выделены.

Вычисление, которая включает длинную операцию, может вызвать ситуацию, называемую блокировкой таймера. Если это случается, **Word** выводит сообщение о блокировке таймера и спрашивает, будете ли вы продолжать вычисления или завершите их в данный момент. Если вы продолжаете вычисления, то **Word** сбрасывает блокировку и начинает отсчет допустимого интервала с начала. Допустимый интевал блокировки является внутренним параметром редактора **Word** и не может быть изменен.

Evaluate Calc Zone (меню Notebook)

Команда **Evaluate Calc Zone** посылает ячейки ввода из текущей зоны вычисления в среду системы MATLAB для обработки. Текущая зона вычисления - это секция документа редактора **Word**, в которой находится курсор. Для каждой ячейки ввода ИС **Notebook** генерирует ячейку вывода. Если ячейка вывода не существует, то она создается и размещается сразу за ячейкой ввода; если существует, то результаты размещаются в этой ячейке, где бы внутри М-книги она не находилась.

Evaluate M-book (меню Notebook)

Команда **Evaluate M-book** вычисляет М-книгу целиком, посылая все ячейки ввода в среду системы MATLAB для обработки. Вычисление начинается с начала книги, независимо от текущей позиции курсора. Для каждой ячейки ввода ИС **Notebook** генерирует ячейку вывода. Если ячейка вывода не существует, то она создается и размещается сразу за ячейкой ввода; если существует, то результаты размещаются в этой ячейке, где бы внутри М-книги она не находилась.

Evaluate Loop (меню Notebook)

Команда **Evaluate Loop** позволяет организовать вычисление выделенных ячеек в цикле.

Для этого необходимо выполнить следующие шаги:

- Выделить ячейки ввода, которые должны быть вычислены несколько раз.
- Выбрать команду из меню Notebook, после чего на экране появится следующая диалоговая панель (Рисунок 2.8):

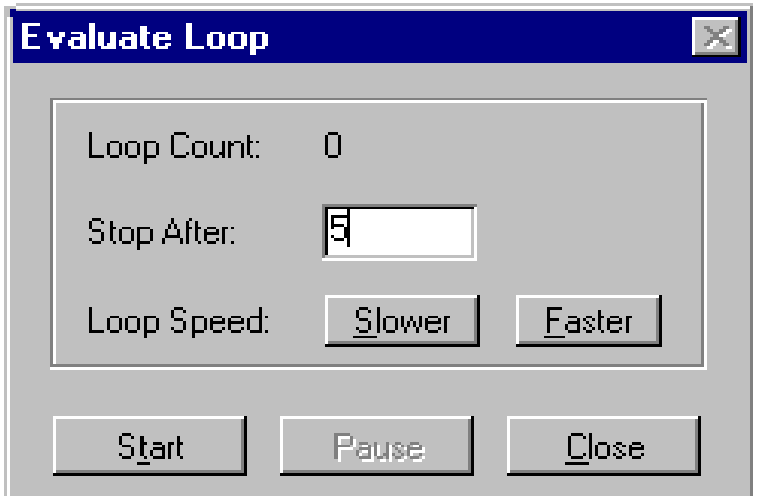


Рисунок 2.8

Указать количество циклов, которое необходимо выполнить над выделенными ячейками в поле **Stop After**, затем нажать на клавишу **Start**. При этом метка на кнопке изменяется на **Stop**. ИС

Notebook начинает исполнение команд и указывает количество завершенных итераций в поле **Loop Count**. Для увеличения задержки в конце каждой итерации, надо нажать кнопку **Slower;** для уменьшения - **Faster**.

Чтобы приостановить выполнение команд, следует нажать кнопку **Pause.** При этом метка на кнопке изменяется на **Resume**. Для продолжения вычислений нажать кнопку **Resume**. Для прекращения обработки предназначена кнопка **Stop.** Чтобы закрыть диалоговое окно **Evaluate Loop,** нажмите кнопку **Close**.

Bring MATLAB to Front (меню Notebook)

Команда **Bring MATLAB to Front** предназначена для того, чтобы на экране терминала вынести командное окно MATLAB на передний план.

Notebook Options (меню Notebook)

Команда **Notebook Options** позволяет проверить и изменить опции, связанные с выводом на экран числовых и графических результатов вычислений в М-книге. При выборе этой команды **Notebook** высвечивает на экране следующую диалоговую панель (рисунок 2.9):

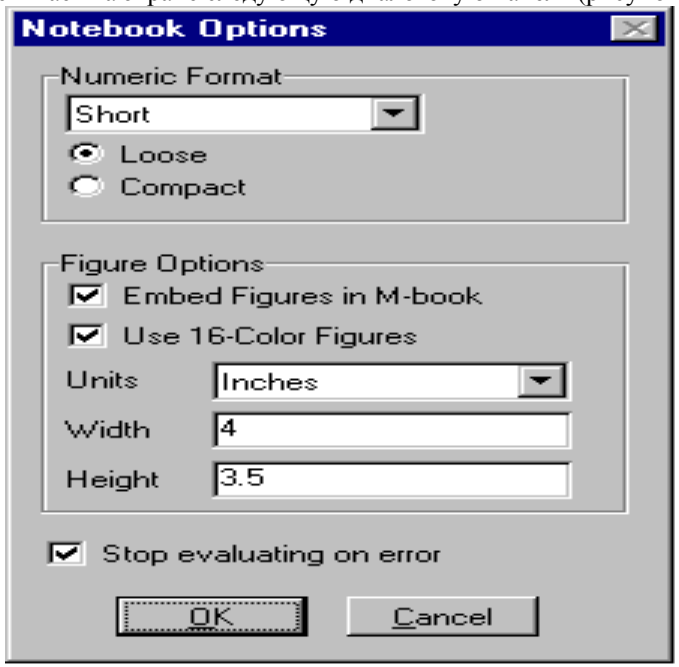


Рисунок 2.9

Управление форматом вывода чисел. Установки **Numeric Format** управляют форматом числового вывода. Можно изменять способ представления числовых данных при их выводе на терминал, выбирая опцию формата из списка. Эти установки соответствуют опциям команды **format** системы MATLAB. Кроме того, можно управлять пробелами между ячейками ввода и ячейками вывода, выбирая форматы **Loose** или **Compact.** Для подтверждения выбранных установок следует нажать кнопку **OK**. Эти установки не влияют на выведенные результаты, они действительны только тогда, когда вы вычисляете новые или перевычисляете прежние ячейки. Кнопка **Cancel** позволяет отказаться от только что выбранных установок.

Управление выводом графики. Следующие установки **Figure Options** позволяют управлять выводом графики:

- флажок **Embed Figures in M-book** управляет помещением рисунков в М-книгу. По умолчанию, флажок вывода графики в М-книгу включен. Если флажок не отмечен, то Notebook выводит графику в отдельном графическом окне. Команда **Toggle Graph Output for Cell** игнорирует опцию **Embed Figures in M-book**, если она была установлена;
- флажок **Use 16-Color Figures** управляет выбором 16-цветной или 256-цветной палитры. Если на печать выводятся цветные многоугольники или поверхности, необходимо использовать 16-цветную палитру вместо 256-цветной, применяемой по умолчанию. 16-цветная палитра обеспечивает правильную печать на цветном или черно-белом с полутонами принтерах. 256-цветная палитра выводит на печать в черном цвете, независимо от устройства вывода;
- список **Units** и поля **Width, Height** управляют размером рисунка. Единица измерения устанавливается из списка **Units**, а точные размеры по ширине и высоте определяются полями **Width и Height**.

Контроль при вычислении множественных ячеек. Флажок **Stop evaluating on error** позволяет контролировать вычисление последовательно исполняемых ячеек при вычислении полной М-книги. Если эта опция отмечена, то при возникновении ошибки ни одни из последующих ячеек не вычисляется; если нет, то вычисления продолжаются независимо от того, возникают ошибки или нет.

Кнопки OK и Cancel. Для подтверждения выбранных установок используется кнопка **OK**. Эти установки действуют только на результаты, которые будут получены после нажатия кнопки. Они не влияю на ранее вычисленные результаты, если только соответствую-

щие ячейки ввода не будут перевычислены. Кнопка **Cancel** служит для отказа от выбранных установок.

3 Программирование в среде Matlab 5

3.1 Введение

Файлы, которые содержат коды языка MATLAB, называются М-файлами. Для создания М-файла используется текстовый редактор; вызову М-файла предшествует присваивание значений входным аргументам; результатом является значение выходной переменной. Таким образом, вся процедура включает две операции:

- Создать М-файл, используя текстовый редактор:

```
function c = myfile(a, b)
c = sqrt((a.^2)+(b.^2))
```

- Вызвать М-файл из командной строки или из другого М-файла:

```
>> a = 7.5
>> b = 3.342
>> c = myfile(a, b)
c =      8.2109
```

Типы М-файлов. Существует два типа М-файлов: М-сценарии и М-функции со следующими характеристиками:

М-сценарий	М-функция
Не использует входных и выходных аргументов	Использует входные и выходные аргументы
Оперирует с данными из рабочей области	По умолчанию, внутренние переменные являются локальными по отношению к функции
Предназначен для автоматизации последовательности шагов, которые нужно выполнять много раз	Предназначена для расширения возможностей языка MATLAB (библиотеки функций, пакеты прикладных программ)

Структура М-файла. М-файл, оформленный в виде функции состоит из следующих компонентов:

```
function f = fact (n)
% FACT Вычисление факториала.
% fact(n) возвращает n! - факториал числа n
% Вычислить fact (n) = prod(1:n).
f = prod(1:n);
```

Строка определения функции
Первая строка комментария
Комментарий

Тело функции

Структура этой простейшей функции содержит компоненты, которые являются общими для любых функций системы MATLAB:

- Строка определения функции задаёт имя, количество и порядок следования входных и выходных аргументов.
- Первая строка комментария определяет назначение функции. Она выводится на экран с помощью команд **lookfor** или **help** имя каталога.
- Комментарий выводится на экран вместе с первой строкой при использовании команды **help** имя функции.
- Тело функции - это программный код, который реализует вычисления и присваивает значения выходным аргументам.

3.2 Создание М-файлов. М-сценарии. М-функции

М-файлы

М-файлы являются обычными текстовыми файлами, которые создаются с помощью текстового редактора. Для операционной среды персонального компьютера система MATLAB поддерживает специальный встроенный редактор/отладчик, хотя можно использовать и любой другой текстовый редактор с ASCII-кодами. Открыть редактор можно двумя способами:

- из меню **File** выбрать опцию **New**, а затем **M-File**.
- использовать команду редактирования **edit**.

Пример. Команда **edit poof** запускает редактор и открывает файле **poof.m**. Если имя файла опущено, то запускается редактор и открывается файл без имени. Далее можно записать, например, функцию **fact**, приведенную выше, вводя строки текста и сохраняя их в файле с именем **fact.m** в текущем каталоге. Как только такой файл создан, можно выполнить следующие команды:

- Вывести на экран имена файлов текущего каталога:
  what
- Вывести на экран текст М-файла **fact.m**:
  type fact

- Вызвать функцию **fact** с заданными параметрами:
  fact (5)
  ans= 120

M-сценарии

Сценарии являются самым простым типом M-файла – у них нет входных и выходных аргументов. Они используются для автоматизации многократно выполняемых вычислений. Сценарии оперируют данными из рабочей области и могут генерировать новые данные для последующей обработки в этом же файле. Данные, которые используются в сценарии, сохраняются в рабочей области после завершения сценария и могут быть использованы для дальнейших вычислений.

Пример. Следующие операторы вычисляют радиус-вектор **rho** для различных тригонометрических функций от угла **theta** и строят последовательность графиков в полярных координатах.

Строка комментария	**% M-file petals - сценарий построения** **% лепесткового графика**
Вычисления	**theta = -pi:0.01:pi;** **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**
Команды графического вывода	**polar (theta, rho(i, :))** **pause** **end**

Создайте M-файл **petals.m**, вводя указанные выше операторы. Этот файл является сценарием. Ввод команды **petals.m** в командной строке системы MATLAB вызывает выполнение операторов этого сценария.

После того, как сценарий отобразит первый график, нажмите клавишу **Return,** чтобы перейти к следующему графику. В сценарии отсутствуют входные и выходные аргументы; программа **petals.m** сама создаёт переменные, которые сохраняются в рабочей области системы MATLAB. Когда выполнение завершено, переменные (**i, theta и rho**) остаются в рабочей области. Для того чтобы увидеть этот список, следует воспользоваться командой **whos**.

M-функции

М-функции являются М-файлами, которые допускают наличие входных и выходных аргументов. Они работают с переменными в пределах собственной рабочей области, отличной от рабочей области системы MATLAB.

Пример. Функция **average** - это достаточно простой М-файл, который вычисляет среднее значение элементов вектора:

```
function y = average (x)
% AVERAGE Среднее значение элементов вектора.
% AVERAGE(X), где X - вектор. Вычисляет среднее значение элементов вектора.
% Если входной аргумент не является вектором, генерируется ошибка.
    [m,n] = size(x);
    if (~((m == 1) | (n == 1)) | (m == 1 & n == 1))
        error('Входной массив должен быть вектором')
    end
    y =sum(x)/length(x);          % Собственно вычисление
```

Попробуйте ввести эти команды в М-файл, именуемый **average.m**. Функция **average** допускает единственный входной и единственный выходной аргументы. Для того чтобы вызвать функцию **average**, надо ввести следующие операторы:

```
z = 1:99;
average(z)
ans = 50
```

Структура М-функции. М-функция состоит из:

- строки определения функции;
- первой строки комментария;
- собственно комментария;
- тела функции;
- строчных комментариев.

Строка определения функции. Строка определения функции сообщает системе MATLAB, что файл является М-функцией, а также определяет список входных аргументов.

Пример . Строка определения функции **average** имеет вид:

```
function y = average(x)
```

Здесь:

1. **function** - ключевое слово, определяющее М-функцию;
2. **y** - выходной аргумент;
3. **average** - имя функции;
4. **x** - входной аргумент.

Каждая функция в системе MATLAB содержит строку определения функции, подобную приведенной. Если функция имеет более одного выходного аргумента, список выходных аргументов помещается в квадратные скобки. Входные аргументы, если они присутствуют, помещаются в круглые скобки. Для отделения аргументов во входном и выходном списках применяются запятые.

Пример

function [x, y, z] = sphere(theta, phi, rho)

Имена входных переменных могут, но не обязаны совпадать с именами, указанными в строке определения функции.

Первая строка комментария. Для функции **average** первая строка комментария выглядит так:

% AVERAGE Среднее значение элементов вектора

Это - первая строка текста, которая появляется, когда пользователь набирает команду **help** <имя_функции>. Кроме того, первая строка комментария выводится на экран по команде поиска **lookfor**. Поскольку эта строка содержит важную информацию об М-файле, она должна быть тщательно составлена.

Комментарий. Для М-файлов можно создать **online**-подсказку, вводя текст в одной или более строках комментария.

Пример. Сформируем несколько строк комментария:

% Функция average(x) вычисляет среднее значение элементов вектора x

% Если входной аргумент не является вектором, выдается ошибка

Тогда при вводе команды подсказки **help** <имя_функции>, система MATLAB отображает строки комментария, которые размещаются между строкой определения функции и первой пустой строкой, либо началом программы. Команда **help** <имя_функции> игнорирует комментарии, размещенные вне этой области.

Пример.

help sin

SIN Sine.

SIN(X) is the sine of the elements of X

SIN(X) вычисляет функцию синуса элементов массива X

Оглавление каталога. Можно создать комментарий для целого каталога, если сформировать специальный файл с именем **Contents.m**. Этот файл должен содержать только строки комментариев. MATLAB выводит на экран строки файла **Contents.m** по команде

help <имя_каталога>. Если каталог не содержит файла **Contents.m**, то по команде : **help** <имя_каталога> - распечатывается первая строка комментария для каждого М-файла данного каталога.

Тело функции. Тело функции содержит код языка MATLAB, который выполняет вычисления и присваивает значения выходным аргументам. Операторы в теле функции могут состоять из вызовов функций, программных конструкций для управления потоком команд, интерактивного ввода/вывода, вычислений, присваиваний, комментариев и пустых строк.

Пример. Тело функции **average** включает ряд простейших операторов программирования:

Оператор вызова функции size	**m, n] = size(x);**
Начало оператора if	**if (~((m == 1) | (n == 1)) | (m == 1 & n == 1))**
Сообщение об ошибке	**Error('Input должно быть вектором')**
Конец оператора if	**end**
Вычисление и присваивание	**y = sum(x)/length(x);**

Как уже говорилось ранее, комментарии отмечаются знаком (%). Строка комментария может быть размещена в любом месте М-файла, в том числе и в конце строки.

Пример.

```
% Найти сумму всех элементов вектора x
    y = sum(x)              % Использована функция sum.
```

Кроме строк комментариев в текст М-файла можно включать пустые строки. Однако надо помнить, что пустая строка может служить указателем окончания подсказки.

Имена М-функций. В системе MATLAB на имена М-функций налагаются те же ограничения, что и на имена переменных - их длина не должна превышать 31 символа. Более точно, имя может быть и длиннее, но система MATLAB принимает во внимание только первые 31 символ. Имена М-функций должны начинаться с буквы; остальные символы могут быть любой комбинацией букв, цифр и подчеркиваний. Имя файла, содержащего М-функцию, составляется из имени функции и расширения **".m"**.

Пример
average.m

Если имя файла и имя функции в строке определения функции разные, то используется имя файла, а внутреннее имя игнорируется. Хотя имя функции, определенное в строке определения функции, может и не совпадать с именем файла, настоятельно рекомендуется использовать одинаковые имена.

Двойственность функций и команд. Команды системы MATLAB - это операторы вида:

load
help

Многие команды могут быть модифицированы добавлением операндов:

load August17.dat
help magic
type rank

Альтернативный метод задания модификаторов - определить их в качестве строковых аргументов функции:

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

В этом заключается двойственность понятий команды и функции в системе MATLAB. Любая команда вида

command argument

может быть записана в форме функции

command('argument').

Преимущество функционального описания проявляется, когда строка аргументов формируется по частям. Следующий пример показывает, как может быть обработана последовательность файлов **August1.dat, August2.dat**, и т.д. Здесь используется функция **int2str**, которая переводит целое число в строку символов, что помогает сформировать последовательность имён файлов.

```
for d = 1:31
    s = ['August' int2str(d) '.dat']
    load(s)    %Загрузить файл с именем August'd'.dat
    % Операторы обработки файла
end
```

3.3 Выполнение М-функций. Списки аргументов.

Типы аргументов. Типы данных

Выполнение М-функций

М-функцию можно вызвать из командной строки системы MATLAB или из других М-файлов, обязательно указав все необходимые атрибуты - входные аргументы в круглых скобках, выходные аргументы в квадратных скобках.

Назначение имени. Когда появляется новое имя, система MATLAB проверяет:

1. Не является ли новое имя именем переменной.
2. Не является ли это имя именем подфункции, то есть функции, которая размещена в этом же М-файле и является вызываемой.
3. Не является ли оно именем частной функции, размещаемой в каталоге private. Этот каталог доступен только М-файлам, размещенным на один уровень выше.
4. Не является ли оно именем функции в пути доступа системы MATLAB. В этом случае система использует тот М-файл, который встречается первым в пути доступа.

В случае дублирования имен система MATLAB использует первое имя в соответствии с вышеприведенной 4-уровневой иерархией. Следует отметить, что в системе MATLAB 5 допускается переопределять функцию по правилам объектно-ориентированного программирования.

Вызов функции. При вызове М-функции, система MATLAB транслирует функцию в псевдокод и загружает в память. Это позволяет избежать повторного синтаксического анализа. Псевдокод остаётся в памяти до тех пор пока не будет использована команда **clear** или завершен сеанс работы.

Допустимы следующие модификации команды **clear**:

clear <имя_функции>	Удалить указанную функцию из рабочей области
clear functions	Удалить все откомпилированные программы
clear all	Удалить программы и данные

Создание Р-кода. Можно сохранить откомпилированные М-функции или М-сценарии для последующих сеансов, используя команду **pcode** в форме:

pcode average

Эта команда выполняет синтаксический анализ М-файла **average.m** и сохраняет результирующий псевдокод в файле с именем **average.p**. Это позволяет избежать повторного разбора во время нового сеанса работы. Поскольку синтаксический анализ выполняется очень быстро, применение команды **pcode** почти не влияет на скорость ее исполнения. Применение Р-кода целесообразно в двух случаях:

- когда требуется выполнять синтаксический анализ большого числа М-файлов, необходимых для визуализации графических объектов в приложениях, связанных с разработкой графического интерфейса пользователя; в этом случае применение Р-кода обеспечивает существенное ускорение;
- когда пользователь хочет скрыть алгоритмы, реализованные в М-файле.

Правила передачи аргументов. С точки зрения программиста, система MATLAB передает аргумент его значением. На самом деле значением передаются только те аргументы, которые изменяются при работе этой функции. Если функция не изменяет значения аргумента, а только использует его для вычислений, то аргумент передается ссылкой, что позволяет оптимизировать использование памяти.

Рабочие области функции. Каждой М-функции выделяется дополнительная область памяти, не пересекающаяся с рабочей областью системы MATLAB. Такая область называется рабочей областью функции. Каждая функция имеет свою собственную рабочую область. При работе с системой MATLAB можно получить доступ только к переменным, размещенным в рабочей области системы или в рабочей области функции. Если переменная объявлена глобальной, то ее можно рассматривать как бы принадлежащей нескольким рабочим областям.

Проверка количества аргументов. Функции **nargin** и **nargout** позволяют определить количество входных и выходных аргументов вызываемой функции. Эту информацию в дальнейшем можно использовать в операторах условия для изменения хода вычислений.

Пример:

```
function c = testarg1(a,b)
if(nargin == 1)
  c = a.^2;
```

```
elseif (nargin == 2)
        c = + b;
end
```

При задании единственного входного аргумента функция вычисляет квадрат входной переменной; при задании двух аргументов выполняется операция сложения.

Рассмотрим более сложный пример - выделение части символьной строки до разделителя, в качестве которого можно использовать пробел или любой другой символ. При задании одного входного аргумента функция должна выделить часть строки до разделителя, в качестве которого по умолчанию используется пробел; причем все пробелы в начале строки удаляются. При задании двух аргументов в качестве второго аргумента должен быть указан символ разделителя. Эта функция оформлена в виде М-функции **strtok**, которая находится в каталоге **strfun**.

```
function [token, remainder] =
   strtok(string, delimiters)
```

Функция должна иметь хотя бы один входной аргумент

```
if nargin < 1,
  error("Недостаточно входных аргуметов");
end
token = [];
remainder = [];
len = length(string);
if len == 0
   return
end
```

Если входной аргумент один, то в качестве разделителя используется пробел.

```
if (nargin == 1)
   delimiters = [9:13 32]; % Символы пробелов
end
```

Определить начало выделяемой подстроки

```
i = 1;
while (any(string(i) == delimiters))
i=i+1;
if (i > len),
  return
end
```

Определить конец выделяемой подстроки

```
end
start = i
while (~any(string( i ) == delimiters))
i = i+1;
if ( i > len),
```

Выделение остатка строки

```
    break
  end
end
finish = i - 1;
token = string(start:finish);
if (nargout == 2)
   remainder = string(finish+1:end);
end
```

Заметим, что порядок следования аргументов в выходном списке имеет важное значение. Если при обращении к М-функции выходной аргумент не указан, по умолчанию выводится первый аргумент. Для формирования и вывода последующих аргументов требуется организовать соответствующее обращение к М-функции.

Списки аргументов

Функции **varargin** и **varargout** позволяют передавать произвольное количество входных и выходных аргументов. Система MATLAB упаковывает все заданные входные и выходные аргументы в массив ячеек. Каждая ячейка может содержать любой тип и любое количество данных.

Пример. Функция **testvar** допускает в качестве входных аргументов любое количество векторов из двух элементов и выводит на экран линии, их соединяющие.

```
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)
```

Таким образом, функция **testvar** может работать с входными списками разной длины.

Пример:

```
testvar([2 3], [1 5], [4 8], [6 5], [4 2], [2 3])
testvar([-1 0], [3 -5], [4 2], [1 1])
```

Формирование входного массива varargin. Поскольку список varargin хранит входные аргументы в массиве ячеек, то необходимо использовать индексы ячеек для извлечения данных. Индекс ячейки состоит из двух компонентов:

- индекс в фигурных скобках;
- индекс в круглых скобках.

Пример:

y(i)= varargin{i}(2);

Здесь индекс в фигурных скобках **{i}** указывает адрес i-ой ячейки массива varargin, а индекс в круглых скобках (2) указывает на второй элемент в ячейке.

Формирование выходного массива varargout. При произвольном количестве выходных аргументов их необходимо упаковать в массив ячеек **varargout.** Чтобы определить количество выходных аргументов функции, надо использовать функцию **nargout**.

Пример

Следующая функция использует в качестве входа массив из двух столбцов, где первый столбец - множество значений координаты x, а второй - множество значений координаты y. Функция разбивает массив на отдельные векторы, которые могут быть переданы в функцию testvar в качестве входов.

```
function [varargout] = testvar2(arrayin)
for i = 1:nargout
    varargout{i} = arrayin(i, :);
end
```

Оператор присваивания в цикле **for** использует синтаксис присваивания массивов ячеек. Левая часть оператора присваивания использует фигурные скобки, чтобы указать, что данные в виде строки массива присваиваются ячейке. Для вызова функции **testvar2** можно использовать следующие операторы:

```
 a = [1 2 3 4 5; 6 7 8 9 0]';
[p1,p2,p3,p4,p5] = testvar2(a)
     p1 =    1    6
     p2 =    2    7
     p3 =    3    8
     p4 =    4    9
     p5 =    5    0
```

Использование массивов ячеек в списках аргументов. Аргументы **varargin** и **varargout** должны быть последними в соответствующих списках аргументов. При вызове функции аргументы, предшествующие **varargout**, должны быть вычислены внутри функции.

Пример. Приведенные ниже заголовки функций показывают правильное использование списков **varargin** и **varargout**:

```
function[out1, out2] = example1(a,b,varargin)
function[i,j,varargout] = example2(x1,y1,x2,y2,flag)
```

Типы переменных

Локальные и глобальные переменные. Использование переменных в М-файле ничем не отличаетсч от использования переменных в командной строке, а именно:

- переменные не требуют объявления; прежде чем переменной присвоить значение, необходимо убедиться, что всем переменным в правой части значения присвоены;
- любая операция присваивания создает переменную, если это необходимо, или изменяет значение существующей переменной;
- имена переменных начинаются с буквы, за которой следует любое количество букв, цифр и подчеркиваний; система MATLAB различает символы верхнего и нижнего регистров;
- имя переменной не должно превышать 31 символа. Более точно, имя может быть и длиннее, но система MATLAB принимает во внимание только первые 31 символ.

Обычно каждая М-функция, задаваемая в виде М-файла, имеет собственные локальные переменные, которые отличны от переменных других функций и переменных рабочей области. Однако, если несколько функций и рабочая область объявляют некоторую переменную глобальной, то все они используют единственную копию этой переменной. Любое присваивание этой переменной распространяется на все функции, где она объявлена глобальной.

Пример. Допустим, требуется исследовать влияние коэффициентов **a** и **b** для модели хищник-жертва, описываемой уравнениями Лотке-Вольтерра:

$$\dot{y}_1 = y_1 - \alpha y_1 y_2$$

$$\dot{y}_2 = -y_2 + \beta y_1 y_2$$

Создадим М-файл **lotka.m**:

```
function yp = lotka(t, y)
 %LOTKA уравнения Лотке-Вольтерра для модели хищник-жертва
 global ALPHA BETA
 yp = [y(1) - ALPHA*y(1)*y(2); -y(2) + BETA*y(1)*y(2)];
```

Затем через командную строку введем операторы:

```
global ALPHA BETA
ALPHA = 0.01;
BETA = 0.02;
```

```
[t,y] = ode23('lotka2',[0 10],[1; 1]);
plot(t,y)
```

Команда **global** объявляет переменные **ALPHA** и **BETA** глобальными и следовательно, доступными в функции **lotka.m**. Таким образом, они могут быть изменены из командной строки, а новые решения будут получены без редактирования М-файла **lotka.m**. Для работы с глобальными переменными необходимо:

- объявить переменную как глобальную в каждой М-функции, которая необходима эта переменная. Для того чтобы переменная рабочей области была глобальной, необходимо объявить ее как глобальную из командной строки;
- в каждой функции использовать команду **global** перед первым появлением переменной; рекомендуется указывать команду **global** в начале М-файла.

Имена глобальных переменных обычно более длинные и более содержательные, чем имена локальных переменных, и часто используют заглавные буквы. Это необязательно, но рекомендуется, чтобы обеспечить удобочитаемость кода языка MATLAB и уменьшить вероятность случайного переопределения глобальной переменной.

Специальные переменные. Некоторые М-функции возвращают специальные переменные, которые играют важную роль при работе в среде системы MATLAB :

ans	Последний результат; если выходная переменная не указана, то MATLAB использует переменную **ans**.
eps	Точность вычислений с плавающей точкой; определяется длиной мантиссы и для PC **eps** = 2.220446049250313e-016
realmax	Максимальное число с плавающей точкой, представимое в компьютере; для PC **realmax** = 1.797693134862316e+308.
realmin	Минимальное число с плавающей точкой, представимое в компьютере; для PC **realmin** = 2.225073858507202e-308.
pi	Специальная переменная для числа p: **pi**=3.141592653589793e+000.

i, j	Специальные переменные для обозначения мнимой единицы
inf	Специальная переменная для обозначения символа бесконечности ?
NaN	Специальная переменная для обозначения неопределенного значения - результата операций типа: **0/0, inf/inf**.
computer	Специальная переменная для обозначения типа используемого компьютера; для PC - PCWIN.
flops	Специальная переменная для обозначения количества операций с плавающей точкой.
version	Специальная переменная для хранения номера используемой версии системы MATLAB.

Соответсвущие М-функции, генерирующие эти специальные переменные, находятся в каталоге **elmat** и поддержаны **online**-подсказкой.

Типы данных

В системе MATLAB определено шесть базовых типов данных, каждый из которых является многомерным массивом. Шесть классов - это **double**, **char, sparse, uint8, cell**, и **struct**. Двумерные версии этих массивов называются матрицами, откуда MATLAB и получил свое имя МАТричная ЛАБоратория.

Диаграмма принадлежности того или иного объекта системы MATLAB к одному из классов имеет следующий вид (рисунок 3.1):

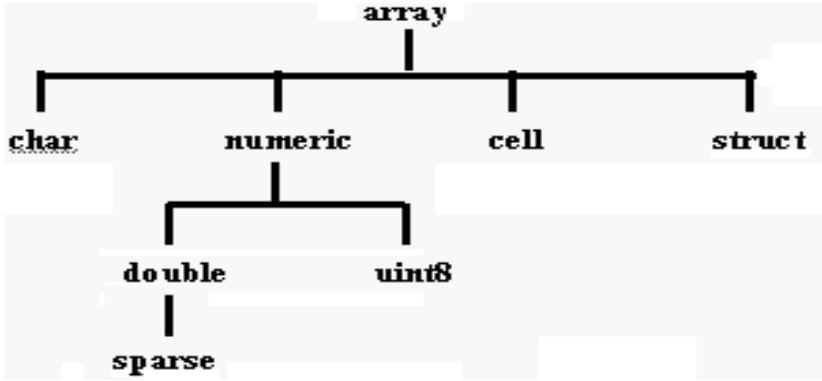


Рисунок 3.1

Вероятно, что чаще всего вам придется иметь дело только с двумя из этих типов данных: массив чисел удвоенной точности (**double**) и массив символов (**char**), или просто строка. Это связано с тем, что все вычисления в системе MATLAB выполняются с удвоенной точностью и большинство функций работают с массивами чисел удвоенной точности или строками.

Другие типы данных предназначены для таких специальных приложений, как работа с разреженными матрицами (**sparse**), обработка изображений (**uint8**), работа с массивами большой размерности (**cell** и **struct**).

Нельзя задать тип переменной **numeric** или **array**. Эти типы называются виртуальными и служат только для того, чтобы сгруппировать переменные, которые имеют общие атрибуты.

Тип **uint8** предназначен для эффективного хранения данных в памяти. К данным этого типа можно применять только базовые операции индексации и изменения размеров, но нельзя выполнить никакой математической операции. Для этого такие массивы необходимо преобразовать в тип **double**.

Создание собственных типов и добавление методов для встроенных типов. Нижеприведенная таблица содержит седьмой тип данных - **UserObject.** Язык MATLAB позволяет создавать собственные типы данных и работать с ними по аналогии со встроенными типами.

Для встроенных типов данных можно переопределять метод точно также, как это делается для объекта. Например, чтобы задать операцию сортировки для массива типа **uint8**, необходимо создать метод (**sort.m** или **sort.mex**) и поместить его в специальный каталог **@uint8**. Следующая таблица описывает типы данных более подробно.

Класс	***Пример***	***Описание***
Double	**[1 2; 3 4]** **5 + 6i**	**Числовой массив** удвоенной точности (это наиболее распространенный тип переменной в системе MATLAB

Char	**'Привет'**	**Массив символов** (каждый символ - длиной 16 битов), часто именуется строкой.
Sparse	**Speye(5)**	**Разреженная матрица** удвоенной точности (только двумерная). Разреженная структура применяется для хранения матриц с небольшим количеством ненулевых элементов, что позволяет использовать лишь небольшую часть памяти, требуемой для хранения полной матрицы. Разреженные матрицы требуют применения специальных методов для решения задач.
Cell	**{ 17 'привет' eye (2)}**	**Массив ячеек**. Элементы этого массива содержат другие массивы. Массивы ячеек позволяют объединить связанные данные, возможно различных размеров, в единую структуру.
Struct	**A.day = 12; A.color = 'Red'; A.mat = magic(3);**	**Массив записей**. Он включает имена полей. Поля сами могут содержать массивы. Подобно массивам ячеек, массивы записей объединяют связанные данные и информацию о них.

Uint8	**Uint8 (magic (3))**	**Массив 8-разрядных целых чисел без знаков**. Он позволяет хранить целые числа в диапазоне от 0 до 255 в 1/8 части памяти, требуемой для массива удвоенной точности. Никакие математические операции для этих массивов не определены.
UserObject	**inline('sin(x)')**	**Тип данных, определяемый пользователем.**

Описание диаграммы. Соединительные линии на диаграмме (рисунок 3.1) определяют принадлежность того или иного типа данных к одному или нескольким классам.

Пример. Матрица типа **sparse** имеет также типы **double** и **numeric**. Операторы

isa(S',sparse')
isa(S',double')
isa(S',numeric')

возвращают значения 1(истина), то есть **S** - числовая разреженная матрица удвоенной точности. Обратите внимание, что тип **array** - массив находится в вершине диаграммы. Это означает, что все данные системы MATLAB являются массивами.

Каждому типу данных можно соотнести свои функции и операторы обработки, или другими словами, методы. Дочерние типы данных, расположенные на диаграмме ниже родительского типа, поддержаны также и методами родителя. Следовательно, массив типа **double** поддержан методами, применяемыми для типа **numeric**. В таблице приведены некоторые из таких методов:

Класс	***Метод***

Массив **array**	Вычисление размера (**size**), длины (**length**), размерности (**ndims**), объединение массивов **([a b]**), транспонирование (**transpose**), многомерная индексация (**subsindex**), переопределение (**reshape**) и перестановка (**permute**) размерностей многомерного массива.
Массив ячеек **cell**	Индексация с использованием фигурных скобок {**e1,…,en**} и разделением элементов списка запятыми.
Строка **Char**	Строковые функции (**strcmp, lower**), автоматическое преобразование к типу **double** для применения методов класса **double.**
Double	Арифметические и логические операции, математические функции, функции от матриц.
Numeric	Поиск (**find)**, обработка комплексных чисел (**real, imag**), формирование векторов, выделение строк, столбцов, подблоков массива, расширение скаляра.
Sparse	Операции над разреженными матрицами.
Массив записей **Struct**	Доступ к содержимому поля .**field** (разделитель элементов списка - запятая).
Uint8	Операция хранения (чаще всего используется с ППП **Image Processing Toolbox)**
UserObject	Определяется пользователем

Пустые массивы. Ранние версии системы MATLAB допускали единственную форму пустого массива размера **0x0**, обозначаемого как [ ]. MATLAB поддерживает массивы, у которых одна, но не все из размерностей, равна нулю, то есть массивы с размерами **1x0**, **10x0x20** или **[3 4 0 5 2]** определяются как пустые. Квадратные скобки **[ ]** продолжают обозначать массив **0x0**. Пустые массивы другого размера могут быть созданы с помощью функций **zeros, ones, rand** или **eye**. Например, для формирования пустого массива размера **0x5**, можно использовать оператор присваивания

E = zeros(0,5).

Основное назначение пустых массивов состоит в том, чтобы любая операция, которая определена для массива(матрицы) размера **m?n**, определяла правильный результат для случая, когда **m** или **n** равно нулю. Размер массива(матрицы) результата должен соответствовать значению функции, вычисленной в нуле. Например, оператор

C = [A B]

требует, чтобы массивы **A** и **B** имели одинаковое число строк. Таким образом, если массив **A** имеет размер **m?n**, а **B - m?p**, то **C** есть массив размера **m?(n+p)**. Результат будет правильным, если любой из параметров **m, n** или **p** равен нулю.

Многие операции в системе MATLAB создают вектор-строку или вектор-столбец. В этом случае результат может быть, либо пустой вектор-строкой

r = zeros(1, 0),

либо пустым вектор-столбцом

C = zeros(0, 1).

MATLAB 5 и более поддерживает правила системы MATLAB 4 для операторов **if** и **while**. Например, условный оператор типа

if A, S1, else, S0, end

выполняет оператор **S0**, когда **A** - пустой массив.

Некоторые функции системы MATLAB такие, как **sum**, **prod, min** и **max** понижают размерность результата: если аргумент массив, то результат - вектор; если аргумент вектор, то результат - скаляр. Для этих функций при пустом массиве входа получаются следующие результаты:

sum([ ]) = 0 ;
prod([ ]) = 1 ;
max([ ]) = [ ] ;
min([ ]) = [ ] .

3.4 Операторы системы MATLAB 5. Объединение операторов в арифметические выражения. Встроенные функции

Операторы системы MATLAB

Операторы системы MATLAB делятся на три категории:

- арифметические операторы позволяют конструировать арифметические выражения и выполнять числовые вычисления.
- операторы отношения позволяют сравнивать числовые операнды.
- логические операторы позволяют строить логические выражения.

Логические операторы имеют самый низкий приоритет относительно операторов отношения и арифметических операторов.

Арифметические операторы. При работе с массивом чисел установлены следующие уровни приоритета среди арифметических операций :

уровень 1:	поэлементное транспонирование (.'), поэлементное возведение в степень (.^), эрмитово сопряженное транспонирование матрицы ('), возведение матрицы в степень (^);
уровень 2:	унарное сложение (+), унарное вычитание (-);
уровень 3:	умножение массивов (.*), правое деление (./), левое деление массивов (.\), умножение матриц (*), решение систем линейных уравнений, операция (/), операция (\);
уровень 4:	сложение (+), вычитание (-);
уровень 5:	оператор формирования массивов (:).

Внутри каждого уровня операторы имеют равный приоритет и вычисляются в порядке следования слева направо. Заданный по умолчанию порядок следования может быть изменен с помощью круглых скобок.

Пример. Пусть заданы 2 вектора
A = [3 9 5];
B = [2 1 5];
Результаты выполнения оператора
C = A./B. ^ 2 равен **C = 0.7500 9.0000 0.2000 ,** а оператора
C = (A./B). ^ 2 равен **C = 2.2500 81.0000 1.0000**. Как видно результаты совершенно различны.

Арифметические операторы допускают использование индексных выражений. Например:

b = sqrt (A(2)) + 2*B (1)
b = 7

Арифметические операторы системы MATLAB работают, как правило, с массивами одинаковых размеров. Для векторов и прямоугольных массивов оба операнда должны быть одинакового размера, за исключением единственного случая, когда один из них - скаляр. Если один из операндов скалярный, а другой нет, в системе MATLAB принято, что скаляр расширяется до размеров второго операнда и заданная операция применяется к каждому элементу. Такая операция называется расширением скаляра.

Операторы отношения. В системе MATLAB определено 6 следующих операторов отношения:

- < Меньше
- <= Меньше или равно
- > Больше
- > = Больше или равно
- == Равно тождественно
- ~ = Не равно

Операторы отношения выполняют поэлементное сравнение двух массивов равных размерностей. Для векторов и прямоугольных массивов, оба операнда должны быть одинакового размера, за исключением случая когда один из них скаляр. В этом случае MATLAB сравнивает скаляр с каждым элементом другого операнда. Позиции, где это соотношение истинно, получают значение 1, где ложно - 0. Операторы отношения, как правило, применяется для изменения последовательности выполнения операторов программы. Поэтому они чаще всего используются в теле операторов **if, for, while, switch**.

Операторы отношения всегда выполняются поэлементно.

Пример. Выполним сравнение двух массивов, используя условие **A<B**:

```
A = [2 7 6; 9 0 -1; 3 0.5 6];
B = [8 0.2 0; -3 2 5; 4 -1 7];
A< B
ans =
     1   0   0
     0   1   1
     1   0   1
```

Полученная матрица указывает позиции, где элемент **A** меньше соответствующего элемента **B.**

При вычислении арифметических выражений операторы отношения имеют более низкий приоритет, чем арифметические, но более высокий, чем логические операторы.

Операторы отношения могут применятся к многомерным массивам, для которых одна из размерностей равна нулю, при условии, что оба массива - одинакового размера или один из них - скаляр. Однако выражения типа **A == [ ]** применимы только к массивам размера **0x0** или **1x1**, а в других случаях вызывают ошибку. Поэтому наиболее универсальный способ проверить, является ли массив пустым - это применить функцию **isempty (A).**

Логические операторы. В состав логических операторов системы MATLAB входят следующие операторы:

&	И
|	ИЛИ
~	НЕТ

В дополнение к этим операторам, каталог **bitfun** содержит ряд функций, которые выполняют поразрядные логические операции.

Логические операторы реализуют поэлементное сравнение массивов одинаковых размерностей. Для векторов и прямоугольных массивов оба операнда должны быть одинакового размера, за исключением случая, когда один из них скаляр. В последнем случае MATLAB сравнивает скаляр с каждым элементом другого операнда. Позиции, где это соотношение истинно, получают значение 1, где ложно - 0.

Каждому логическому оператору соответствует некоторый набор условий, которые определяю результат логического выражения:

- Логическое выражение с оператором **AND (&)** является истинным, если оба операнда - истинны. Если элементами логического выражения являются числа, то выражение истинно, если оба операнда отличны от нуля.

  Пример. Пусть заданы два числовых вектора:

  u = [1 0 2 3 0 5];

  v = [5 6 1 0 0 7];

  и логическое выражение с оператором **AND (&)** :

  U & v

  ans =

  1 0 1 0 0 1

- Логическое выражение с оператором **OR (|)** является истинным, если один из операндов или оба операнда логически истин-

ны. Выражение ложно, только если оба операнда логически ложны.

Если элементами логического выражения являются числа, то выражение ложно, если оба операнда равны нулю.

Пример. Используем векторы **u** и **v**, определенные выше, и выполним логическое выражение с оператором **OR (|):**

```
U | v
ans =
 1     1     1     1     0     1
```

- Логическое выражение с оператором **NOT (~)** строит отрицание. Результат логически ложен, если операнд истинен, и истинен, если операнд ложен. Если элементами логического выражения являются числа, то любой операнд, отличный от нуля, становится нулем, и любой нулевой операнд становится единицей.

Пример. Используем вектор **u**, заданный выше и построим логическое выражение с оператором **NOT (~):**

```
~ u
ans =
0     1     0     0     1     0
```

Логические функции. В дополнение к логическим операторам в состав системы MATLAB включено ряд логических функций:

- Функция **xor(a, b)** реализует операцию ИСКЛЮЧИТЕЛЬНОЕ ИЛИ. Выражение, содержащее ИСКЛЮЧИТЕЛЬНОЕ ИЛИ истинно, если один из операндов имеет значение TRUE, а другой FALSE. Для числовых выражений, функция возвращает 1, если один из операндов отличен от нуля, а другой - нуль.

Пример. Рассмотрим два числовых операнда **a** и **b**:

```
a = 1;
b = 1;
```

Тогда операция **xor** даёт результат:

```
Xor (a, b)
ans =     0
```

- Функция **all** возвращает 1, если все элементы вектора истинны или отличны от нуля.

Пример. Пусть задан вектор **u** и требуется проверить его на условие «все ли элементы меньше 3?». Если это условие выполняется, то выдается сообщение «Все элементы меньше 3».

```
u = [1 2 3 4 0];
if all(u < 3)
   Disp ('Все элементы меньше 3')
end
```

В данном случае никакого сообщения не появится, но если в каче-

стве вектора **u** взять вектор

u = [0 1 2 0]

то появится сообщение

ans = ' Все элементы меньше 3'

В случае массивов функция **all** проверяет столбцы, то есть является ориентированной по столбцам.

Пример.

```
A = [0 1 2; 3 5 0]
 all(A)
```

- Функция **any** возвращает 1, если хотя бы один из элементов аргумента отличен от нуля; иначе, возвращается 0. В случае обработки массивов функция **any** является столбцовоориентированной . Функции **isnan** и **isinf** возвращают 1 для **NaN** и **Inf,** соответственно. Функция **isfinite** истинна только для величин, которые не имеют значения **inf** или **NaN**.

Пример. Рассмотрим сдующие два числовых массива **A** и **B**

```
A = [0 1 5;2 NaN -inf];
B = [0 0 15; 2 5 inf];
```

Образуем массив **C** и применим перечисленные выше функции

```
C = A./B
C =
   NaN       Inf    0.3333
1.0000       NaN       NaN

isfinite (C)        isnan (C)        isinf (C)
ans =               ans =            ans =
  0   0   1           1   0   0        0   1   0
  1   0   0           0   1   1        0   0   0
```

Полный список логических функций системы MATLAB содержится в каталоге **ops.**

Функция find. Функция **find** определяет индексы элементов массива, которые удовлетворяют заданному логическому условию. Как правило, она используется для создания шаблонов для сравнения и создания массивов индексов. В наиболее употребительной форме функция **к = find**(x <условие>) возвращает вектор индексов тех элементов, которые удовлетворяет заданному условию.

```
A = magic(4)
A =
    16     2     3    13
     5    11    10     8
```

```
     9     7     6    12
     4    14    15     1
k = find(A > 8);
A(k) = 100
A =
   100     2     3   100
     5   100   100     8
   100     7     6   100
     4   100   100     1
```

Функция вида **[i, j] = find(x)** позволяет получить индексы ненулевых элементов прямоугольного массива. Функция вида **[i, j, s] = find(x)** возвращает кроме того и их значения в виде вектора **s**.

Объединение операторов в арифметические выражения

Теперь вы имеете возможность строить выражения, которые используют любую комбинацию арифметических, логических операторов и операторов отношения.

Пример. Рассмотрим пример оператора сравнения, в котором сравниваются результаты двух выражений

```
(a*b) <  (c*d)
```

Используя скобки, можно управлять последовательностью выполнения операций

```
(A & B) == (C | D)
```

Управление последовательностью исполнения операторов. Существуют четыре основных оператора управления последовательностью исполнения инструкций:

- оператор условия **if,** в сочетании с оператором **else** и **elseif** выполняет группу инструкций в соответствии с некоторыми логическими условиями;
- оператор переключения **switch**, в сочетании с операторами **case** и **otherwise** выполняет различные группы инструкций в зависимости от значения некоторого логического условия;
- оператор условия **while** выполняет группу инструкций неопределенное число раз, в соответствии с некоторым логическим условием завершения;
- оператор цикла **for** выполняет группу инструкций фиксированное число раз. Все операторы управления включают оператор **end**, чтобы указать конец блока, в котором действует этот оператор управления.

If...else...elseif...end - Оператор условия

Синтаксис:

```
if  логиче-
ское_выражение
    инструкции
end
```

```
if логиче-
ское_выражение
      инструкции
else
    инструкции
end
```

```
if логическое_выражение
        инструкции
elseif  логиче-
ское_выражение
      инструкции
else
       инструкции
end
```

Описание:

Оператор условия **if .... end** вычисляет некоторое логическое выражение и выполняет соответствующую группу инструкций в зависимости от значения этого выражения. Если логическое выражение истинно, то MATLAB выполнит все инструкции между **if и end**, а затем продолжит выполнение программы в строке после end. Если условие ложно, то MATLAB пропускает все утверждения между **if** и **end** и продолжит выполнение в строке после **end**.

Пример.

```
if rem(a, 2) == 0
   disp('a четно')
   b = a/2;
end
```

Если логическое условие включает переменную, не являющуюся скаляром, то утверждение будет истинным, если все элементы отличны от нуля.

Пример. Пусть задана матрица **X**; запишем следующий оператор условия:

```
 if X
   инструкции
end
```

Этот оператор равносилен следующему:

```
if all(X(:))
   инструкции
end
```

Операторы **if ... else ... end** и **if ... elseif ... end** создают дополнительные ветвления внутри тела оператора **if:**

- Оператор **else** не содержит логического условия. Инструкции, связанные с ним, выполняются, если предшествующий оператор **if** (и возможно **elseif**) ложны. Оператор **elseif** содержит логическое условие, которое вычисляется, если предшествующий оператор **if**

(и возможно **elseif)** ложны. Инструкции, связанные с оператором **elseif** выполняются, если соответствующее логическое условие истинно.

- Оператор **elseif** может многократно использоваться внутри оператора условия **if.**

Пример. Рассмотрим фрагмент программы:

```
 if n < 0          % Если n < 0, вывести сообщение об ошибке.
    disp('Введенное число должно быть положительным');
 elseif rem(n,2) == 0    %Если n положительное и четное, раз
                         %     делить на 2.
    a= n/2;
 else
    a = (n+1)/2;  %Если n > 0 и нечетное, увеличить на 1
                  % и разделить.
 end
```

Если в операторе **if** условное выражение является пустым массивом, то такое условие ложно. То есть оператор условия вида

```
if A
  S1
else
  S0
end
```

выполнит инструкции **S0** только тогда, когда **A** - пустой массив.

switch...case...otherwise...end - Оператор переключения

Синтаксис:

```
switch <выражение>
        % выражение - это обязательно скаляр или строка
case  <значение1>
          инструкции
        % выполняются, если < выражение> =< значение1>
case  <значение2>
          инструкции
        % выполняются, если  <выражение> = < значение2>
...
otherwise
          инструкции
        % выполняются, если <выражение> не совпало ни с
одним из
        %значений
end
```

Оператор **switch ... case 1 ... case k ... otherwise ... end** выполняет ветвления, в зависмости от значений некоторой переменной или выражения.

Оператор переключения включает:

- Заголовок **switch**, за которым следует вычисляемое выражение (скаляр или строка).
- Произвольное количество групп **case**; Заголовок группы состоят из слова **case,** за которым следует возможное значение выражения, расположенное на одной строке. Последующие строки содержат инструкции, которые выполняются для данного значения выражения. Выполнение продолжается до тех пор, пока не встретится следующий оператор **case** или оператор **otherwise**. На этом выполнение блока **switch** завершается
- Группа **otherwise**. Заголовок включает только слово **otherwise**, начиная со следующей строки размещаются инструкции, которые выполняются, если значение выражения оказалось не обработанным ни одной из групп **case.** Выполнение завершается оператором **end.**
- Оператор **end** является последним в блоке переключателя.

Оператор **switch** работает, сравнивая значение вычисленного выражения со значениями групп **case**. Для числовых выражений оператор **case** выполняется, если <значение>== <выражение>. Для строковых выражений, оператор **case** истинен, если **strcmp**(значение, выражение) истинно.

Пример. Рассмотрим оператор **switch** со следующими условиями: он проверяет переменную **input_num**; если **input_num** равно -1, 0 или 1, то операторы **case** выводят на экран соответствующее сообщения. Если значения выражения **input_num** не равно ни одному из этих значений, то выполнение переходит к оператору **otherwise.**

```
switch input_num
  case -1
    disp('минус один')
  case 0
    disp('нуль')
  case 1
    disp('плюс один')
  otherwise
    disp('другое значение')
end
```

Оператор **switch** может использовать множественное условие в единственной группе **case** посредством включения выражения **case**, если выражение для этого условия записано в виде массива ячеек:

```
switch var
  case 1
    disp('1')
  case{2,3,4}
     disp('2 или 3 или 4')
  case 5
    disp('5')
 otherwise
    disp('что-то другое')
end
```

while...end - Оператор цикла с неопределенным числом операций

Синтаксис:

```
while выражение
  инструкции
end
```

Описание. Оператор цикла с неопределенным числом итераций **while ... end** многократно выполняет инструкцию или группу инструкций, пока управляющее выражение истинно.

Если выражение использует массив, то все его элементы должны быть истинны для продолжения выполнения. Чтобы привести матрицу к скалярному значению, следует использовать функции **any** и **all**.

Пример. Этот цикл с неопределенным числом операций находит первое целое число **n**, для которого **n!** - записывается числом, содержащим 100 знаков:

```
n = 1;
 while prod(1:n) < 1e100
     n = n + 1;
 end
```

Выход из **while**-цикла может быть реализован с помощью оператора **break**.

Если в операторе **while,** управляющее условие является пустым массивом, то такое условие ложно, то есть оператор вида **while A, S1, end** никогда не выполнит инструкции **S1**, если **A** - пустой массив.

For...end - Оператор цикла с определенным числом операций

Синтаксис:

```
for <переменная цикла> = <начальное значение>: <приращение>:<конечное значение>
   инструкции
end
```

Описание. Оператор цикла **for .... end** выполняет инструкцию или группу инструкций предопределенное число раз. По умолчанию приращение равно 1. Можно задавать любое приращение, в том числе отрицательное. Для положительных индексов выполнение завершается, когда значение индекса превышает <конечное значение>; для отрицательных приращений выполнение завершается, когда индекс становится меньше чем <конечное значение>.

Пример. Этот цикл выполняется пять раз:

```
for i = 2:6
   x(i) = 2*x(i-1);
end
```

Допустимы вложенные циклы типа:

```
for i = 1:m
  for j = 1:n
     A(i,j) = 1/(i + j - 1);
  end
end
```

Использование массива в качестве переменной цикла. В качестве переменной цикла **for** могут использоваться массивы.

Пример. Рассмотрим массив **A** размера **mxn**. Оператор цикла

```
for i = A
  инструкции
end
```

определяет переменную цикла **i** как вектор **A(:, k).** Для первого шага цикла **k** равно 1; для второго **k** равно 2, и так далее, пока k не достигнет значения **n**. То есть цикл выполняется столько раз, сколько столбцов в матрице A. Для каждого шага **i** - это вектор, содержащий один из столбцов массива A.

Встроенные функции

Начиная с версии MATLAB 5, M-файлы могут содержать коды для более, чем одной функции. Первая функция в файле -это основная функция, вызываемая по имени M-файла. Другие функции внутри файла - это подфункции, которые являются видимыми только для основной функции и других подфункций этого же файла.

Каждая подфункция имеет свой собственный заголовок. Подфункции следуют друг за другом непрерывно. Подфункции могут вызываться в любом порядке, в то время как основная функция выполняется первой.

Основная функция **function [avg, med] = newstats (u)**

```
% NEWSTATS Находит среднее значение и медиану для элементов вектора u , используя встроенные функции.
n = length(u);
avg = mean(u,n);
med = median(u,n);
```

Подфункция 1

```
function m = mean(v,n)
% Вычислить среднее.
a = sum(v)/n;
```

Подфункция 2

```
function m = median(v,n)
% Вычислить 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 и медиану входного списка. Основная функция **newstats** определяет длину списка и вызывает подфункции, передавая им длину списка **n**. Функции внутри одного и того же М-файла не могут обращаться к одним и тем же переменным, если они не объявлены глобальными переменными внутри соответствующих функций, или не переданы им в качестве параметров. Следует иметь в виду, что справка **help** может видит только основную функцию и не видит подфункций.

Когда приходит вызов функции из М-файла, то MATLAB, в первую очередь, проверяет, не является ли эта функция подфункцией. Поскольку первой проверяется наличие подфункций, то можно в качестве имени подфункции использовать имена функций системы MATLAB.

Частные каталоги. Они введены в систему MATLAB, начиная с версии 5.0. Частные каталоги представляют собой подкаталог с именем **private** родительского каталога. М-файлы частного каталога доступны только М-файлам родительского каталога. Поскольку файлы частного каталога не видимы вне родительского каталога, они могут иметь имена совпадающие, с именами файлов других каталогов системы MATLAB. Это удобно в тех случаях, когда пользователь создает собственные версии некоторой функции, сохраняя оригинал в

другом каталоге. Поскольку MATLAB просматривает частный каталог раньше каталогов стандартных функций системы MATLAB он в первую очередь находит функцию из частного каталога.

3.5 Индексы и подиндексы

Индексы. Элемент массива **A**, расположенный на пересечении строки **i** и столбца **j**, обозначается как **A(i, j).**

Пример. Рассмотрим в качестве массива **A** матрицу **magic(4):**

```
A = magic(4)
A =
16    2    3   13
 5   11   10    8
 9    7    6   12
 4   14   15    1
```

Тогда **A(4, 3)** - это элемент, расположенный на пересечении строки 4 и столбца 3, равный 15.

Можно также вычислить сумму элементов четвертого столбца

```
A(1, 4) + A(2, 4) + A(3, 4) + A(4, 4)
ans =     34
```

На элементы массива **A** можно ссылаться, используя единственный индекс, **A(k).** Это обычный способ ссылки на элементы векторов. Но точно так же можно ссылаться на элементы двумерного массива, и в этом случае этот массив рассматривается как один длинный вектор-столбец, сформированный из столбцов исходного массива. В рассматриваемом примере **A(12)** - это другой способ ссылки на значение 15, соответствующее элементу **A(4, 3).**

Если будет сделана попытка обратиться к элементу вне матрицы, то программа выдаст ошибку:

```
t = A(4, 5)
??? Index exceeds matrix dimensions
```

(Индекс превышает размерность матрицы).

Если же выполняется присвоение значения элементу с индексами, выходящими за пределы массива, то система MATLAB автоматически увеличивает размер матрицы.

Пример:

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

16 2 3 13 0
5 11 10 8 0
9 7 6 12 0
4 14 15 1 17

Выделение подблоков массива. Если в индексных выражениях использовать двоеточие, то можно ссылаться на подблоки массива. Так индексное выражение **A(1:k, j)** ссылается на блок из **k** элементов столбца **j**.

Пример:

A(1:4,3)
ans =
3
10
6
15

Здесь выделен столбец 3 матрицы **magic(4).**

Оператор

sum(A(1:4, 3))
ans = 34

вычисляет сумму элементов столбца 3.

Однако существует способ лучше. Поскольку двоеточие само по себе ссылается на все элементы строки или столбца, то последнюю сумму можно вычислить так

sum(A(:,3))
ans = 34

Кроме того, начиная с версии 5.0, на последнюю строку или столбец массива можно ссылаться с помощью ключевого слова **end**. Таким образом, оператор

sum(A(:, end))
ans = 34

вычисляет сумму элементов в последнего столбца матрицы A.

Объединение подблоков в массив. Операция объединения отдельных подблоков в массив называется конкатенацией. Даже при формировании исходной матрицу, когда объединяются отдельные элементы, осуществляется операция конкатенации. Оператор конкатенации - это пара квадратных скобок [ ], внутри которых указываются отдельные элементы или блоки массива.

Пример. Используя матрицу **A**, равную **magic(4),** образуем новую матрицу **B** размера **8x8** **B = [A A+32; A+48 A+16]**

B =

16 2 3 13 48 34 35 45

5	**11**	**10**	**8**	**37**	**43**	**42**	**40**
9	**7**	**6**	**12**	**41**	**39**	**38**	**44**
4	**14**	**15**	**1**	**36**	**46**	**47**	**33**
64	**50**	**51**	**61**	**32**	**18**	**19**	**29**
53	**59**	**58**	**56**	**21**	**27**	**26**	**24**
57	**55**	**54**	**60**	**25**	**23**	**22**	**28**
52	**62**	**63**	**49**	**20**	**30**	**31**	**17**

Эта матрица состоит из четырех блоков размера 4x4

B =

16	**3**	**2**	**13**	**48**	**35**	**34**	**45**
5	**10**	**11**	**8**	**37**	**42**	**43**	**40**
9	**6**	**7**	**12**	**41**	**38**	**39**	**44**
4	**15**	**14**	**1**	**36**	**47**	**46**	**33**
64	**51**	**50**	**61**	**32**	**19**	**18**	**28**
53	**58**	**59**	**56**	**21**	**26**	**23**	**28**
57	**54**	**55**	**60**	**25**	**22**	**23**	**28**
52	**63**	**62**	**49**	**20**	**31**	**30**	**17**

Эта матрица есть половина другого волшебного квадрата, элементы которого находятся в диапазоне целых чисел 1:64. Суммы по столбцам уже имеют правильное значение для волшебного квадрата размера **8x8**:

```
sum(B)
ans = 260   260  260  260  260  260  260  260
```

Однако сумма строк

```
sum (B')
ans = 196   196  196  196  324  324  324  324
```

совсем не та.

Попробуйте найти те перестановки элементов, которые приводят матрицу B к истинному волшебному квадрату порядка 8.

Удаление строк и столбцов. Используя понятие пустого массива, можно легко удалять строки, столбцы и целые подблоки. Допустим,

```
X = A
X =
16     2     3     13
5     11     10     8
```

```
     9      7      6      12
     4      14     15     1
```

Чтобы удалить второй столбец массива X достаточно применить оператор

```
X (:, 2) = [ ]
X =
   16     3      13
   5      10     8
   9      6      12
   4      15     1
```

При попытке удалить отдельный элемент массива возникает ошибка, поскольку результат не является массивом:

```
X (1, 2) = []
??? Indexed empty matrix assignment is not allowed.
```

(Запрещено присвоение пустой матрицы индексному выражению).

Однако использование единственного индекса позволяет удалить одиночный элемент или последовательность элементов, при этом остающиеся элементы преобразуются в вектор-строку.

Пример:

```
X = A;
X(:, 2) = [ ]
X =
     16     3      13
     5      10     8
     9      6      12
     4      15     1
X(2:1:12) = [ ]
X =      16
```

или

```
X = A;
X(:, 2) = [ ]
X =
     16     3      13
     5      10     8
     9      6      12
     4      15     1
X(2 : 2 : 10) = [ ]
X = 16   9   3   6   13   12   1
```

Индексация многомерных массивов. В системе MATLAB принято хранить каждый массив, независимо от его размерности, как вектор-столбец. Этот вектор образован объединением (конкатенацией) столбцов исходного массива.

Пример.

Система MATLAB хранит массив A

A = [2 6 9; 4 2 8; 3 0 1]

в виде следующего вектора-столбца

2
4
3
6
2
0
9
8
1

При обращении к массиву **A** с указанием единственного индекса происходит непосредственное обращение к этому вектору-столбцу. Обращение **A(3)** ссылается на третье значение в столбце; **A(7)** - на седьмое и так далее.

Если количество индексов массива больше 1, то MATLAB вычисляет индекс в столбце хранения, используя значения размерностей массива. Если двумерный массив **A** имеет размер **[d1 d2],** где **d1** - число строк, а **d2** - число столбцов, то для элемента с номером **(i, j)** его позиция в векторе хранения определяется как **(j-1)*d1+i**.

Пример. Для элемента **A(3, 2)** MATLAB вычисляет следующую позицию в векторе хранения **(2-1)*3+3 = 6**. Элементу с номером **6** соответствует значение 0.

Этот способ хранения и индексная схема распространяются и на многомерные массивы. В этом случае MATLAB использует схему постраничного объединения, чтобы создать столбец хранения. Использование единственного индекса приводит к непосредственному обращению к вектору хранения.

Если задано два индекса **(i, j),** то MATLAB вычисляет позицию описанным выше способом, причем только для первой страницы многомерного массива и при условии, что эти индексы находятся внутри диапазона размерностей исходного массива. Если задано более двух индексов, схему индексации усложняется. Если задано четыре индекса **(i, j, k, l)** для четырехмерного массиву размера **d1xd2xd3xd4**, то позиция элемента в векторе хранения вычисляется следующим образом

s = (l-1)(d3)(d2)(d1)+(k-1)(d2)(d1)+(j-1)(d1)+i

Общая формула для позиции элемента в векторе хранения, соответствующего элементу **(j1 j2 ... jn-1 jn) n**-мерного массива раз-

мера **d1xd2xd3x ... xdn**, имеет вид

s = (jn-1)(dn-1)(dn-2)...(d1)+(jn-1-1)(dn-2)...(d1)+...+(j2-1)(d1)+j1

Пример. Рассмотрим многомерный массив **C** размера **5x4x3x2**. На Рисунок 3.2 показаны форматы вывода на экран и хранения.

Вывод на экран				Порядок хранения
page(1,1) =				1
1	4	3	5	2
				5
				0
				3
2				4
	1	7	8	1
				6
				1
5				2
	6	3	2	3
				7
				3
0				5
	1	5	9	7
				5
				8
3				2
	2	7	5	9
				5
page(2,1) =				6
				7
6	2	4	2	0
				9
				1
7	1	4	9	2
				1
				0
				4
0	0	1	5	8
				4
				4

9	4	4	2
1	8	2	5

page(3,1) =

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

page(2,2) =

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

page(2,2) =

2	2	8	3
2	5	1	8

1422952252250925194815053829322509251948150

```
     5     1     5     2     5
     3     8     2     9     3
     9     4     5     3     2
     2     5     0     9     2
     5     1     9     4     8
     1     5     0     5     3
     8     2     9     3

page(3,2) =

     2     2     8     3     2
     5     1     8     5     1
     9     4     8     1     5
     0     5     3     8     2
     9     3     2     2     5
     0     9     2     5     1
     9     4     8     1     5
     0     5     3     8     2
     9     3
```

3.6 Вычисление строковых выражений

Возможность интерпретации и исполнения символьных последовательностей, записанных в виде строковых выражений, придает языку MATLAB дополнительную мощь и гибкость. Это позволяет конструировать символьные последовательности в процессе решения задачи и в зависимости от хода ее решения и таким образом придать сценарию решения задачи элементы искусственного интеллекта. Кроме того, открывается возможность обращаться по имени к ранее написанным функциям и вызывать их в зависимости от ситуации.

Eval - Вычисление строковых выражений

Синтаксис:

eval('выражение')

Описание:

Функция **eval**('выражение') интерпретирует и вычисляет текстовую строку, которая может содержать либо арифметическое выражение, либо инструкцию, либо обращение к функции.

Примеры. Вычислим текущее время **t**:

```
format rational
eval('t = clock')
t = 1997   12    7   17   24   241/100
```

Следующий программный код позволяет сформировать матрицу Гильберта порядка **n**:

```
t = '1/(i + j-1)';
n = 4;
for i = 1:n
  for j = 1:n
      G(i, j) = eval(t);
  end
end
format rational
G
G =
  1      1/2    1/3   1/4
  1/2    1/3    1/4   1/5
  1/3    1/4    1/5   1/6
  1/4    1/5    1/6   1/7
```

Feval - Вычисление функции по заданному имени

Синтаксис:

feval('<имя_функции>', x1, ..., xn)

Описание:

Функция **feval** отличается от функции **eval** тем, что она позволяет передать аргументы вызываемой функции. Используя функцию **feval** и функцию **input**, можно организовать диалог, который позволяет выбрать функцию из списка.

Пример. Пусть задан некоторый список функций **fun e.** Требуется выбрать по номеру функцию из списка и вычислить ее для заданного значения **x**:

```
fun = ['sin'; 'cos'; 'log']
fun =
   sin
   cos
   log
k = input('Указать номер функции в списке: ')
```

```
Указать номер функции в списке: 2
k =      2
x = input('Ввести значение x: ')
Ввести значение x: 1
x =     1
format rational
feval(fun(k, :), x)
ans =     429/794
```

Всякий раз, когда это возможно, рекомендуется применять функцию **feval** вместо **eval**. М-файлы, которые используют функцию **feval** выполняются быстрее и могут обрабатывться компилятором системы MATLAB .

Формирование исполняемых строк. Пользователь может сам формировать символьные строки, которые являлись бы входами функции **eval**.

Пример. Следующий фрагмент программного кода демонстрирует, как можно сформировать 3 переменных с именами **P1, P2, P3** и присвоить каждой из них различное значение:

```
for i=1 : 3
    eval(['P', int2str(i),' = i.^2'])
end
P1 =   1
P2 =   4
P3 =   9
```

3.7 Ошибки и предупреждения

В тех случаях, когда возникают различного рода ошибки, желательно, чтобы система реагировала на них специальным образом. Возможности обработки ошибок в системе MATLAB позволяют прикладной программе проверить специфические условия возникновения ошибки и выдать код ошибки в зависимости от ситуации.

Использование функций **eva**l и **lasterr**. Основные средства обработки ошибок в системе MATLAB основаны на двух функциях:

- функция **eval** позволяет не только выполнить основную функцию, но и определить вспомогательную, которая будет выполняться при возникновении шибки;
- функция **lasterr** возвращает строку, сгенерированную системой MATLAB и содержащую последнюю ошибку.

Функция **eval** позволяет обработать ошибки, используя при обращении к ней два аргумента **eval('try', 'catch')**. Здесь **'try'** - строка, содержащая исполняемое выражение, а **'catch'** - строка, содержащая обращение к функции обработки ошибки или ошибок.

Если операция, определенная строкой **'try'**, выполняется правильно, то функция **eval** просто возвращает результат; если же генерируется ошибка, то функция обработки ошибок, указанная в строке **'catch'**, определяет ошибку, генерируемую строкой **'try'**, и выполняет действия, которые позволяют избежать этой ошибки.

Функция **eval('try', 'catch')** особенно полезна в связке с функцией **lasterr**. Функция **lasterr** возвращает строку, содержащую последнее сообщение об ошибке. Применяя функцию l**asterr** внутри функции обработки ошибок, можно перехватить и проанализировать сообщение об ошибке, сгенерированное выполняемой функцией.

Пример. Следующая М-функция **catch(A, B)** для обработки ошибок использует встроенную функцию **lasterr**, чтобы проверить сообщение об ошибке, которая возникает из-за несогласованности размеров операндов. Если возникает ошибка, то программа уменьшает размеры одной из матриц:

```
function C = catch(A, B)
l = lasterr;
j = findstr(l, 'Inner matrix dimensions');
if (~isempty(j))
    [m, n] = size(A)
    [p, q] = size(B)
    if (n>p)
     A(:, p+1:n) = []
   else if (n < p)
     B(n+1:p, :) = []
   end
   C = A*B;
else
   C = 0;
end
```

Использование функции **eval** с двумя аргументами, одним из которых является функция **catch**, показано ниже:

```
clear
A = [1 2 3; 6 7 2; 0 1 5];
B = [9 5 6; 0 4 9];
eval('A*B','catch(A, B)')
m =    3
n =    3
p =    2
```

```
q =    3
A =
       1     2
       6     7
       0     1
ans =
       9    13    24
      54    58    99
       0     4     9
```

Вывод на экран ошибок и предупреждающих сообщений. Для вывода на экран терминала информации об ошибке предназначены функции **error** и **fprintf**. Функция вывода сообщения об ошибке имеет следующий синтаксис:

error('<сообщение об ошибке>')

Если эта функция ошибки вызывается внутри М-файла, то ошибка отображается в текущей строке, а выполнение М-файла прекращается.

Пример. Допустим, что в текст М-файла **myfile.m** включено условие

```
if n < 1
  error('n должно быть больше или равно 1.')
end
```

Для **n**, равного 0, следующий текст появляется на экране, а выполнение М-файла прекратится:

```
??? Error using ==> myfile
```

(n должно быть больше или равно 1).

Предупреждения системы MATLAB аналогичны сообщениям об ошибках, за исключением того, что выполнение программы не прекращается. Для вывода на экран предупреждающих сообщений предназначена функция **warning**, имеющая следующий синтаксис:

warning('<строка_предупреждения>').

Пример.

```
n = 0;
if n < 1
  warning('n должно быть больше или равно 1.')
end
```

Warning: n должно быть больше или равно 1.

3.8 Время и даты

Система MATLAB включает следующие функции для обработки времени и дат:

Тип функции	*Имя функции*	*Назначение*
Время и даты	**Now**	Текущее время и дата в форме числа
	Date	Текущая дата в форме строки
	Clock	Текущее время и дата в форме вектора
Преобразование	**Datenum**	Перевести в номер порядковой даты
	Datestr	Строковое представление даты
	Datevec	Векторное представление даты
Утилиты	**Calendar**	Календарь.
	Weekday	День недели
	Eomday	Последний день месяца
	Datetick	Дата с метками времени
Интервалы	**Cputime**	Время работы центрального процессора
	Tic	Начало отсчёта
	Toc	Конец отсчёта
	Etime	Прошедшее время

Эти функции размещены в каталоге **timefun**.

Формат даты. Система MATLAB работает с тремя форматами даты: строковым, числовым и векторным. Обычно работают с датами в строковом формате: 10-Nov-1997.

Внутреннее представление даты - числовое и соответствует количеству дней, прошедших с некоторой фиксированной даты, в качестве которой в системе MATLAB принять 1 января 0000 года. Числовой формат даты отсчитывается от полуночи, то есть 18 часов соответствует 0.75 дня. Таким образом, дате **'10-Nov-1997, 6:00 pm'** в строковом формате соответствует числовой формат **729284.75**.

Все функции, использующие дату, работают как со строковым, так и числовым форматами. При работе в командной строке предпочтителен строковый формат; если же необходимо выполнять вычисления, то эффективнее числовой формат.

Для некоторых функций системы MATLAB внутренним форматом даты является векторный формат, который состоит из следующих элементов [год месяц день час минута секунда].

В системе MATLAB есть специальные функции, которые преобразовывают форматы даты.

Пример. В системе MATLAB используется три формата даты:

Числовой формат	729300
Строковый формат	02-Oct-1996
Векторный формат	1996 10 2 0 0 0

Преобразование форматов дат. Для преобразования форматов дат предназначены следующие функции:

datenum	Преобразует строковый формат даты в числовой
datestr	Преобразует числовой формат в строковый
datevec	Преобразует числовой или строковый формат в векторный

Пример. Рассмотрим следующие преобразования форматов даты:

```
d1 = datenum('02-Oct-1996')
d1 = 729300
d2 = datestr(d1+10)
d2 = 12-Oct-1996
dv1 = datevec(d1)
dv1 = 1996 10 2 0 0 0
dv2 = datevec(d2)
dv2 = 1996 10 12 0 0 0
```

Строковый форматы даты. Функция **datenum**, которая преобразует строковое представление в числовое, имеет важное значение для организации эффективных вычислений. Функция **datenum** допус-

кает представление аргумента в одном из следующих форматов: **'dd-mmm-yyyy', 'mm/dd/yyyy', 'dd-mmm-yyyy, hh:mm:ss.ss'** . Таким образом можно сформировать до шести полей из символов и цифр:

- поле **dd** - целое число от 1 до 31;
- поле месяца mm или mmm является, либо целым числом от 1 до 12, либо строкой из трёх символов;
- поле года **yyyy** или **yy** - не отрицательное целое число; если указано только две цифры, то предполагается, что это **19yy**; если год опущен, то по умолчанию принимается текущий год;
- поля часов, минут и секунд **hh:mm:ss.ss** - необязательные. Это целые числа, разделённые двоеточиями, за которыми могут следовать атрибуты **'am' (ante meridiem** -до полудня) или **'pm' (post meridiem** - после полудня).

Пример. Все представленные даты эквивалентны, если текущий год - 1998:

'01-Jun-1998' **'Jun 01, 1998'**
'01-Jun-98' **'06/01/98'**
'01-Jun' **'06/01'**

Следующие два представления также эквивалентны:
'01-Jun-1998, 18:30'
'06/01/98/6:30 pm'

Обратите внимание, что заданный по умолчанию формат ввода даты в виде чисел следует принятому в США, то есть 6/1 - 1 июня, а не 6 января.

Если вводится вектор-столбец дат в строковом формате, то убедитесь, что все строки имеют одинаковую длину. По мере необходимости заполните их пробелами или нулями.

Форматы вывода. Функция **datestr(D,** номер формата) преобразовывает числовой формат даты **D** в один из 19 выходных строковых форматов даты и времени.

По умолчанию, для строкового представления даты используется формат день-месяц-год: 01-Mar-1996. Чтобы выбрать альтернативный формат, надо использовать один из номеров форматов, указанных в таблице.

Номер формата	***Формат***	***Описание***
0	**01-Mar-1996**	День-месяц-год

1	**01-Mar-1996 15:45:17**	День-месяц-год час:минута:секунда
2	**03/01/96**	Месяц/день/год
3	**Mar**	Месяц, (три буквы)
4	**M**	Месяц, (одна буква)
5	**3**	Месяц
6	**03/01**	Месяц/день
7	**1**	День месяца
8	**Wed**	День недели, (три буквы)
9	**W**	День недели, (одна буква)
10	**1996**	Год, (четыре цифры)
11	**96**	Год, (две цифры)
12	**Mar96**	Месяц год
13	**15:45:17**	Час: минута:секунда
14	**03:45:17 PM**	Час:минута:секунда (AM или PM)
15	**15:45**	Час:минута
16	**03:45 PM**	Час:минута (AM или PM)
17	**Q1-96**	Квартал - год
18	**Q1**	Квартал

d = '01-Mar-1996'
d = 01-Mar-1996

datestr(d)	**datestr(d, 2)**	**datestr(d, 17)**
ans = 01-Mar-1996	ans = 03/01/96	ans = Q1-96

Текущая дата и время. Функция **date** возвращает текущую дату в строковом формате:

date
ans = 02-Oct-1996

Функция **now** возвращает числовой формат текущих даты и времени:

now
ans = 729300.71
datestr(now)
ans = 02-Oct-1996 16:56:16
datestr(floor(now))
ans = 02-Oct-1996

3.9 Ввод иформации

В процессе выполнения М-файла пользователь может:

- вывести на экран запрос и ввести соответствующую информацию с клавиатуры;
- сделать паузу до нажатия клавиши;
- использовать графический интерфейс пользователя.

Формирование запроса для ввода с клавиатуры. Функция **input** выводит на экран запрос и ждет ответа пользователя. Ёе синтаксис выглядит следующим образом:

n = input('запрос')

Функция возвращает введённое с клавиатуры значение. Если пользователь вводит арифметическое выражение, функция вычисляет его и возвращает соответствующее значение. Функция полезна для реализации диалоговых прикладных программ. Функция **input** может также возвращать не числовое, а строковое выражение, вводимое пользователем. Для ввода символьного выражения необходимо добавить строку **'s'** к списку параметров функции:

Пример
name = input('Введите адрес:','s');

Задание паузы. В некоторых случаях целесообразно устанавливать паузу между отдельными шагами алгоритма, например, при выводе графиков.
Команда **pause** <без параметров> останавливает выполнение до тех, пока не будет нажата какая-нибудь клавиша. Чтобы реализовать паузу в **n** секунд, необходимо применить оператор **pause(n).**

Выход в оболочку DOS. Для обращения к программам, написанным на языках **C** или **Fortran**, можно использовать команду перехода в среду DOS, которая обозначается символ (**!**). Это позволяет выполнять автономную внешнюю программу по аналогии с выполнением М-функции. Такая функция М-функция с вызовом внешней автономной программы равносильна М-файлу, который реализует следующие условия:

- Сохраняет переменные на диске.
- Выполняет внешнюю программу, которая читает файлы данных, обрабатывает их, и записывает результаты на диск.
- Загружает обрабатываемый файл в рабочую область.

Пример. Рассмотрим М-файл **garfield.m**, который обращается к внешней функции **gareqn**:

```
function y = garfield(a, b, q, r)
   save gardata a b q r
   !gareqn
   load gardata
```

Этот М-файл:

- сохраняет входные параметры **a, b, q и r** в виде MAT-файла **gardata**, используя команду **save;**
- использует оператор перехода в среду DOS, чтобы обратиться к программе **garegn** на языке **C** или **Fortran**, которая использует переменные из рабочей области для выполнения вычислений. Программа **gareqn** записывает результаты в MAT-файл **gardata**;
- загружает MAT-файл **gardata**, чтобы сформировать выходные данные.

3.10 Повышение эффективности обработки М-файлов

Этот раздел описывает методы повышения быстродействия при выполнении программы и управление памятью:

- векторизация циклов;
- предварительное размещение векторов.

MATLAB - это язык, специально разработанный для обработки массивов и выполнения матричных операций. Всюду, где это возможно, пользователь должен учитывать это обстоятельство.

Векторизация циклов

Под векторизацией понимается преобразование циклов **for** и **while** к эквивалентным векторным или матричным выражениям. При векторизации алгоритма ускоряется выполнение М-файла.

Пример. Вот один из способов вычислить 1001 значение функции синуса на интервале [0 10], используя оператор цикла:

```
i = 0;
for t = 0:.01:10
      i = i + 1;
   y(i) = sin(t);
end
```

Эквивалентная векторизованная форма имеет вид

```
t = 0:.01:10;
y = sin(t);
```

В этом случае вычисления выполняются намного быстрее, и такой подход в системе MATLAB является предпочтительным. Время выполнения этих М-файлов можно оценить, используя команды **tic** и **toc**.

Пример. Функция **repmat** - формирование массива из частей - использует преимущество векторизации. Она имеет три входных аргумента: массив **A**, количество строк **M** и столбцов **N** для вновь создаваемого массива. Функция **repmat** возвращает массив **B** , который использует массив А в качестве основы для построения блочной матрицы с количеством блоков **MxN**:

```
A = [1 2 3; 4 5 6];
B = repmat(A, 2, 3);
B =
 1   2   3   1   2   3   1   2   3
 4   5   6   4   5   6   4   5   6
 1   2   3   1   2   3   1   2   3
 4   5   6   4   5   6   4   5   6
```

Функция **repmat** использует векторное представление для индексов, указывающих размещение блоков:

```
 function B = repmat(A, M, N)
 if nargin 2
error('Требуется, по крайней мере, 2 входа.')
elseif nargin == 2
N = M;
end
```

[m,n] = size(A);	%Определить количество строк и столбцов блока A.
mind = (1:n)'; **nind = (1:n)';**	%Векторизовать индексы в соответствии с количеством вертикальных и горизонтальных блоков.
mind = mind(:, ones(1, M)); **nind = nind(:, ones(1, N));** **B = A(mind, nind);**	%Сформировать матрицы индексов для размещения блока A.

Предварительное выделение памяти. В системе MATLAB есть возможность для существенного сокращения времени выполнения программы за счёт предварительного размещения массивов для выходных данных. Предварительное распределение избавляет от необходимости изменять массив при увеличении его размеров. Используйте соответствующие функции предварительного выделения памяти, как это показано в таблице для различных типов массивов:

Тип массива	**Функция**	**Примеры**
Массив чисел	**zeros**	**y = zeros(1, 100)** **for i = 1:100** ** y(i) = det(X^i);** **end**
Массив записей	**struct**	**data = struct([1 3], 'x', [1 3], 'y', [5 6])** **data(3).x = [9 0 2];** **data(3).y = [5 6 7];**
Массив ячеек	**cell**	**B = cell(2, 3)** **B{1, 3} = 1:3;** **B{2, 2} = 'string';**

Предварительное выделение памяти позволяет избежать фрагментации памяти при работе с большими матрицами. В ходе сеанса работы системы MATLAB, память может стать фрагментирован-

ной из-за работы механизмов динамического распределения и освобождения памяти. Это может привести к появлению большого количества фрагментов свободной памяти, и непрерывного пространства памяти может оказаться недостаточно для хранения какого-либо большого массива. Предварительное выделение памяти позволяет определить непрерывную область, достаточную для проведения всех вычислений.

Функции управления памятью

Существует несколько подходов к повышению эффективности использования памяти, рассмотренные ниже. В системе MATLAB предусмотрено пять функций для работы с памяти:

- команда **clear** - удаление переменных из оперативной памяти;
- команда **pack** - запись текущих переменных на диск и последующей их загрузкой;
- функция **quit** - по мере необходимости выход системы MATLAB с освобождением всей памяти;
- команда **save** - сохранение переменных в файле.
- команда **load** - считывание данных из файла.

Замечание. Команды **save** и **load** работают быстрее, чем утилиты ввода-вывода системы MATLAB. Эти команды оптимизированы как по скорости выполнения, так и по возможности фрагментации памяти.

На некоторых платформах команда **whos** выводит на экран количество оставшейся свободной памяти. Тем не менее полезно знать:

- поскольку команда **whos** выдаёт размер непрерывной свободной памяти, то при удалении переменной из рабочей области, объём свободной памяти не увеличивается, так как переменная занимает старшие адреса памяти;
- на компьютерах с виртуальной памятью количество оставшейся свободной памяти не отображается, поскольку ни система MATLAB, ни аппаратные средства не налагают на размер используемой памяти никаких ограничений.

Удаление функции из памяти. При загрузке MATLAB создает список имён всех M- и MEX-файлов, которые находятся в каталоге **matlab/toolbox**. Этот список сохраняется в памяти и освобождается только при создании нового списка с помощью функции **path**. Коды M- и MEX-файлов загружаются в память только при вызове соответствующей функции. Они удаляются из памяти:

- при повторном вызове или обновлении;
- при удалении командой **clear**;

- при удалении командой clear <список функций>;
- при завершение работы системы MATLAB.

Рекурсивный вызов функций. При использовании вложенных функций требуется тот же объём памяти, как при их последовательном вызове.

Пример. При вызове следующих последовательностей функций требуется одинаковый объем памяти:

1. **result = function2(function1(input99));**
2. **result = function1(input99);**
   result = function2(result);

Переменные и память. Память выделяется для переменной всякий раз, когда эта переменная не существует.

Для экономии памяти надо:

- избегать использовать одни и те же переменные в качестве входных и выходных аргументов функции, поскольку они будут передавтся ссылкой;
- после использования переменной целесообразно либо присвоить ей пустой массив, либо удалить с помощью команды clear имя переменной;
- стремиться использовать переменные повторно.

Глобальные переменные. При объявлении глобальной переменной в таблицу переменных просто помещается флаг. При этом не требуется дополнительной памяти. Например, последовательность операторов **a = 5; global a** определяет переменную a как глобальную и формируется дополнительная копия этой переменной.

Функция **clear** a удаляет переменную a из рабочей области системы MATLAB, но сохраняет её в области глобальных переменных.

Функция **clear global a** удаляет переменную a из области глобальных переменных.

Особенности платформы PC:

- На этой платформе не реализованы функции управления ресурсами системы Windows. Windows использует системные ресурсы, чтобы контролировать шрифты, окна и объекты на экране. Ресурсы могут быть исчерпаны при использовании большого количества графических окон, шрифтов или управляющих элементов графического интерфейса пользователя. Лучший способ освободить ресурсы системы - закрыть все неактивные окна. Окна в виде иконок также используют ресурсы.

- Эффективность постоянного файла для свопинга выше, чем временного.
- Обычно размер файла подкачки, вдвое превышающий размер ОП, оказывается достаточным.

Предварительное выделение памяти для самых больших массивов позволяет оптимизировать использование доступной памяти.

Сообщение " Out of Memory". Обычно сообщение **‘Out of Memory - недостаточно памяти‘** появляется, когда система MATLAB запрашивает памяти больше, чем ей доступно в текущий момент. Для оптимизации доступной памяти необходимо воспользоваться одним или несколькими методами, перечисленными выше. Если и этого оказывается недостаточным, надо:

- увеличить размер файла подкачки;
- убедиться, что нет никаких внешних ограничений на использование памяти;
- увеличить объём памяти системы;
- уменьшить размер используемых данных.

4 Отладка и профилирование М-файлов

Отладка программного кода - это процесс, в ходе которого могут быть выявлены ошибки двух видов:

- синтаксические, которые связаны с неточностью записи имен М-функций или арифметических выражений. MATLAB обнаруживает большинство синтаксических ошибок, сопровождая их сообщением об ошибке с указанием номера строки соответствующего М-файла;
- ошибки времени выполнения, которые как правило, имеют алгоритмическую природу и проявляются в том, что приводят к непредвиденным результатам.

Достаточно легко можно исправить синтаксические ошибки, которые сопровождаются сообщениями о причинах их возникновения. Ошибки времени выполнения выявить более сложно, потому что локальная рабочая область М-функции оказывается потерянной, если ошибка приводит к возврату в рабочую область системы MATLAB. Чтобы определить причину такой ошибки можно использовать один из следующих приемов:

- реализовать вывод результатов промежуточных вычислений на дисплей, удалив в соответствующих операторах точки с запя-

той, которые подавляют вывод на экран промежуточных результатов;

- добавить в М-файл команды **keyboard,** которые останавливают выполнение М-файла и разрешают проверить и изменить переменные рабочей области вызываемой М-функции. В этом режиме появляется специальное приглашение **K>>.** Возврат к выполнению функции реализуется командой return;
- закомментировать заголовок функции и выполнить М-файл как сценарий. Это позволяет проследить результаты промежуточных вычислений в рабочей области системы;
- использовать отладчик системы MATLAB.

Отладчик, реализованный в системе MATLAB, предназначен для выявления ошибок при программировании на языке MATLAB. С помощью отладчика можно просматривать состояние рабочей области в процессе выполнения, просматривать стек вызова М-функций, выполнять код М-файла построчно.

Отладчик может функционировать как в режиме командной строки, так и в режиме графического интерфейса пользователя.

Отладчик полезен для исправления ошибок во время выполнения программы именно потому, что он дает возможность отслеживать рабочие области функции и проверять или изменять значения соответствующих переменных. Отладчик позволяет устанавливать и удалять контрольные точки, то есть специальным образом помеченные строки М-файла, в которых выполнение останавливается. Это дает возможность изменять содержимое рабочей области, просматривать стек вызова М- функций и выполнять М-файл построчно.

Для того чтобы ознакомиться с возможностями отладчика, сформируем М-файл **variance.m**, который вычисляет несмещенную оценку дисперсии для элементов входного вектора. Этот файл, в свою очередь, вызывает другой М-файл sqsum, который вычисляет сумму квадратов разности элементов входного вектора и их математического ожидания:

```
function y = variance(x)
    mu = sum(x)/length(x);
    tot = sqsum(x, mu);
     y = tot/(length(x)-1);
```

Сформировать файл **sqsum.m** точно в том виде, в каком это показано ниже, вместе с преднамерено введенной ошибкой:

```
function tot = sqsum(x, mu)
   tot = 0;
   for i = 1:length(mu)
      tot = tot + ((x(i)-mu).^2);
   end
```

Замечание. Данный пример носит исключительно иллюстративный характер. Старайтесь, когда это возможно, избегать циклов **for** и везде, где это возможно, использовать векторные конструкции, чтобы повысить эффективность выполнения.

Проверим правильность работы вышеприведенных М-файлов, взяв за эталон функцию **std** системы MATLAB.
Сфорормируем следующий текстовый вектор:

v = [1 2 3 4 5];

Вычислим дисперсию,используя функцию std:

var1 = std(v).^2
var1 = 2.5000

Теперь вычислим дисперсию, используя функцию variance:

myvar1 = variance(v)
myvar1 = 1

Ответ неверен. Воспользуемся отладчиком, чтобы выявить ошибки в приведенных выше М-файлах.

4.1 Режим графического интерфейса

Рассмотрим процесс отладки с использованием отладчика **M-File Editor/Debugger**.
Для его запуска используется команда **edit** со следующим синтаксисом:

edit
edit <имя_М-файла>
edit <имя_файла.*>**

Команда **edit** запускает отладчик **M-File Editor/Debugger**.

Команда **edit** <имя_М-файла> запускает отладчик **M-File Editor/Debugger** и открывает М-файл с указанным именем.

Команда **edit** <имя_файла.***> запускает отладчик **M-File Editor/Debugger** и открывает файл с расширением ***. Экран отладчика приведен на рисунке 4.1

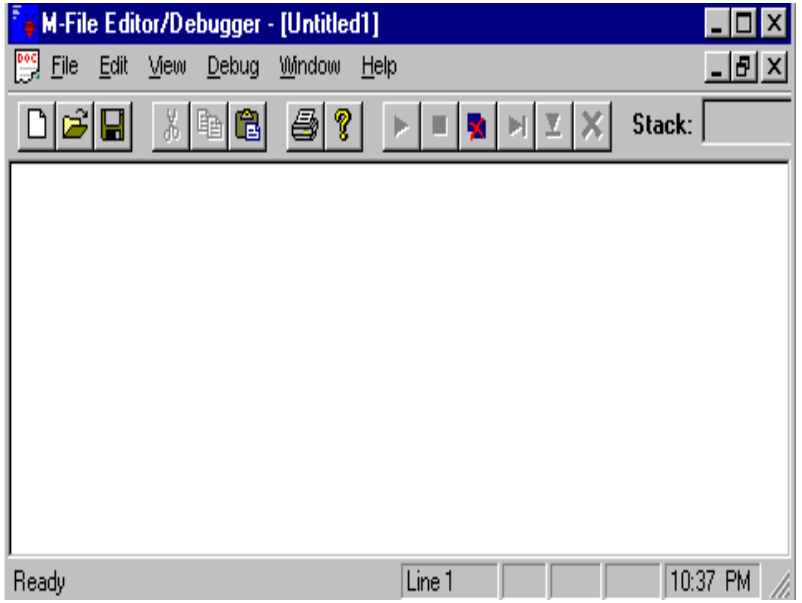


Рисунок 4.1

Иконки отладки на инструментальной панели имеют следующее назначение:

Иконка отладки	***Описание***	***Команда***
Continue	Продолжить выполнение М-файла до его завершения или до первой контрольной точки.	**dbcont**
Set/Clear Breakpoint	Установить/удалить контрольную точку в той строке, на которой находится курсор	**dbstop/ dbclear**
Single Step	Исполнить текущую строку.	**dbstep**
Step In	Исполнить текущую строку, а если она включает вызов М-функции, выполнить в ней останов.	**dbstep in**

Quit Debugging	Выйти из режима отладки.	**dbquit**

Щелчок правой кнопкой мыши в окне Отладчика приводит к появлению следующего всплывающего меню (рисунок 4.2), в котором задействованы некоторые из опций отладчика

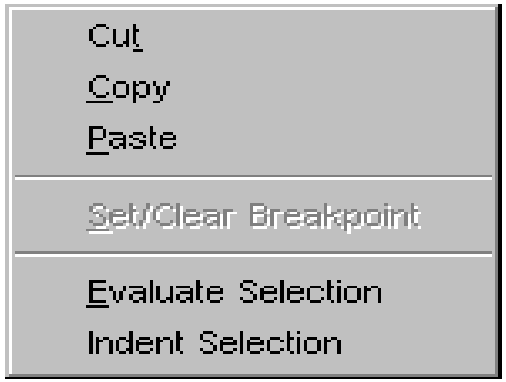


Рисунок 4.2

Установка контрольных точек. Большинство сеансов отладки начинается с установки контрольных точек. Контрольные точки позволяют приостановить выполнение М-файла в указанных строках и просмотреть или изменить значения переменных. Красный знак рядом со строкой указывает, что установлена в контрольная точка этой строке. Если строка, выбранная для контрольной точки, является невыполнимой строкой, то контрольная точка устанавливается в первой выполнимой строке.

Замечание. Меню **Debug** содержит опции, которые позволяют приостановить выполнение М-файла в следующих ситуациях:

Ситуация	***Опция***
При возникновении ошибки	**Stop if Error**
При появлении предупреждения	**Stop if Warning**
При появлении результата типа **NaN** или **Inf**	**Stop if NaN or Inf**

В начале сеанса отладки, когда нет уверенности, в каком из модулей **variance.m** или **sqsum.m** может возникнуть ошибка, видимо, разумно установить контрольные точки после вычисления среднего значения и квадрата суммы центрированных значений в модуле **variance.m**. Откроем модуль **variance.m** и установим контрольную точку в строке 4:

y = tot/(length(x)-1);

Номер строки обозначен в правой нижней части строки состояния (Рисунок 4.1). Установите контрольную точку, позиционируя курсор в строке текста, и нажмите на иконку **Set/Clear Breakpoint** инструментальной панели. Другой способ - выбрать опцию **Set/Clear Breakpoint** из меню **Debug,** либо выбрать эту же опцию из контекстного меню, вызываемого щелчком правой кнопки мыши.

Контрольные точки М-файла привязаны к его копии, которая хранится в оперативной памяти. Если из М-файла удаляются строки в процессе редактирования или используется команда clear имя М-файла, то соответствующие или все контрольные точки будут удалены.

Проверка переменных. Чтобы задействовать контрольную точку и проверить значения переменных, следует в командном окне вызвать функцию **variance(v)**. Когда выполнение М-файла будет приостановлено в контрольной точке, слева от текста появится желтая стрелка, которая указывает на строку, которая должна выполняться следующей. Желтая стрелка, направленная вниз, указывает на паузу по завершении выполнения сценария или функции. Это позволяет проверить переменные перед возвратом в вызывающую функцию.

Проверим значения переменных **mu** и **tot** из среды отладчика. Для этого выделим имя каждой переменной и вызвав контекстное меню щелчком правой кнопки мыши, выберем опцию **Evaluate Selection**. Другой способ - выбрать опцию **Evaluate Selection** из меню **View**.

В обоих случаях в командном окне высветится следующий результат:

K» mu
mu = 3
K» tot
tot = 4

Отсюда следует, что в М-функции **sqsum** есть ошибка .

Просмотр рабочих областей. Для просмотра содержимого рабочих областей следует использовать ниспадающее меню **Stack** в правом верхнем углу окна Отладчика. Оно позволяет в данном случае просматривать как содержимое рабочей области М-функции

variance.m, так и рабочей области системы MATLAB (**Base Workspace**).

Применение команды **whos** в среде М-функции **variance.m** дает следующий результат

K» whos

Name	**Size**	**Bytes**	**Class**
mu	**1x1**	**8**	**double array**
tot	**1x1**	**8**	**double array**
x 1x5	**40**	**double array**	

Grand total is 7 elements using 56 bytes

Применение команды **whos** в **среде Base Workspace** - рабочей области системы дает следующий результат

whos

Name	**Size**	**Bytes**	**Class**
ans	**1x1**	**8**	**double array**
myvar1	**1x1**	**8**	**double array**
v 1x5	**40**	**double array**	
var1	**1x1**	**8**	**double array**

Grand total is 8 elements using 64 bytes

В этом списке присутствуют переменные **v**, **var1, myvar1** и, возможно, другие, созданные пользователем в процессе работы.

Процесс отладки.Удалим контрольную точку в строке 4 модуля variance.m, поместив курсор на эту строку и выбрав соответствующую иконку инструментальной пвнели, либо опцию **Clear Breakpoint** из меню **Debug**. Другой способ - щелчком правой кнопки мыши вызвать контекстное меню и выбрать опцию **Clear Breakpoint**.

Продолжим выполнение, выбрав опцию **Continue** из меню **Debug**. Установим контрольную точку в строке 4 подпрограммы **sqsum.m** и проверим изменение индекса и результаты вычислений внутри цикла. При запуске модуля **variance.m** останов произойдет в строке 4 модуля **sqsum**:

tot = tot + (x(i) - mu).^2;

Вычислим переменную цикла **i:**

K» i

i = 1

Чтобы выполнить следующую строку, выберем опцию **Single Step** из меню **Debug** или соответствующую иконку инструментальной шкалы. После останова вычислим переменную **tot**:

K »tot

tot = 4

Снова выполним команду **Single Step** и убедимся, что модуль sqsum выполняет цикл **for i = 1:length(mu)** только один раз. Таким образом, цикл выполняется только до верхнего предела, равного длине скаляра **mu**, а не длине входного вектора **x**.

Выберем опцию **Quit Debugging** из меню **Debug** или соответствующую иконку инструментальной шкалы и завершим выполнение М-файла.

Исправим заголовок цикла:

Было	*Стало*
for i = 1:length(mu)	for i = 1:length(x)

Чтобы убедиться, что внесенное изменение приводит к правильному результату, переустановим контрольную точку в строку 4 модуля variance.m и снова запустим его на выполнение.

Выполнение приостанавливается после возврата управления из модуля **sqsum** в модуль **variance**, но перед той строкой, где используется это значение. Выведем значение **tot** после внесенных изменений:

K» tot

tot = 10

Выберем опцию **Continue** из меню **Debug** и убедимся, что полученный результат верен.

Завершение процесса отладки. Выбрать опцию **Exit Editor/Debugger** из меню **File** или соответствующую иконку инструментальной шкалы и завершить процесс отладки.

4.2 Режим командной строки

Команды отладки системы MATLAB - это набор инструментальных средств, позволяющих выполнять отладку М-файлов из командной строки. Краткое описание и синтаксис команд представлены в следующей таблице:

Описание команды	*Синтаксис команды*
Установить контрольную точку	**dbstop [in] <имя_М-файла>[at] <номер_строки>**
Удалить контрольную точку	**dbclear [in] <имя_М-файла> [at] <номер_строки>**
Остановить выполнение при возникновении предупреждения или ошибки, либо при появлении результата **NaN** или **Inf**	**dstop [if] warning** **dstop [if] error** **dstop [if] naninf** **dstop [if] infnan** **[ ] - необязательные служебные слова**
Продолжить выполнение	**dbcont**
Стек вызываемых функций	**dbstack**
Список контрольных точек данной М-функции	**dbstatus <имя_М-файла>**
Выполнить одну или несколько строк	**dbstep <количество_строк>**
Листинг М-функции с пронумерованными строками	**dbtype <имя_М-файла>**
Переход между рабочими областями сверху вниз	**dbdown**
Переход между рабочими областями снизу вверх	**dbup**
Завершить отладку	**dbquit**

Используя командную строку, можно выполнить все функции отладки. В качестве примера воспользуемся М-функциями variance и sqsum, описанными выше. Тексты этих функций представлены в следующей таблице:

function y = variance(x) **mu = sum(x)/length(x);** **tot= sqsum(x, mu);** **y = tot/(length(x)-1);**	**function tot = sqsum(x, mu)** **tot = 0;** **for i = 1:length(mu)** **tot = tot + ((x(i)-mu).^2);** **end**

Установка контрольных точек. Команда **dbstop** позволяет установить контрольную точку в определенной строке программы.

Для того чтобы установить контрольные точки в модуле **variance** после вычисления среднего значения (строка 2) и квадрата суммы центрированных значений (строка 3) следует воспользоваться следующими командами

dbstop variance 3
dbstop variance 4

Отладка кода в режиме работы с клавиатурой. Возьмем в качестве входных данных вектор v = [1 2 3 4 5]. Ожидаемые величины для среднего значения и суммы квадратов центрированных значений соответственно 3 и 10. Эти значения и должны быть получены в контрольных точках. Обратимся к М-функции из командной строки:

variance(v)

При останове в контрольной точке MATLAB автоматически переходит в режим работы с клавиатурой, который помечается приглашением K. В этом режиме можно исполнять любые команды системы MATLAB. Для продолжения выполнения программы следует использовать команду **dbcont.** При первой остановке в контрольной точке рекомендуется использовать команду **whos**, которая позволяет увидеть все переменные рабочей области модуля **variance**:

K» whos

Name	**Size**	**Bytes**	**Class**
mu	**1x1**	**8**	**double array**
x 1x5	**40**	**double array**	

Grand total is 6 elements using 48 bytes
Общее количество элементов - 6; использовано байтов - 48

Проверим значение **mu:**

K» mu

mu = 3

Воспользуемся командой **dbstep**, чтобы выполнить одну строку программы; произойдет останов в строке 4 и теперь можно проверить значение переменной tot, которое соответствует сумме квадрратов центрированных входных значений:

K >> tot

tot = 4

Это означает, что неправильные вычисления связаны с модулем **sqsum**.

Перемещение по рабочим областям. Используя функции **dbup** и **dbdown**, можно перемещаться от рабочей области одной функции к другой, а также рабочей области системы MATLAB. Для того чтобы войти в рабочую область системы MATLAB из модуля **variance**, достаточно применить команду

K» dbup

In base workspace.

K» whos

Name	**Size**	**Bytes**	**Class**
v	**1x5**	**40**	**double array**

Grand total is 5 elements using 40 bytes

Общее количество элементов - 5; использовано байтов - 40

Переменная **v**, а также, возможно, другие переменые, введенные пользователем, могут появиться в этом списке. Чтобы вернуться обратно в рабочую область модуля **variance** достаточно применить команду **dbdown**.

Листинг М-файла. В режиме клавиатуры можно использовать команду **dbtype sqsum**, чтобы вывести на экран листинг М-файла **sqsum** с пронумерованными строками:

```
K» dbtype sqsum
1   function tot = sqsum(x,mu);
2   tot = 0;
3   for i = 1:length(x)
4      tot = tot + ((x(i)-mu).^2);
5   end
```

Установим контрольные точки, чтобы проверить величины вычисляемых значений и переменной цикла:

```
K>> dbstop sqsum 4
K>> dbstop sqsum 5
```

Просмотр стека вызванных функций и продолжение выполнения. Для выхода из режима клавиатуры предназначена команда **dbquit**. Для удаления контрольных точек в модуле **variance** следует использовать команду

```
dbclear variance
```

При этом контрольные точки в модуле **sqsum** сохраняются.

Снова запустим на выполнение модуль **variance**, вновь используя в качестве входного вектор **v**:

```
variance(v)
4       tot = tot + ((x(i)-mu).^2);
```

Применим команду **dbstack**, чтобы проверить стек вызванных функций и убедиться, что М-функция **variance** действительно вызвала М-функцию **sqsum**:

```
K» dbstack
> In d:\matlab5\sqsum.m at line 4
  In d:\matlab5\variance.m at line 3
```

Проверим значение переменной цикла **i** и значение **tot**.

```
K» i
i =      1
K» tot
tot =   0
```

Выполним строку 4 и в контрольной точке строки 5 снова проверим значения **i** и **tot**:

```
K» dbstep
5      end
K» i
i =      1
K» tot
tot =   4
```

Значение **tot** изменилось; выполним строку **end** и вернемся к заголовку цикла; сделав еще один шаг, применим команду **dbstack**

```
K>> dbstep
K» dbstack
>In d:\matlab5\variance.m at line 4
```

Таким образом модуль **sqsum** выполнил цикл только один раз и возвратился в модуль **variance.**

Внимательно присмотревшись к заголовку цикла

```
for i = 1:length(mu)
```

можно придти к выводу, что в нем есть ошибка. Окончание цикла задано длиной переменной mu, а не длиной входного вектора **x.** Убедимся, что правильное значение **tot**, равное 10, приводит к верному

результату; присвоим переменной **tot** значение 10; тогда продолжив выполнение, получим верный результат:

K» tot=10
tot = 10
K>> dbcont
ans = 2.5000

Таким образом, в заголовке цикла надо заменить переменную **mu** на переменную **x**.

Завершение сеанса отладки. Чтобы завершить сеанс отладки, следует применить команду **dbquit** и таким образом возвратиться в рабочую область системы MATLAB. После этого следует внести исправления в М-функцию **sqsum.m**, так чтобы переменная цикла изменялась от 1 до **length(x),** а не от 1 до **length(mu),** то есть заголовок цикла должен иметь вид:

for i = 1:length(x)

Повторим запуск модуля **variance.m** и получим требуемый результат:

variance(v)
ans = 2.5000

4.3 Профилировщик М-файлов. Функционирование профилировщика

Один из способов повысить эффективность кодирования М-файлов - это профилировать их, то есть определить время, затрачиваемое на вычисление каждой строки.

Профилирование - это процедура измерения затрат времени на выполнение строк программы.

Результат таких измерений часто оказывается неожиданным, поскольку затраты оказываются максимальными вовсе не там, где предполагалось. Поэтому первоначальную реализацию прграммы надо делать настолько простой, насколько это возможно, а затем использовать профилировщик, чтобы выявить критические участки программы, если быстродействие действительно является главным показателем эффективности создаваемой программы. Преждевременная оптимизация может усложнить код, не обеспечив реального повышения эффективности.

Профилировщик целесообразно использовать, чтобы выявить М-функции, которые требуют большого времени, затем определить, почему и как они вызываются и поискать способы минимизации их использования. Часто бывает полезно задаться вопросом, а требуется ли вызывать М-функцию столько раз. Поскольку программы часто имеют несколько уровней, может оказаться, что созданный код вызывает наиболее трудоемкие функции неявно. Более точно, функции внутри вашего кода могут вызывать другие функции, которые требуют большого времени и могут находиться на более низких уровнях. В этом случае важно определить, какие из функций высшего уровня являются ответственным за такие обращения.

Часто профилировщик помогает выявить проблемы, которые могут быть решены:

- отказом от лишних вычислений, которые могут быть следствием невнимательности;
- корректировкой алгоритма, чтобы избежать вызова неэффективных М-функций;
- отказом от многократных повторных вычислений путем хранения результатов для последующего использования.

Конечная цель профилирования состоит в том, чтобы повысить быстродействие программы. Как только достигается сосотояние, когда наибольшее время тратится на обращения к малому числу встроенных функций, то это означает, что достигнута оптимизация кода.

Функционирование профилировщика

Команда **profile** позволяет обратиться к тем М-функциям, которые предполагается профилировать. При каждом обращении можно профилировать только одну М-функцию. В процессе профилирования ведется отсчет интервалов в 0.01 секунду, необходимых для вычисления каждой строки.

Оператор profile. Эта команда позволяет запустить профилировщик из командной строки, используя следующую форму обращения

profile <*ключевое слово*>,

где в качестве ключевого слова могут быть использованы:

- имя М-функции;
- опции **on, off, done**, **reset** для управления процессом профилирования;
- опция **report** для вывода на экран результатов профилирования текущего М-файла;
- опция **plot** для вывода результатов в графической форме в виде функции Парето.

Рассмотрим применение функции **profile** на следующем примере.

Пример:

```
profile ellipke
[k, e] = ellipke(.01:.01:.99);
profile report
```

Спецификация файла. Команда **profile** позволяет указать имя М-файла для профилирования. Она автоматически запускает профилировщик, вызывая специфицированную функцию. Задать встроенную или М-функцию можно с помощью следующей команды

profile

profile <*имя_функции*>,

где *имя_функции* может содержать указание пути доступа. В случае М-файла профилировщик подсчитывает количетво строк и нумерует их; в процессе исполнения М-файла идет подсчет времени выполнения строки с тактом 0.01 с.

Опции on и off. Эти опции позволяют запускать и приостанавливать профилировщик. Заметим, что спецификация файла в команде **profile** автоматически запускает профилировщик; в процессе профилирования команды **profile on** и **profile off** позволяют управлять этим процессом. Если делается попытка запустить профилировщик командой **profile on**, не указав имени файла, то возвращается ошибка.

Просмотр результатов профилирования. Опция **report** выводит на экран результаты профилирования. Отчет включает полную длительность исполнения функции, а также листинг пронумерованных строк с указанием затраченного времени и его процентной доли от полного времени.

Пример. Сеанс работы профилировщика. Рассмотрим следующую последовательность операций при работе с профилировщиком:

1. Специфицировать профилируемый **файл hilb.m**, который генерирует матрицу Гильберта:
   profile hilb
   Чтобы просмотреть текст М-файла воспользуйтесь командой type **hilb function**
   H = hilb(n)
2. Выполнить М-файл **hilb.m:**
   H = hilb(40);
3. Завершение профилирования:
   profile done

Полученные результаты будут зависеть от используемого компьютера.

Визуализация результатов профилирования. Функция **pareto** позволяет достаточно просто реализовать графический образ результатов профилирования.

```
profile
erfcore z = erf(0:.01:100);
profile report
```

Теперь сформируем выход профилировщика, используя команду profile без входных аргументов:

```
t = profile
```

Выход **t** это структура, которая содержит результаты профилирования функции **erfcore.m** в поле **count.** Чтобы увидеть эти результаты, надо воспользоваться функцией **pareto** в форме:

```
pareto(t.count)
```

4.4 Команды отладки и профилирования

DBSTOP - Установить контрольную точку

Синтаксис:

dbstop [in] <имя М-функции>
dbstop [in] <имя М-функции> [at] <номер строки>
dbstop [if] error
dbstop [if] naninf
dbstop [if] infnan

Описание:

Группа команд **dbstop** устанавливает режим отладки в среде системы MATLAB. Она позволяет установить контрольную точку в определенной строке М-функции или вызвать прерывание в случае возникновения предупреждения или ошибки. Если установленное условие оказалось выполненным, выводится специальное приглашение **K>>,** которое разрешает выполнить любую команду системы MATLAB.

Команда **dbstop [in] <имя М-функции>** останавливает исполнение процедуры в первой строке М-функции как только она будет вызвана. Служебное слово **in**, помещенное в квадратные скобки **[ ]**, является необязательным.

Команда **dbstop [in] <имя М-функции> [at] <номер строки>** останавливает выполнение в заданной строке указанного М-файла. В состав имени функции может быть включен путь доступа. Служебные слова **in** и **at**, помещенные в квадратные скобки **[ ]**, являются необязательными.

Команда **dbstop if error** устанавливают контрольную точку по условию, связанному с возникновением ошибки при исполнении модуля. В случае возникновения такой ситуации можно проверить переменные рабочей области и последовательность вызова функций, приведших к ошибке. Однако продолжить выполнение М-файла оказывается невозможным.

Команды **bstop if naninf** и **bstop if infnan** устанавливают контрольную точку по условию, связанному с появлением результата **NaN** или **inf.**

Команда **dbstop if warning** устанавливают контрольную точку по условию, связанному с появлением предупреждения при исполнении модуля:

- если контрольная точка определяется номером строки, то исполнение модуля прерывается перед этой строкой;
- если именем модуля, то останов перед первой исполнимой строкой;
- если условием **if error**, то останов при появлении ошибки;
- если условиями **if naninf, if infnan**, то останов при получении результата **NaN** или **inf**.

Сопутствующие команды: **DBCONT, DBSTEP, DBCLEAR, DBTYPE, DBSTACK, DBUP, DBDOWN, DBSTATUS, DBQUIT, PARTIALPATH.**

DBCLEAR - Удаление контрольных точек

Синтаксис:

dbclear in <имя М-функции>
dbclear in <имя М-функции> at <номер строки>
dbclear all in <имя М-функции>
dbclear all
dbclear if error
dbclear if warning
dbclear if naninf
dbclear if infnan

Описание. Команды из группы **dbclear** удаляют контрольные точки, установленные ранее соответствующей командой **dbstop**.

Команда **dbclear [in] <имя М-функции>** удаляет все контрольные точки в данном М-файле. Служебное слово **in**, помещенное в квадратные скобки **[ ],** является необязательным.

Команда **dbclear [in] <имя М-функции> [at] <номер строки>** удаляет контрольную точку в заданной строке данной М-функции. Служебные слова **in** и **at,** помещенные в квадратные скобки **[ ]**, являются необязательными.

Команда **dbclear all [in] <имя М-функции>** удаляет все контрольные точки в данном М-файле. Служебное слово **in**, помещенное в квадратные скобки **[ ]**, является необязательным.

Команда **dbclear all** удаляет во всех активных М-функциях все контрольные точки, за исключением тех, которые связаны с фиксацией предупреждений и ошибок.

Команды **dbclear if error** и **dbclear if warning** удаляют контрольные точки, установленные командами **dbstop if error** и **dbstop if warning**, соответственно.

Команды **dbclear if naninf** и **dbclear if infnan** удаляют контрольные точки, установленные командами **dbstop if naninf** и **dbstop if infnan**, соответственно.

Сопутствующие команды: **DBCONT, DBDOWN, DBQUIT, DBSTACK, DBSTATUS, DBSTEP, DBSTOP, DBTYPE, DBUP, PARTIALPATH**.

DBSTEP - Выполнить одну или несколько строк программы в режиме отладки

Синтаксис:

dbstep
dbstep <количество строк>
dbstep in

Описание. Группа команд **dbstep** позволяет управлять режимом отладки и включает 3 команды.

- Команда **dbstep** реализует построчное исполнение М-функции.
- Команда **dbstep <количество строк>** допускает исполнение сразу нескольких строк.
- Команда **dbstep in** связана с исполнением строки, в которой присутствует вызов другой М-функции. В последнем случае, если следующая строка содержит вызов М-функции, то применение команды **dbstep in** позволяет создать контрольную точку в первой строке вызываемой функции.

Сопутствующие команды: **DBCLEAR, DBCONT, DBDOWN, DBQUIT, DBSTACK, DBSTATUS, DBSTOP, DBTYPE, DBUP.**

DBCONT - Продолжить выполнение

Синтаксис:

dbcont

Описание. Команда **dbcont** вызывает исполнение М-функции до следующей контрольной точки, установленной командами **dbstop** или **dbstep**.

Сопутствующие команды: **DBCLEAR, DBDOWN, DBQUIT, DBSTACK, DBSTATUS, DBSTEP, DBSTOP, DBTYPE, DBUP .**

DBSTACK - Стек вызываемых М-функций

Синтаксис:

dbstack [ST, I] = dbstack

Описание. Команда **dbstack** выводит на терминал номера строк и имена вызванных М-функций, начиная от контрольной точки и до самого внешнего модуля, за исключением М-сценария (**Script**-файла). Оператор **[ST, I] = dbstack** возвращает стек вызванных функций в виде массива записей (структуры) **ST** размера **m?1** с полями **ST.line**, **ST.name**. Текущей рабочей области присваивается индекс **I=1**; при однократном использовании команды **dbup** индекс **I** увеличивается на **1**, так что самый высокий индекс имеет базовая рабочая область системы MATLAB.

Пример. Рассмотрим использование команды **dbstack** при останове в некоторой контрольной точке:

```
K" dbstack
> In d:\matlab5\sqsum.m at line 3
  In d:\matlab5\variance.m at line 3
```

Рассмотрим использование оператора **[ST, I] = dbstack** при останове в той же контрольной точке:

```
[ST, I] = dbstack
ST =
2x1 struct array with fields:
   line
   name
I =   1
```

Выведем содержимое полей **ST.name** и **ST.line**

```
K"ST.name
ans = d:/matlab5/sqsum.m
ans = d:/matlab5/variance.m
K" ST.line
ans = 3
ans = 3
```

Сопутствующие команды: **DBCLEAR, DBCONT, DBDOWN, DBQUIT, DBSTATUS, DBSTEP, DBSTOP, DBTYPE, DBUP.**

DBUP - Переход между рабочими областями снизу вверх

Синтаксис:

dbup

Описание. Команда **dbup** осуществляет переход в стеке вызываемых М-функций снизу вверх. Все переменные доступны для про-

смотра (команды **who, whos**) и обработки, и можно проследить, как они изменялись вплоть до значения, которое было передано отлаживаемому модулю. Рабочая область переменных самого внешнего модуля называется базовоой рабочей областью. Для продолжения отладки выполнения команды, обратной **dbup** (команда **dbdown**), не требуется.

Сопутствующие команды: **DBCLEAR, DBCONT, DBDOWN, DBQUIT, DBSTACK, DBSTATUS, DBSTEP, DBSTOP, DBTYPE**.

DBDOWN - Переход между рабочими областями сверху вниз

Синтаксис:

dbdown

Описание. Команда **dbdown** применяется совместно с командой **dbup** для перемещения между рабочими областями вызываемых модулей. Эта команда противоположна по своему действию команде **dbup**. Команда **dbdown** реализует перемещение только в том случае, если выполнена хотя бы одна команда **dbup.**

Сопутствующие команды: **DBCLEAR, DBCONT, DBQUIT, DBSTACK, DBSTATUS, DBSTEP, DBSTOP, DBTYPE, DBUP.**

DBSTATUS - Список контрольных точек данной М-функции

Синтаксис:

dbstatus
dbstatus <имя М-функции>
s = dbstatus

Описание. Команда **dbstatus** выводит на терминал список всех контрольных точек, включая контрольные точки, связанные с ошибками и предупреждениями, а также с результатами вычислений вида **NaN** и **Inf**.

Команда **dbstatus <имя М-функции>** выводит на терминал список всех контрольных точек, определенных для данной М-функции. Эту команду можно использовать в формах **dbstatus class/<имя М-функции>, dbstatus private/<имя М-функции>, dbstatus private/class/<имя М-функции>,** чтобы создать список контрольных точек для методов, частных функций или частных методов соответственно. Кроме того, во всех этих случаях можно связывать имя функции с подфункцией в форме **dbstatus <имя М-функции>/<имя подфункции>.** Оператор **s = dbstatus** возвращает информацию о контрольных точках в виде массива записей (структуры) размера **m?1** с полями **s.name, s.line, s.cond**, которые содержат имена М-функций, вектор номеров строк с контрольными точками, строки условий (**error, warning** или **naninf)**.

Пример. Получим список всех контрольных точек:

K” dbstatus
Breakpoint for d:\matlab5\sqsum.m is on line 3.
Breakpoints for d:\matlab5\variance.m are on lines 3, 4.

Получим список всех контрольных точек в виде массива записей размера 2x1:

K” s = dbstatus
s =
2x1 struct array with fields:
name
line
cond

Выведем содержимое полей **s.name, s.line, s.cond:**

K” s.name	**K” s.line**	**K” s.cond**
ans = d:\matlab5\sqsum.m	**ans = 3**	**ans = ''**
ans = d:\matlab5\variance.m	**ans = 3 4**	**ans = ''**

Получим список контрольных точек, когда в состав М-функции **variance** включена подфункция **sqsum**:

K” dbstatus
Breakpoint for d:\matlab5\variance.m is on line 4.
Breakpoint for d:\matlab5\variance.m (sqsum) is on line 7.

Получим список контрольных точек подфункция **sqsum**:

K” dbstatus variance/sqsum
Breakpoint for d:\matlab5\variance.m (sqsum) is on line 7.

Сопутствующие команды: **DBCLEAR, DBCONT, DBDOWN, DBQUIT, DBSTACK, DBSTEP, DBSTOP, DBTYPE, DBUP.**

DBTYPE - Текст М-функции с указанием номеров строк

Синтаксис:

dbtype<имяМ-функции>
dbtype <имя М-функции><начало> : <конец>

Описание. Команда **dbtype <имя М-функции> <начало> : <конец>** позволяет вывести на терминал текст М-функции с указанием номеров строк; для вывода части текста следует указать диапазон номеров выводимых строк; для вывода одной строки достаточно указать ее номер.

Сопутствующие команды: **DBCLEAR, DBCONT, DBDOWN, DBQUIT, DBSTACK, DBSTATUS, DBSTEP, DBSTOP, DBUP, PARTIALPATH**.

DBQUIT - Выход из режима отладки

Синтаксис:

dbquit

Описание. Команда **dbquit** немедленно прекращает режим отладки и возвращает управление базисному модулю. Исполнение текущего М-файла прерывается, результаты не возвращаются. Все контрольные точки сохраняются. Если основной модуль является М-сценарием, то его выполнение также прерывается, появляется сообщение об ошибке и управление передается в среду системы MATLAB.

Пример. Выход из режима отладки в случае, когда внешний модуль **mslsnae2.m** является М-сценарием (**Script**-файлом):

```
K” dbstack
In d:\toolbox\snae\pencil.m at line 20
In d:\toolbox\snae\msnae2.m at line 34
K” dbquit
Error in ==> d:\toolbox\snae2\mslsnae2.m
On line 70 ==> [x, y, err] = msnae2(Axy, P, sx, sy, nv)
 ”
```

Сопутствующие команды: **DBCLEAR, DBCONT, DBDOWN, DBSTACK, DBSTATUS, DBSTEP, DBSTOP, DBTYPE, DBUP.**

PROFILE - Измерить и вывести на экран профиль исполняемого М-файла

Синтаксис:

profile имя М-функции
profile report _ | n | frac
profile plot
profile on | off | reset | done
info = profile

Описание. Утилита профилировщика помогает отладить и оптимизировать М-функции, фиксируя время выполнения каждой строки программы. Утилита создает вектор измерений для каждой строки программы в профилируемом М-файле. При выполнении программы профилировщик обновляет вектор измерений с учетом времени, затрачиваемого на выполнение соответствующей строки.

Команда **profile имя М-функции** стартует профилировщик для заданной функции, имя которой должно быть именем М-файла, возможно, с указанием частичного пути доступа.

Команда **profile report _ | n | frac** выводит на экран либо полный отчет о профиле М-файла (в отсутствии каких-либо опций), либо только об **n** строках с наибольшим временем исполнения, либо о тех строках, доля затраченного времени для которых от общего времени выполнения превышает значение **frac** из диапазона от 0 до 1.

Команда **profile plot** выводит на экран результаты профилирования в виде диаграммы Парето.

Команды **profile on ? profile off** запускают или приостанавливают процесс профилирования, соответственно; команда **profile reset** очищает вектора измерений, не отключая профилровщика; команда **profile done** завершает работу профилировщика и удаляет сопутствующие данные.

Оператор **info = profile** возвращает результаты профилирования в виде структуры со следующими полями: **file** - Полный путь доступа к профилируемой функции. **function** Имя профилируемой функции. **interval** Интервал измерения, измеренный в секундах. **Count** - Вектор измерений. **state** - Состояние профилировщика: **on** - активен; **off** - не активен. Профилировщик отслеживает количество интервалов, затраченных на выполнение встроенной функции. Поведение профилировщика зависит от свойств корневого объекта и может управляться с помощью команд **set** и **get**. Одновременно профилировщик может обрабатывать только один М-файл.

Сопутствующие команды: **DEBUG, PROFSUMM**.

5 Многомерные массивы

Многомерные массив - это расширение понятия числового массива, когда количество измерений (размерность) становится больше двух. Многомерные массивы применяются при описании страниц двумерных данных.

MATLAB поддерживает следующие функции при работе с многомерными массивами:

Функция	***Назначение***
cat	Сформировать многомерный массив.
ndims	Определить размерность многомерного массива.

ndgrid	Сгенерировать сетку для многомерной функции.
permute, ipermute	Переставить размерности.
shiftdim	Изменить размерность массива.
squeeze	Удалить одну из размерностей.

Пользователь может расширить состав этих функций, создавая специальные М-файлы для обработки конкретных данных.

5.1 Определение многомерного массива

Многомерные массивы в системе MATLAB являются расширением обычных двумерных массивов, которые имеют две размерности: размерность строк и размерность столбцов. Ниже приведена структура 2-мерного массива размера 6x4:

	столбцы			
строки	(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)
	(5,1)	(5,2)	(5,3)	(5,4)
	(6,1)	(6,2)	(6,3)	(6,4)

Доступ к элементам двумерного массива достигается путем введения двух индексов -строки и столбца.

Рассмотрим 3-мерный массив; для обозначения элементов требуется три индекса: первый - для обозначения строк; второй -

столбцов;третий - страниц. Для любой размерности многомерного массива выше 2 используется понятие страницы. Ниже приведена структура 3-мерного массива размера 6x4x3:

Страница 3

Столбцы			
(1,1,3)	(1,2,3)	(1,3,3)	(1,4,3)
(2,1,3)	(2,2,3)	(2,3,3)	(2,4,3)
(3,1,3)	(3,2,3)	(3,3,3)	(3,4,3)
(4,1,3)	(4,2,3)	(4,3,3)	(4,4,3)
(5,1,3)	(5,2,3)	(5,3,3)	(5,4,3)
(6,1,3)	(6,2,3)	(6,3,3)	(6,4,3)

Строки

Страница 2

Столбцы			
(1,1,2)	(1,2,2)	(1,3,2)	(1,4,2)
(2,1,2)	(2,2,2)	(2,3,2)	(2,4,2)
(3,1,2)	(3,2,2)	(3,3,2)	(3,4,2)
(4,1,2)	(4,2,2)	(4,3,2)	(4,4,2)
(5,1,2)	(5,2,2)	(5,3,2)	(5,4,2)
(6,1,2)	(6,2,2)	(6,3,2)	(6,4,2)

Строки

Столбцы

Страница 1	Строки			
	(1,1,1)	(1,2,1)	(1,3,1)	(1,4,1)
	(2,1,1)	(2,2,1)	(2,3,1)	(2,4,1)
	(3,1,1)	(3,2,1)	(3,3,1)	(3,4,1)
	(4,1,1)	(4,2,1)	(4,3,1)	(4,4,1)
	(5,1,1)	(5,2,1)	(5,3,1)	(5,4,1)
	(6,1,1)	(6,2,1)	(6,3,1)	(6,4,1)

При расширении размерности массива добавляются новые индексы. Так 4-мерный массив характеризуется 4 индексами: первые 2 определяют строку и столбец, вторые 2 - третью и четвертую размерность данных. Есть возможность построить графическое изображение структуры такого массива. Ниже приведена структура 4-мерного массива размера 3x3x3x2:

A(:,:,3,1)=

5	5	5
5	5	5
5	5	5

A(:,:,3,2)=

1	0	1
1	1	0
0	1	1

A(:,:,2,1)=

1	0	4
3	5	6
9	8	7

A(:,:,2,2)=

9	8	7
6	5	4
3	2	1

A(:,:,1,1)=

5	7	8
0	1	9
4	3	6

A(:,:,3,1)=

1	2	3
4	5	6
7	8	9

Возможно дать графическое изображение 5-мерного массива, повторяя эту структуру в вертикальном направлении.

5.2 Формирование многомерных массивов

Для создания многомерных массивов можно использовать те же приемы индексирования и применения встроенных функций, которые используются при создании двумерных массивов. Тем не менее в системе MATLAB добавлена специальная функция **cat**, которая позволяет сформировать структуру многомерного массива. Таким образом, можно определить три подхода к созданию многомерных массивов:

- использование индексов;
- использование встроенных функций для формирования массивов специального вида;
- использование функции **cat.**

Использование индексов. Один из способов формирования многомерного массива вытекает из его представления как совокупности 2-мерных массивов, размещаемых на новых страницах. Он состоит в том, чтобы просто добавлять новые размерности для формирования нужных страниц (3-ю, 4-ю, 5-ю и т. д.).

Пример. Сначала сформируем двумерный массив A:

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

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

Этот массив имеет размерность 2 и размер 3x3. Добавим новую страницу в третьей размерности массива с помощью следующего оператора присваивания

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

     5     7     8
     0     1     9
     4     3     6
A(:, :, 2) =
```

1 0 4
3 5 6
9 8 7

Сформирован массив **A** размерности 3 и размера 3x3x2. Можно продолжить добавлять строки, столбцы и страницы с целью формирования многомерных массивов различных размерностей и размеров.

Для изменения размерности многомерного массива нужно:

- уменьшить или увеличить соответствующий индекс;
- присвоить одному или нескольким ранее не существовавшим элементам некоторые значения; в этом случае для числовых массивов произойдет увеличение количества строк, столбцов или страниц, поскольку структура числового массива требует одинакового количества строк, одинакового количества столбцов, одинакового количества страниц в соответствующих размерностях.

Реализованный в системе MATLAB механизм присваивания скаляра массиву вместе с механизмом индексации позволяет заполнять целые страницы массива, используя одно число.

Пример.
A(:, :, 3) = 5

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

```
A(:,:,2)=
     1     0     4
     3     5     6
     9     8     7
```

```
A(:,:,1)=
     5     7     8
     0     1     9
     4     3     6
```

Дополним массив А четвертой размерностью, введя

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]

```
A(:,:,3,1)=
     5     5     5
     5     5     5
     5     5     5
```

```
A(:,:,3,2)=
     1     0     1
     1     1     0
     0     1     1
```

```
A(:,:,2,1)=
     1     0     4
     3     5     6
     9     8     7
```

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

```
A(:,:,1,1)=
     5     7     8
     0     1     9
     4     3     6
```

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

Использование встроенных функций. Такие встроенные функции системы MATLAB, как **randn, ones** и **zeros**, могут быть использованы для формирования многомерных массивов, поскольку каждый аргумент такой функции определяет размер соответствующего измерения.

Пример. Сформировать 3-мерный массив нормально распределенных случайных чисел размера 4x3x2:

B = randn(4, 3, 2)

B(:, :, 1) =

```
    -0.4326   -1.1465    0.3273
    -1.6656    1.1909    0.1746
     0.1253    1.1892   -0.1867
     0.2877   -0.0376    0.7258
B(:, :, 2) =
    -0.5883    1.0668    0.2944
     2.1832    0.0593   -1.3362
    -0.1364   -0.0956    0.7143
     0.1139   -0.8323    1.6236
```

Чтобы сформировать массив, заполненный константой, удобно применить функцию repmat. Эта функция использует заданный массив (в случае константы - размера 1x1) для формирования многомерного массива в соответствии с его размерностью:

```
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
```

***Замечание**:* Если хотя бы одна из размерностей массива имеет значение 0, то это означает, что многомерный массив - пустой.

Использование функции cat. Применение функции **cat** существенно упрощает формирование многомерных массивов, поскольку позволяет задать размещение 2-мерных массивов вдоль указанной размерности, используя следующий синтаксис:

```
B = cat(dim, A1, A2 ...),
```

где **dim** - номер размерности, вдоль которой размещаются массивы; **A1, A2, …..** - список 2-мерных массивов.

Пример. Сформируем 3-мерный массив, который объединяет два 2-мерных массива размера 2x2:

B = cat(3, [2 8; 0 5], [1 3; 7 9])

B(:, :, 1) =

2	**8**
0	**5**

B(:, :, 2) =

1	**3**
7	**9**

Функция **cat** допускает использование любых комбинаций существующих и вновь вводимых данных.

Пример. Сформировать 4-мерный массив **D** с помощью следующей последовательности операторов **cat**:

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

D(:,:,2,1)=

7	**1**
8	**4**

D(:,:,2,2)=

5	**6**
2	**1**

D(:,:,2,3)=

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

D(:,:,1,1)=

9	**2**
6	**5**

D(:,:,1,2)=

3	**5**
0	**1**

D(:,:,1,3)=

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

Функция **cat** автоматически добавляет промежуточные индексы, равные 1, если в этом возникает необходимость.

Пример. Сформировать 4-мерный массив, разместив вдоль четвертой размерности два массива размера 2x2:

C = cat(4, [1 2; 4 5], [7 8; 3 2])

Таблица 5.7 - 4-мерный массив размера 2x2x1x2

C(:,:,1,1)=

1	2
4	5

C(:,:,1,2)=

7	8
3	2

Сформированный 4-мерный массив имеет размер 2x2x1x2. Если бы аргумент **dim** был равен 5, то был бы сформирован 5-мерный массив размера 2x2x1x1x2. С учетом принятых допущений его можно было бы изобразить следующим образом

C(:,:,1,1,2)=

7	**8**
3	**2**

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

1	**2**
4	**5**

Характеристики многомерного массива. Для получения информации о характеристиках многомерного массива используются следующие функции:

- whos - информация о типе массива и используемой памяти;
- ndims - количество размерностей;
- size - размер массива.

Извлекаемая информация	**Функция**	**Пример**

Имя, размер, количество байтов используемой памяти, тип массива	**whos**	**Name Size Bytes Class** **A** 2x2x2 64 **double array** **B** 2x2x2 64 **double array** **C** **4-D** 64 **double array** **D** **4-D** 192 **double array** **Grand total is 48 elements using 384 bytes**
Количество размерностей	**ndims**	**ndims(D)** **ans = 4**
Размер массива	**size**	**size(D)** ans = 2 2 2 3

5.3 Работа с многомерными массивами

Много приемов, связанных с работой с двумерными массивами, переносится на многомерные. В этом разделе описано, как применяется техника индексации и переопределения размеров к многомерным массивам.

В качестве сквозного примера будем рассматривать 3-мерный массив нормально распределенных случайных целых чисел **nddata** размера 4x5x3:

nddata = fix(8*randn(4, 5, 3))

nddata(:, :, 1) =

-3	**-9**	**2**	**-4**	**8**
-13	**9**	**1**	**17**	**0**
1	**9**	**-1**	**-1**	**0**

```
     2     0     5     0     -6

nddata(:, :, 2) =

     2     -5    -11    6     9
    -10    6     4     5     -9
     5     10    -3     10    0
     12    -12   5     5     -1

nddata(:, :, 3) =

    -12    -6    -17    4     3
     2     4     0     13    -8
    -8     1     -8    4     0
     11    -7    4     -5    0
```

Индексация

Чтобы получить доступ к элементу (3, 2) на странице 2 массива **nddata**, надо использовать обращение **nddata(3, 2, 2)**.

В качестве индексов можно использовать вектор, каждый элемент которого должен быть допустимым индексом многомерного массива. Чтобы получить доступ к трем элементам (2, 1), (2, 3) и (2, 4) на странице 3 массива **nddata**, надо использовать обращение

```
nddata(2, [1 3 4], 3)
ans =     2     0    13
```

Индексация столбцов многомерных массивов. Индексация столбцов, широко применяемая в системе MATLAB, может быть распространена и на многомерные массивы.

Пример. Чтобы получить доступ к столбцу 3 на странице 2 массива **nddata**, надо воспользоваться следующим оператором

```
nddata(:, 3, 2)
ans =
    -11
     4
     -3
```

5

Используя индексацию столбцов, можно извлечь следующий массив размера 2x2 со страницы 1 массива **nddata**:

```
nddata(2:3,2:3,1)
ans =
   9
   1
   9
  -1
```

Индексация столбцов может быть использована как в правой, так и в левой частях оператора присваивания.

Пример. Сформируем 2-мерный массив **C** размера 4x4, заполненный нулями:

```
C = zeros(4, 4)
C =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Теперь разместим в центре массива **C** подмассив **nddata(2:3, 1:2, 2)** размера 2x2, извлеченный из многомерного массива **nddata**:

```
C(2:3, 2:3) = nddata(2:3, 1:2, 2)
C =
     0     0     0     0
     0    10    60     0
     0     5    10     0
     0     0     0     0
```

Неоднозначность многомерной индексации. Некоторые типы операторов присваивания, например, **A(:, :, 1) = 1:10,** в случае многомерных массивов не обеспечивают однозначности, поскольку не содержат достаточной информации для выполнения оператора присваивания. В данном случае делается попытка присвоить 1-мерный массив 2-мерному подмассиву массива **A.** Система MATLAB в таких случаях выдает сообщение об ошибке. Чтобы обеспечить однознач-

ность, необходимо убедиться, что данные и массив-адресат имеют согласованные размеры.

Пример. Правильное присваивание:

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

A = 1 2 3 4 5 6 7 8 9 10

Переопределение размеров

Изменение размеров и структуры многомерного массива может происходить в двух случаях :

- при добавлении или удалении элементов;
- при переопределении размеров или размерностей, причем в этом случае общее количество элементов должно оставаться неизменным.

Для выполнения второй группы операций предназначена функция **reshape**, синтаксис которой для многомерного случая имеет вид

B = reshape(A, [s1 s2 s3 ...])

где **s1, s2**,.... - новые значения размерностей.

Пример. Рассмотрим переопределение размеров и размерностей для многомерного массива **nddata**

B = reshape(nddata, [6 10])

B =

-3	**9**	**-4**	**0**	**-5**	**-3**	**9**	**-8**	**-17**	**4**
-13	**0**	**17**	**-6**	**6**	**5**	**-9**	**11**	**0**	**-5**
1	**2**	**-1**	**2**	**10**	**6**	**0**	**-6**	**-8**	**3**
2	**1**	**0**	**-10**	**-12**	**5**	**-1**	**4**	**4**	**-8**
-9	**-1**	**8**	**5**	**-11**	**10**	**-12**	**1**	**4**	**0**
9	**5**	**0**	**12**	**4**	**5**	**2**	**-7**	**13**	**0**

C = reshape(nddata, [5 4 3])

C(:,:,3)=

-12	**4**	**-8**	**-5**
2	**1**	**4**	**3**
-8	**-7**	**4**	**-8**
11	**-17**	**13**	**0**
-6	**0**	**4**	**0**

```
C(:,:,2)=
2     6     -3    5
-10   10    5     9
5     -12   6     -9
12    -11   5     0
-5    4     10    -1
```

```
C(:,:,1)=
-3    9     -1    0
-13   9     5     8
1     0     -4    0
2     2     17    0
-9    1     -1    -6
```

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

```
D(:,:,2,1)=
9   2   -1
0   1   5
```

```
D(:,:,2,2)=
0   2   5
-6  -10 12
```

```
D(:,:,2,3)=
-3  6   10
5   5   5
```

```
D(:,:,2,4)=
-8  -6  1
11  4   -7
```

```
D(:,:,2,5)=
4   3   0
-5  -8  0
```

```
D(:,:,1,1)=
-3  1   -9
0   1   5
```

```
D(:,:,1,2)=
-4  -1  8
17  0   0
```

```
D(:,:,1,3)=
-5  10  -11
6   -12 4
```

```
D(:,:,1,4)=
9   0   -12
-9  -1  2
```

```
D(:,:,1,5)=
-17 -8  4
0   4   13
```

Удаление размерностей размера 1x1. Система MATLAB создает размерности размера 1x1, если такая спецификация указывается точно при создании или переопределении массива, либо если в процессе вычислений возникает массив размера 1x1.

Пример.

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

```
B(:,:,1,1)=
5   5   5
5   5   5
```

```
B(:,:,1,2)=
5   5   5
5   5   5
```

```
B(:,:,1,3)=
5   5   5
5   5   5
```

```
B(:,:,1,4)=
5   5   5
5   5   5
```

size(B)

ans = 2 3 1 4

Функция **squeeze** удаляет из многомерного массива измерение размера 1x1:

C = squeeze(B)

```
C(:,:,4)=
5     5     5
5     5     5
```

```
C(:,:,3)=
5     5     5
5     5     5
```

```
C(:,:,2)=
5     5     5
5     5     5
```

```
C(:,:,1)=
5     5     5
5     5     5
```

size(C)

ans = 2 3 4

Замечание. Функция **squeeze** транспонирует вектор-строку в вектор-столбец; вектор-столбец функция **squeeze** оставляет без изменения.

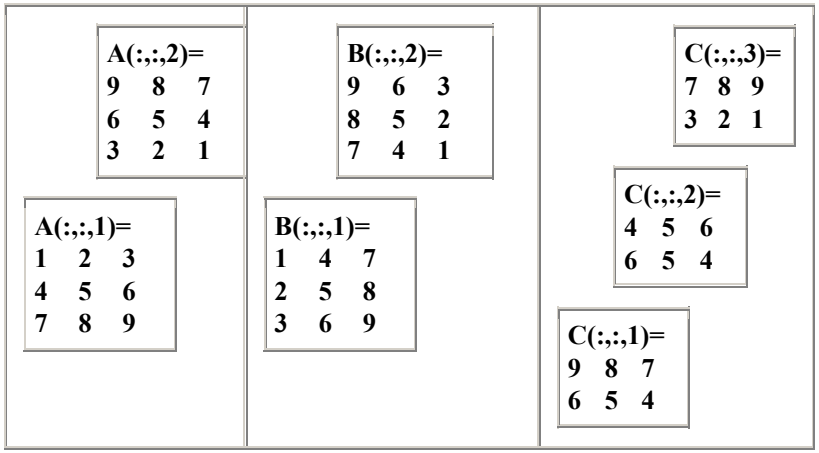
Перестановки размерностей. Функция **permute** позволяет выполнить перестановку размерностей

B = permute(A, dims);

где **dims** - вектор, который задает новый порядок следования размерностей. При этом индекс 1 соответствует строкам, индекс 2 - столбцам, индекс 3 - страницам и т. д.

Пример. Применения функции **permute:**

A	**B = permute(A,[2 1 3]**	**C = permute(A,[3 2 1]**



Функция **ipermute** обратна по отношению к функции **permute**. Вводя массив **A** и вектор перестановок **v**, функция **ipermute** формирует такой массив **B**, что функция **permute(B, v)** возвращает **A**.

Пример. В результате выполнения следующих операторов формируется массив **E**, который совпадает с исходным массивом **C**:

а) **D = ipermute(C, [1 4 2 3])**

```
D(:,:,1,1)=
5     5     5     5
5     5     5     5

D(:,:,1,2)=
5     5     5     5
5     5     5     5

D(:,:,1,3)=
5     5     5     5
5     5     5     5
```

б) **E = permute(D, [1 4 2 3])**

```
E(:,:,4)=
5     5     5
5     5     5

E(:,:,3)=
5     5     5
5     5     5

E(:,:,2)=
5     5     5
5     5     5
```

```
E(:,:,1)=
5     5     5
5     5     5
```

Обратите внимание, что функции **permute** и **ipermute** используют один и тот же вектор перестановок.

Вычисления на многомерных массивах

Многие функции системы MATLAB допускают использование многомерных массивов в качестве входных аргументов. Есть функции, которые могут использовать только отдельные размерности, соответствующие матрицам, векторам или отдельным элементам.

Функции, работающие с векторами. Функции, использующие векторы, такие как **sum**, **mean**, по умолчанию используют в качестве аргумента первую размерность многомерного массива, значение которой не равно 1. Большинство таких функций позволяют пользователю указать ту конкретную размерность, которую надо использовать в качестве входа. Однако есть и некоторые исключения. Например, функция **cross** использует в качестве входного аргумента первую размерность с длиной вектора, равной 3.

В ряде случаев могут возникать и другие ограничения на входные аргументы. Например, может потребоваться, чтобы массивы имели одинаковые размеры. В любом случае это означает, что при работе с многомерными массивами надо внимательно следить за требованиями, которые предъявляет используемая функция к входным аргументам.

Функции, работающие с отдельными элементами. Функции системы MATLAB, которые оперируют с отдельными элементами двумерного массива, будут точно также работать и с элементами многомерного массива. В первую очередь, это все элементарные функции. Например, функция **sin** всегда возвращает массив того же размера, как и массив входа. Каждый элемент выходного массива является синусом соответствующего элемента входного массива.

Точно также операторы отношения и логические операторы используют отдельные элементы многомерного массива. Если один из операндов - скаляр, а другой массив, то исполняемый оператор сопоставляет скаляр с каждым элементом массива.

Функции, работающие с матрицами. Функции линейной алгебры и матричные функции из каталога **matfun** не допускают многомерные массивы в качестве входных аргументов. Если такое происходит, то появляется сообщение об ошибке. Использовать такие матричные функции можно только по отношению к двумерным подмножествам многомерного массива.

Пример. Сформируем 3-мерный массив **A**, состоящий из трех 2-мерных массивов размера 3x3:

```
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])
```

Таблица 5.14 - 3-мерный массив размера 3x3x3

A(:,:,3)=		
6	4	7
6	8	5
5	4	3

A(:,:,2)=		
0	3	2
8	8	4
5	3	5

A(:,:,1)=		
1	2	3
9	8	7
4	6	5

Применение функции вычисления собственных значений **eig** к массиву в целом вызовет ошибку:

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

(Входные аргументы должны быть 2-мерными)

Однако ее можно применять к 2-мерным множествам.

Пример. Вычислим собственные значения для второго из трех массивов:

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

12.9129

2.7131

Для того чтобы вычислить собственные значения для матрицы, составленной из вторых строк каждого массива, надо использовать функцию **squeeze**, которая преобразует 3-мерное подмножество **A(2, :, :)** размера 1x3x3 в матрицу размера 3x3.

Функция **eig(A(2, :, :))** выдает ошибку

eig(A(2,:,:))
??? Error using ==> eig
Input arguments must be 2-D
(Входные аргументы должны быть 2-мерными)

Функция **eig(squeeze(A(2, :, :)))** вычисляет собственные значения (но не собственные векторы - вспомните о транспонировании строк при использовании функции **squeeze**) абсолютно правильно

eig(squeeze(A(2, :, :)))
ans =
21.2293
0.3854+ 1.5778i
0.3854- 1.5778i

Организация данных в многомерных массивах

Существует два подхода к тому, как организовать данные в многомерный массив:

1. Страничная интерпретация. В этом случае за основу берутся 2-мерные массивы, которые считаются размещенными на страницах. Множество таких страниц может затем быть организовано в 3-мерные, 4-мерные и т. д. массивы.
2. Пространственная интерпретация (многомерные данные). В этом случае рассматриваются измерения физических величин (температуры, давления и т. п.) в точках трехмерного пространства.

Первый подход использовался на протяжении всего раздела. Второй может представлять интерес при решении уравнений в частных производных, а также физических задач и измерений.

Применение последнего подхода проиллюстрируем на примере измерения температуры в равноотстоящих точках в пределах некоторого объема. В этом случае каждое измерение привязано к определенной точке 3-мерного пространства и такие данные могут быть организованы в 3-мерный массив **TEMP**:

67.9°	**68.0°**	**67.9°**
67.8°	**67.8°**	**67.9°**
67.7°	**67.9°**	**67.7°**

67.9°	**68.0°**	**68.0°**
67.7°	**67.8°**	**67.7°**
67.8°	**67.7°**	**67.5°**

68.0°	**68.0°**	**67.8°**
67.9°	**67.8°**	**67.6°**
67.8°	**68.0°**	**67.6°**

Для вычисления среднего значения температуры в этом объеме можно использовать оператор **mean(mean(mean(TEMP))).** Для вычисления значений температуры посередине этого объема - элементы (2, 2) на каждой странице - надо применить оператор **B = TEMP(2, 2, :).**

5.4 Команды и функции обработки многомерных массивов

CAT - Объединение массивов

Синтаксис:

C = cat(dim, A, B)

C = cat(dim, A1, A2, A3, A4 ...)

Описание. Функция **C = cat(dim, A, B)** объединяет массивы **A** и **B** вдоль размерности **dim**.

Функция **C = cat(dim, A1, A2, A3, A4 ?)** объединяет множество исходных массивов **Ai** вдоль размерности **dim**. При этом cat**(1, A, B)** равносильно массиву **[A; B],** объединяемому вдоль строк; **cat(2, A, B)** равносильно массиву **[A B],** объединяемому вдоль столбцов.

Функции вида **cat(dim, A{:})** и **cat(dim, A.<имя_поля>)** задают объединение массива ячеек или массива записей, содержащего числовые матрицы, в некоторый многомерный массив.

Пример. Пусть заданы два 2-мерных массива **A и B**:

A =

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

B =

5	**6**
7	**8**

Выполним их объединение вдоль разных размерностей:

C = cat(1,A,B)

1	**2**
3	**4**
5	**6**
7	**8**

C = cat(2,A,B)

1	**2**	**3**	**4**
5	**6**	**7**	**8**

C = cat(3,A,B)

5	**6**
7	**8**

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

Последовательность операторов

M = magic(3);
P = pascal(3);
C = cat(4, M, P)

создает следующий многомерный массив размера 3x3x1x2:

C(:, :, 1, 1) =

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

C(:, :, 1, 2) =

1	**1**	**1**
1	**2**	**3**
1	**3**	**6**

Объединяя массив **M** и массив **P,** дополненный единичным столбцом, в массив ячеек **S,**

```
S = {M [P ones(size(P, 1), 1)]};
for i=1:length(S),
   siz{i} = size(S{i});
end
sizes = cat(1, siz{:})
```

можно сформировать 2-мерный массив **sizes,** элементами которого являются размеры входных массивов

```
sizes =
    3     3
    3     4
```

Сопутствующие функции: **NUM2CELL**.

Переопределение метода: **funfun\inline\cat.m**

NDIMS - Количество размерностей многомерного массива

Синтаксис:

n = ndims(A)

Описание. Функция **n = ndims(A)** возвращает количество размерностей многомерного массива **A**, которое всегда больше или равно 2. Оконечные единичные размерности **dim** массива **A**, то есть такие размерности, для которых выполняется условие **size(A, dim) = 1**, во внимание не принимаются.

Алгоритм:

ndims(X) = length(size(X))

Сопутствующие функции: **SIZE**.

NDGRID - Генерация сетки для многомерных функций и интерполяции

Синтаксис:

[X1, X2, X3, ...] = ndgrid(x1, x2, x3,.....)
[X1, X2, ?] = ndgrid(x)

Описание. Функция **[X1, X2, X3,.....] = ndgrid(x1, x2, x3,.....)** преобразует области, заданные векторами **x1, x2, x3,....** в массивы **X1, X2, X3, ...,** которые можно использовать в качестве сетки для вычис-

ления функций нескольких переменных и многомерной интерполяции. При этом **i**-ая размерность выходного массива **Xi** повторяет элементы вектора **xi.**

Функция **[X1, X2, ...] = ndgrid(x)** равносильна функции **[X1, X2, ...] = ndgrid(x, x, ...).**

Пример. Вычислить функцию от трех переменных **x2*exp(-x1^2-x2^2-x^3)** на области **-2 < x1 < 2, -2 < x2 < 2, -2 < x3 < 2** и построить ее сечения, используя команду **slice (**Рисунок 5.2):

```
[x1, x2, x3] = ndgrid(-2:.2:2, -2:.25:2, -2:.16:2);
z = x2 .* exp(-x1.^2 - x2.^2 - x3.^2);
slice(x2, x1, x3, z, [-1.2 .8 2], 2, [-2 -.2])
```

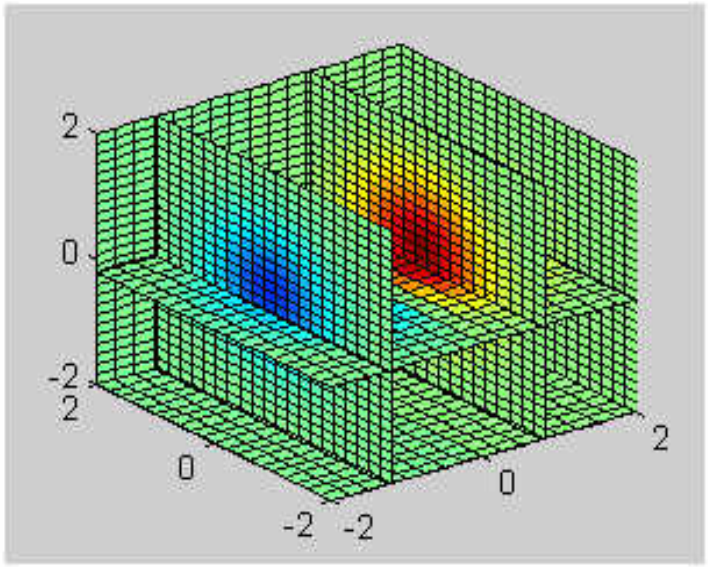


Рисунок 5.2

Замечание. Функция **ndgrid** аналогична функции **meshgrid**, за исключением того, что первые два аргумента переставлены местами, то есть функция **[X1, X2, X3] = ndgrid(x1, x2, x3)** дает тот же результат, что и функция **[X2, X1, X3] = meshgrid(x2, x1, x3).** В силу этого обстоятельства функция **ndgrid** лучше подходит для решения много-

мерных задач, в то время как функция **meshgrid** - для решения пространственных задач в 2-мерном и 3-мерном пространствах.

Сопутствующие функции: **MESHGRID, INTERPN**.

PERMUTE - Прямая и обратная перестановки
IPERMUTE размерностей многомерного массива

Синтаксис:

B = permute(A, <вектор перестановок>)
A = ipermute(B, <вектор перестановок>)

Описание. Функция **B = permute(A, <вектор перестановок>)** осуществляет перестановку размерностей многомерного массива **A** в соответствии с порядком, определенным вектором перестановок. Значения элементов массива остаются неизменными, но порядок размещения последних определяется вектором перестановок; в свою очередь, элементы вектора перестановок - это числа от 1 до **N**, переставленные соответствующим образом.

Функция **A = ipermute(B, <вектор перестановок>)** осуществляет обратную перестановку размерностей многомерного массива **A** в соответствии с порядком, определенным вектором перестановок.

Замечание. Функции **permute** и **ipermute** обобщают операцию траспонирования **(.')** на случай многомерных массивов.

Пример

```
A = rand(1,2,3,4);
B = permute(A, [3 2 1 4]);
size(B)
ans = 3   2   1   4
```

Массив с переставленными размерностями имеет размер 3x2x1x4.

```
C = ipermute(B, [3 2 1 4]);
isequal(A, C)
ans = 1
```

Таким образом, массивы **C** и **A** идентичны.

Сопутствующие функции: нет.

SHIFTDIM - Сдвиг размерностей многомерного массива

Синтаксис:

B = shiftdim(X, n)
[B, n] = shiftdim(X)

Описание. Функция **B = shiftdim(X, n)** сдвигает **n** размерностей многомерного массива **X**; если **n** положительное число, выполняется сдвиг на **n** размерностей влево, а **n** первых размерностей подставляются в конец (круговая перестановка); если **n** отрицательное

число, выполняется сдвиг на **n** размерностей вправо, а **n** первых размерностей дополняются единичными.

Функция **[B, n] = shiftdim(X)** возвращает с тем же количеством элементов, что и **X,** но с удаленными ведущими единичными размерностями; количество удаленных размерностей фиксируется переменной **n.**

Функция **shiftdim** удобна тем, что подобно функциям **sum** и **diff**, работает с первой неединичной размерностью.

Замечание. Если **X** - скаляр, то функции **shiftdim(X)** не выполняется.

Пример.

```
A = rand(1, 1, 3, 1, 2);
[B, n] = shiftdim(A)
B(:, :, 1) =
 0.3046
 0.1897
 0.1934
B(:, :, 2) =
 0.6822
 0.3028
 0.5417
 n =     2
 size(B)
 ans = 3    1   2
```

Массив **B** имеет размер 3x1x2 и **n = 2**.

```
C = shiftdim(B, -n);
isequal(A, C)
ans =     1
```

Таким образом, массивы **C** и **A** идентичны.

```
D = shiftdim(A, 3);
size(D)
ans = 1    2   1   1   3
```

Сопутствующие функции: **RESHAPE, SQUEEZE**.

SQUEEZE - Удаление всех единичных размерностей многомерного массива

Синтаксис:

```
B = squeeze(A)
```

Описание. Функция **B = squeeze(A)** возвращает массив **B** с теми же элементами, что и **A**, но в котором удалены размерности, равные 1.

Пример. Рассмотрим 3-мерный массив **A=rand(2, 1, 3)** размера **2?1?3**. Этот массив имеет размерность столбца, равную 1, то есть на каждой странице размещен один вектор-столбец:

```
A = rand(2, 1, 3)
A(:, :, 1) =
    0.9218
    0.7382

A(:, :, 2) =
    0.1763
    0.4057
A(:, :, 3) =
    0.9355
    0.9169
```

Применение функции **squeeze** превращает его в 2-мерный размера 2x3:

```
squeeze(A)
ans =
0.9218   0.1763   0.9355
0.7382   0.4057   0.9169
```

Сопутствующие функции: **RESHAPE, SHIFTDIM**.

6 Массивы записей

Массив записей - это новый тип массива, в котором разрешается накапливать в виде записей разнородные данные. Отличительная особенность такого массива наличие именованных полей.

MATLAB поддерживает следующие функции при работе с массивами записей:

Функция	*Описание*
struct	Создать массив записей.
fieldnames	Получить имена полей.
getfield	Получить содержимое поля.
setfield	Установить содержимое поля.
rmfield	Удалить поле.
isfield	Истинно, если это поле массива записей.
isstruct	Истинно, если это массив записей.

Пользователь может расширить состав функций, создавая специальные М-файлы для обработки конкретных данных.

Определение структуры. Структура - это массив записей с именованными полями, предназначенными для хранения данных; причем поле может содержать данные любого типа.
Пример. Рассмотрим структуру **patient**, в которой поле **name** предназначено для записи имени пациента, поле **billing** - для счета на оплату, поле **test** - результатов медицинского обследования (рисунок 6.1).

6.1 Построение структур

Структуру можно построить двумя способами:

- с использованием операторов присваивания;
- с использованием функции **struct.**

Применение оператора присваивания. Для того чтобы сформировать простейшую структуру размера 1x1, необходимо присвоить данные соответствующим полям. Система MATLAB автоматически формирует структуру по мере ее заполнения.

Пример. Сформируем показанную на рисунке 6.1 структуру **patient** размера 1x1, состоящую из следующих полей:

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

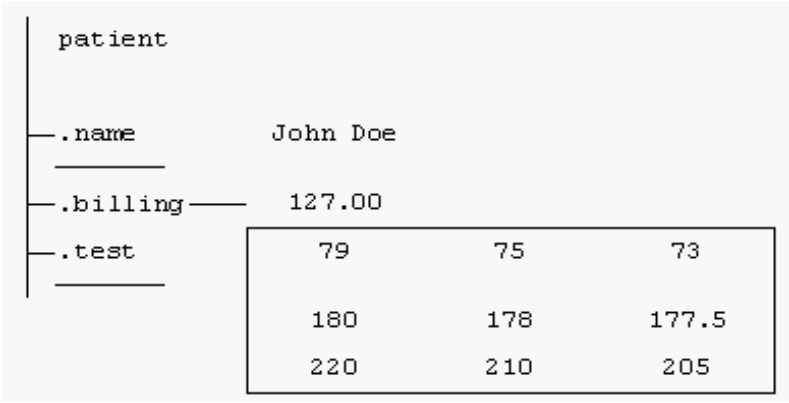


Рисунок 6.1. Массив записей patient размера 1x1

Теперь введя в командной строке имя структуры

patient,

получим описание записи

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

Таким образом, **patient** - это пока массив из одной записи с тремя полями. Для того чтобы расширить его, достаточно добавить индекс в имени структуры.

Пример. Создадим вторую запись в структуре **patient**:

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

Теперь структура **patient** имеет размер 1x2. Заметим, что когда структура содержит более одной записи, при ее запросе, содержимое полей не выводится, а выводится только обобщенная информация о структуре в следующем виде:

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

Для получения этой же информации можно использовать функцию **fieldnames**, которая возвращает массив ячеек, содержащий строки с именами полей.

При расширении структуры система MATLAB заполняет неприсвоенные поля пустыми массивами и, как следствие, по отношению к структуре выполняются следующие условия:

- все записи структуры имеют одинаковое количество полей;
- все имена полей одинаковы.

Пример.

При вводе строки

```
patient(3).name = 'Alan Johnson'
 patient =
      1x3 struct array with fields:
          name
          billing
          test
```

массив **patient** увеличивает размер до 1?3. Поля **patient(3).billing** и **patient(3).test** содержат пустые матрицы.

Размеры полей могут быть разными для разных записей. Для структуры **patient** поле **name** может иметь различные длины, поля **test** могут содержать массивы разных размеров и так далее.

Применение функции struct. Функция **struct** имеет следующий синтаксис:

str_array = struct('<имя_поля1>','<значение>',
'<имя_поля2>','<значение>', ...).

Пример. Воспользуемся функцией **struct**, чтобы создать структуру **patient** размера 1x1:

```
patient = struct('name',' John Doe', 'billing', 127.00,....
              'test', [79 75 73; 180 178 177.5; 220 210 205])
 patient =
    name: ' John Doe'
    billing: 127
       test: [3x3 double]
```

Функция **struct** позволяет сформировать структуру с указанными значениями полей. То есть все поля **name** будут содержать строку **'John Doe'**, все поля **billing** - значение 127.00 и т. д. В дальнейшем можно изменить значения полей, используя операторы присваивания.

6.2 Доступ к полям и данным структуры

Используя индексацию, можно легко определить значение любого поля или элемента структуры. Точно также можно присвоить значение любому полю или элементу поля.
В качестве примера рассмотрим структуру **patient** вида:

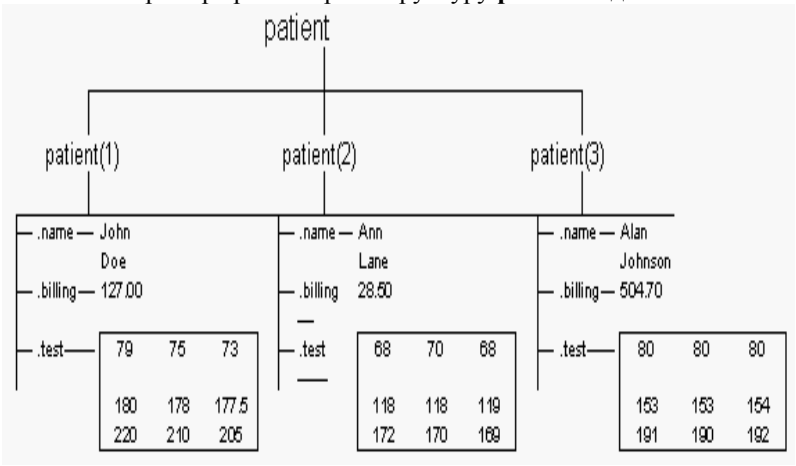


Рисунок 6.2

Чтобы обратиться к некоторому полю, необходимо ввести точку (.) после имени структуры, за которым должно следовать имя поля:

str = patient(2).name
str = Ann Lane

Чтобы обратиться к элементам поля, надо использовать индексацию поля в правой части оператора присваивания. Другими словами, если поле является числовым массивом, использовать индексы массива; если поле - массив ячеек, использовать индексы массива ячеек и т. п.

Пример. Определить для пациента 2 показатель (2, 2) медицинского теста

n = patient(2).test(2, 2)
n = 118

Используя тот же подход, можно присваивать значения элементам поля в левой части оператора присваивания.

Пример. Записать для пациента 3 показатель (2, 2) медицинского теста

```
patient(3).test(2, 2) = 167;
```

Получить значение некоторого поля для всех записей структуры нельзя; это можно сделать только для отдельной записи.

Пример. Для вывода всех значений поля **name** необходимо организовать цикл:

```
for i = 1 : length(patient)
    disp(patient(i).name)
end
John Doe
Ann Lane
Alan Johnson
```

Чтобы получить доступ к записи, необходимо индексировать имя структуры.

Пример. Результатом выполнения нижеследующего оператора является структура размера 1x1, которая соответствует второй записи структуры **patient**:

```
B = patient(2).
B =
     name: 'Ann Lane'
  billing: 28.5000
     test: [3x3 double]
```

Функции setfield и getfield. Непосредственная индексация - это, как правило, наиболее эффективный способ определить или присвоить значение полю записи. Однако, если использовалась функция **fieldnames** и известно имя поля, то можно воспользоваться функциями **setfield** и **getfield**.

Функция **getfield** позволяет определить значение поля или элемента поля:

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

где аргументы **array_index** и **field_index** задают индексы для структуры и поля; они не являются обязательными для структуры размера 1x1. Результат применения функции **getfield** соответствует элементу следующей структуры

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

Пример. Чтобы получить доступ к полю **name** второй записи структуры **patient**, необходимо использовать функцию **getfield** в следующей форме

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

По аналогии функция **setfield** позволяет присваивать значения полям, используя обращение следующего вида

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

Применение функции size. Функция **size** позволяет получить размер массива записей (структуры) или любого ее поля. Задавая в качестве аргумента имя структуры, функция **size** возвращает ее размеры. При задании аргумента в форме **array(n).field** функция **size** возвращает размеры поля.

Пример. Функция

size(patient)

для структуры **patient** размера 1x3 возвращает вектор

ans = 1 3

Обращение

ize(patient(2).name)

возвращает размер поля name для записи **patient(2)**

ans = 1 8

Добавление полей. Для того чтобы добавить новое поле к структуре, достаточно добавить поле к единственной записи.

Пример. Чтобы добавить поле для индивидуального номера страхования **pin (personal insurance number)** в структуре **patient**, надо выполнить следующий оператор присваивания

```
patient(2).pin = '125-33-5555';
patient(1:3).pin
ans =     [ ]
ans = 125-33-5555
ans =     [ ]
```

Теперь поле **patient(2).pin** имеет присвоенное значение и каждая запись включает поле **pin**. Этим полям соответствуют пустые массивы до тех пор, пока им не будут присвоены конкретные значения.

Удаление полей. Для удаления поля из структуры предназначена функция **rmfield**, которая имеет следующий синтаксис

struc2 = rmfield(array, 'field'),

где **array** - имя структуры, а **'field'** - имя поля, которое подлежит удалению.

Для удаления поля **name** в структуре **patient** надо использовать оператор

patient = rmfield(patient, 'name');

6.3 Обработка структур

Выполнение операций с полями и элементами полей абсолютно аналогично операциям с элементами обычного числового массива. В обоих случаях надо использовать индексные выражения.

Пример. Вычислить среднее арифметическое строк массива **test** для записи **patient(2):**

patient(2).test

Поле **patient(2).test** содержит следующий массив

```
ans =
    68    70    68
   118   118   119
   172   170   169
```

Средние значения его строк могут быть вычислены следующим образом

```
mean((patient(1,2).test)')
 ans = 68.6667   118.3333   170.3333
```

Существует несколько способов применения функций и операторов системы MATLAB для работы с полями структуры:

- использование циклов;
- заключение обозначения поля в квадратные скобки [<имя_структуры>.<имя_поля>].

Пример. Рассмотрим операцию суммирования полей **billing** в структуре **patient**:

использование цикла:

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

Проверим длину записи **patient** и содержимое поля **patient.billing**

```
[patient.billing]
ans = 127.0000    28.5000
length(patient)
ans =    3
```

Таким образом, одно из значений поля **patient.billing** оказывается неприсвоенным и поэтому результат **total** оказывается пустым.

Использование квадратных скобок для полей:

```
total = sum ([patient.billing])
 total = 155.5000
```

Если одному или нескольким полям значения не были присвоены, то результат приведенного выше цикла будет пустым, в то время как применение функции **sum** даст сумму значащих полей.

Суммирование значений некоторого поля эквивалентно оператору

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

Написание функций для работы со структурами. Для обработки структур со специфической архитектурой полей могут понадобиться специальные функции обработки полей и их элементов. При написании М-файлов для обработки структур необходимо помнить, что пользователь должен сам выполнить анализ возникновения возможных ошибок, связанных с обработкой полей.

Пример. Рассмотрим набор данных, связанных с замером в разные моменты времени токсинов в воде. Данные состоят из 15 отдельных наблюдений, где каждое наблюдение содержит три измерения. Можно объединить эти данные в массив из 15 записей, каждая из которых имеет 3 поля, по одному на каждое измерение. Приведенная ниже функция **concen** оперирует со специфическими харктеристиками структуры, содержащей поля **lead**, **mercury** и **chromium**, которым соответствуют концентрации свинца, ртути и хрома.

```
function [r1,r2] = concen(toxtest);
 k = length(toxtest);
 % Вычислить 2 вектора:
 % r1 - отношение концентраций ртути к свинцу
 % r2 - отношение концентраций свинцу к хрому.
 for i = 1:k
     r1 = [toxtest.mercury]./[toxtest.lead];
     r2 = [toxtest.lead]./[toxtest.chromium];
 end
 % Графики концентраций свинца, ртути и хрома
 for j = 1:k
     lead = [toxtest.lead];
     mercury = [toxtest.mercury];
     chromium = [toxtest.chromium];
 end
```

```
plot(lead, 'r'); hold on
plot(mercury, 'b')
plot(chromium, 'g'); hold off
```

Проверим эту функцию на примере структуры **test:**

```
test(1).lead = .007;    test(2).lead = .031;    test(3).lead = .019;

test(1).mercury =       test(2).mercury =       test(3).mercury =
.0021;                  .0009;                  .0013;

test(1).chromium =      test(2).chromium =      test(3).chromium =
.025;                   .017;                   .10;
```

```
[r1, r2] = concen(test)
    r1 = 0.3000   0.0290   0.0684
    r2 = 0.2800   1.8235   0.1900
```

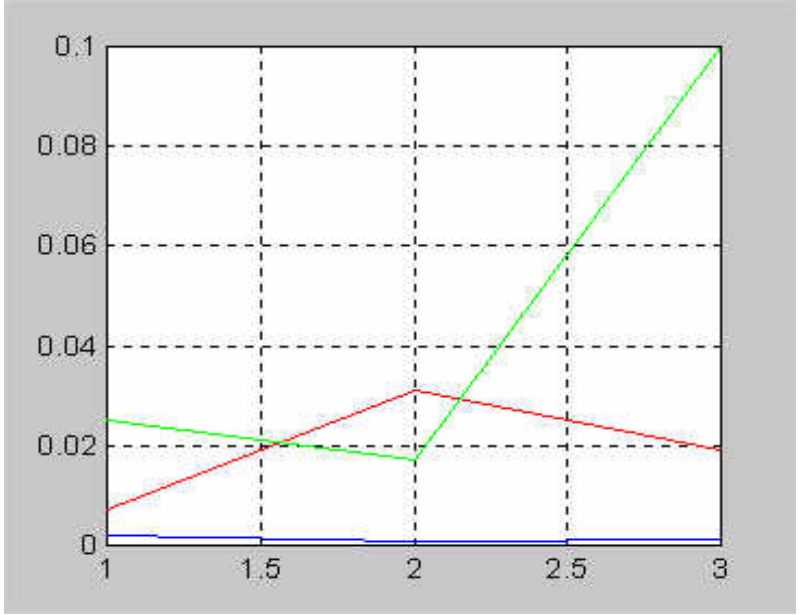


Рисунок 6.3

6.4 Организация данных. Вложенные структуры. Многомерные массивы структур

Организация данных

Ключ к организации структуры состоит в указании способа доступа к элементам сруктуры. Это, в свою очередь, определяет, как надо формировать массив записей и его поля.

Рассмотрим **RGB**-изображение размером 128x128 пикселей, записанное в трех массивах: **RED, GREEN, BLUE:**

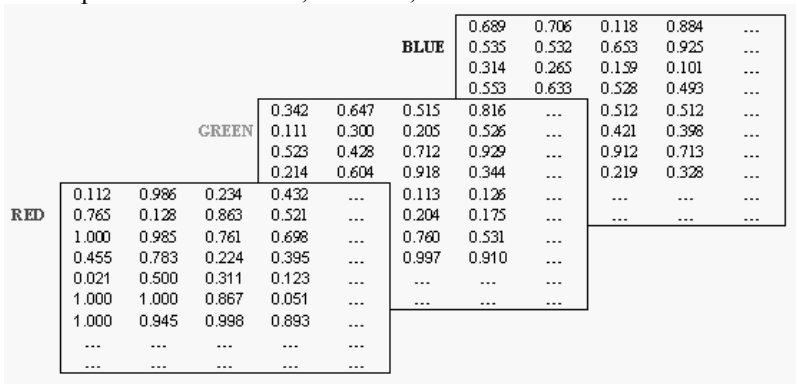


Рисунок 6.4

Имеются, по крайней мере, два способа организации этих данных в структуру (рисунки 6.5 и 6.6).

Матричная организация. В этом случае каждое поле структуры - это массив координат точек одного цвета. Такую структуру можно создать, используя следующие присваивания

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

Этот подход позволяет выделить целые массивы для отображения, фильтрации или выполнения других функций над изображением. В этом случае, чтобы получить доступ к множеству точек

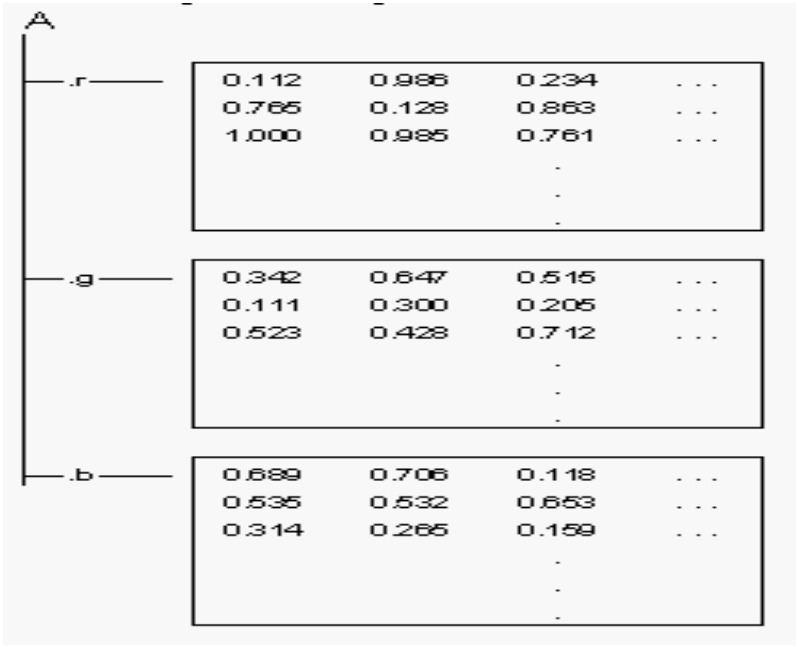

A
.r
0.112 0.986 0.234 . . .
0.765 0.128 0.863 . . .
1.000 0.985 0.761 . . .
.g
0.342 0.647 0.515 . . .
0.111 0.300 0.205 . . .
0.523 0.428 0.712 . . .
.b
0.689 0.706 0.118 . . .
0.535 0.532 0.653 . . .
0.314 0.265 0.159 . . .

Рис. 6.5

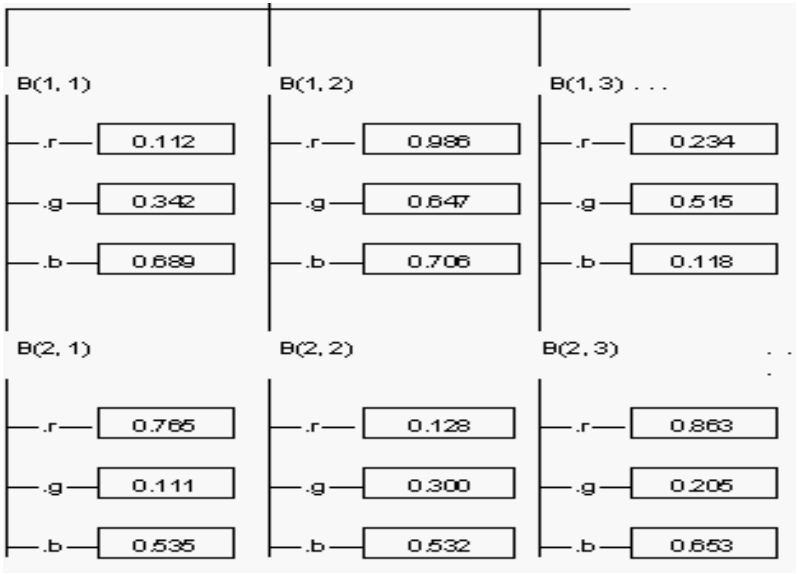

B(1, 1)
.r 0.112
.g 0.342
.b 0.689
B(1, 2)
.r 0.986
.g 0.647
.b 0.706
B(1, 3) . . .
.r 0.234
.g 0.515
.b 0.118
B(2, 1)
.r 0.765
.g 0.111
.b 0.535
B(2, 2)
.r 0.128
.g 0.300
.b 0.532
B(2, 3)
.r 0.863
.g 0.205
.b 0.653

Рис. 6.6

красного цвета, надо всего лишь воспользоваться присваиванием

```
red_plane = A.red;
```

Матричная организация имеет преимущества при работе со множеством изображений, когда эти изображения можно накопить в виде массивов **A(2), A(3), ….. ,** содержащих целые образы.

Недостаток такой организации очевиден, когда требуется получить доступ к подмассиву массива цветов. Для того чтобы получить доступ к подобразу, необходимо получить доступ к определенным участкам поля структуры, например, таким

```
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. В этом случае определение отдельных элементов данных реализуется в виде циклов

```
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 такой организации доступ к подмножеству данных может быть выполнен с помощью оператора присваивания

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

Однако, чтобы получить доступ к матрице, требуется организовать цикл:

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

Поэлементная организация - не лучший выбор для приложений, связанных с обработкой изображений; однако он может оказаться предпочтительным, когда необходим доступ к отдельным участкам полей. Следующий пример демонстрирует преимущества такого подхода (рисунки 6.7 и 6.8).

Пример. Рассмотрим следующие два варианта организации базы данных. Каждая из них имеет свои преимущества и недостатки:

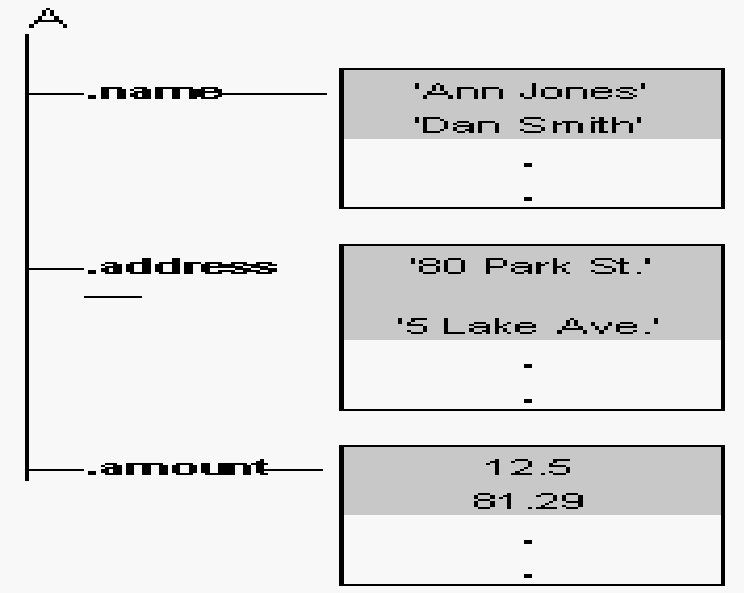

A
.name
'Ann Jones'
'Dan Smith'
.address
'80 Park St.'
'5 Lake Ave.'
.amount
12.5
81.29

Рис. 6.7

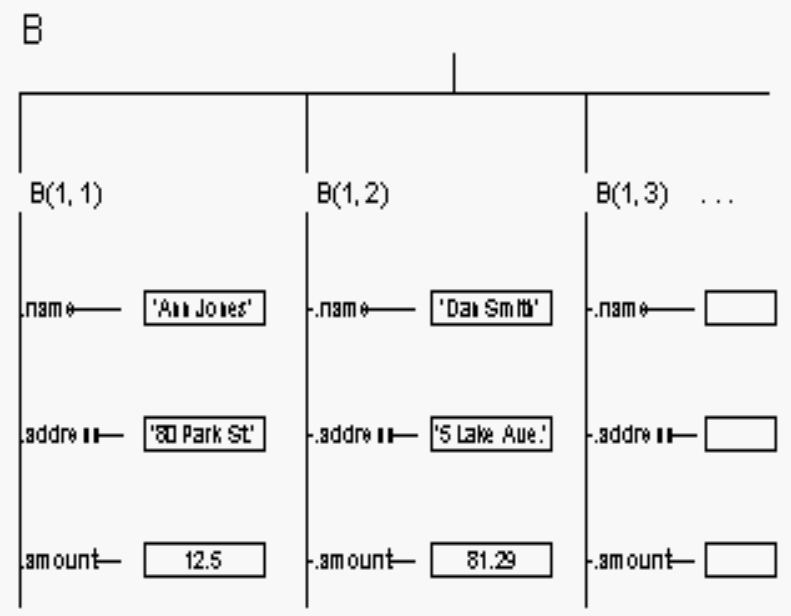

B
B(1, 1)
B(1, 2)
B(1, 3)
.name
'Ann Jones'
'Dan Smith'
.address
'80 Park St'
'5 Lake Ave.'
.amount
12.5
81.29

Рис. 6.8

- Матричная организация позволяет упростить обработку полей данных.

Пример. Нахождение среднего всех значений поля **amount** реализуется следующим образом:

в случае матричной организации:

avg = mean(A.amount);

в случае поэлементной организации:

avg = mean([B.amount]);

- Поэлементная организация позволяет упростить доступ к полям, связанным с отдельной записью.

Пример. Рассмотрим М-файл **client.m,** который выводит на экран имя и адрес клиента.

- в случае матричной организации надо отображать индивидуальные поля:

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

- в случае поэлементной организации можно отобразить целую запись:

```
function client(B)
disp(B)
```

Вызов функции **client** должен быть организован следующим образом:

- в случае матричной организации:

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

- в случае поэлементной организации:

```
client(B(2))
```

Поэлементная организация упрощает работу со строками переменной длины. В случае матричной организации, если априори неизвестна максимальная длина строки, придется многократно определять размеры полей **name** или **address**, чтобы разместить более длинные строки.

Обычно сами данные не диктуют выбор схемы организации; ее определяют операции доступа и обработки данных.

Вложенные структуры

Поле структуры может само включать другую структуру или даже массив структур. Как только структура создана, с помощью операторов присваивания или функции **struct** можно вложить структуры в существующие поля.

Применение функции struct. Чтобы сформировать вложенную структуру, необходимо организовать рекурсию при вызове функции struct.

Пример. Допустим, что требуется создать структуру размера 1x2. Организуем следующий рекурсивный вызов функции struct:

```
A = struct('data',[3 4 7; 8 0 1],'nest',...
   struct('testnum','Test 1','xdata',[4 2 8],'ydata',[7 1 6]))
A =
  data: [2x3 double]
  nest: [1x1 struct]
```

Запись **A(1)** содержит требуемые значения, благодаря вызову внешней функции **struct.**

Следующая последовательность операторов производит результат, аналогичный предыдущему:

```
A(1).data = [3 4 7; 8 0 1];
A(1).nest.testnum = 'Test 1';
A(1).nest.xdata = [4 2 8];
A(1).nest.ydata = [7 1 6];
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]
A =
  1x2 struct array with fields:
    data
    nest
```

Введем изменения в запись **A(2**):

```
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]
A =
  1x2 struct array with fields:
    data
    nest
```

Тогда получим структуру, приведённую на рисунке 6.9. Как и в случае массивов записей с одним уровнем вложения, можно с использованием операторов присваивания сформировать вложенные массивы структур.

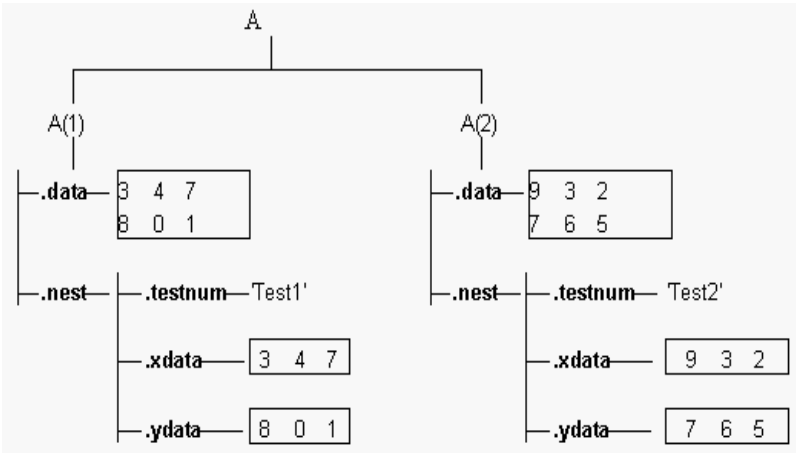


Рисунок 6.9

Индексация вложенных структур. Для того чтобы проиндексировать структуру, надо добавить имена вложенных полей, используя в качестве разделителя точку (.). Первая текстовая строка индексного выражения определяет имя структуры, а последующие имена полей, содержащих другие структуры.

Пример. Вышеописанный массив **A** имеет 2 уровня вложенности:

- для получения доступа к вложенной структуре внутри **A(1)** надо использовать **A(1).nest**.
- для получения доступа к полю **xdata** вложенной структуры внутри **A(1)** надо использовать **A(2).nest.xdata**.
- для получения доступа к элементу 2 поля **ydata** вложенной структуры внутри **A(1)** надо использовать **A(1).nest.ydata(2).**

Многомерные массивы структур

Многомерные массивы структур рассматриваются как расширение прямоугольных массивов структур. По аналогии с другими типами многомерных массивов их можно формировать, либо используя операторы присваивания, либо функцию **cat**.

Пример. Сформируем многомерный массив структур следующего вида, используя операторы присваивания (рисунок 6.10):

```
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 73 75; 103 103 102; 201 198 200];

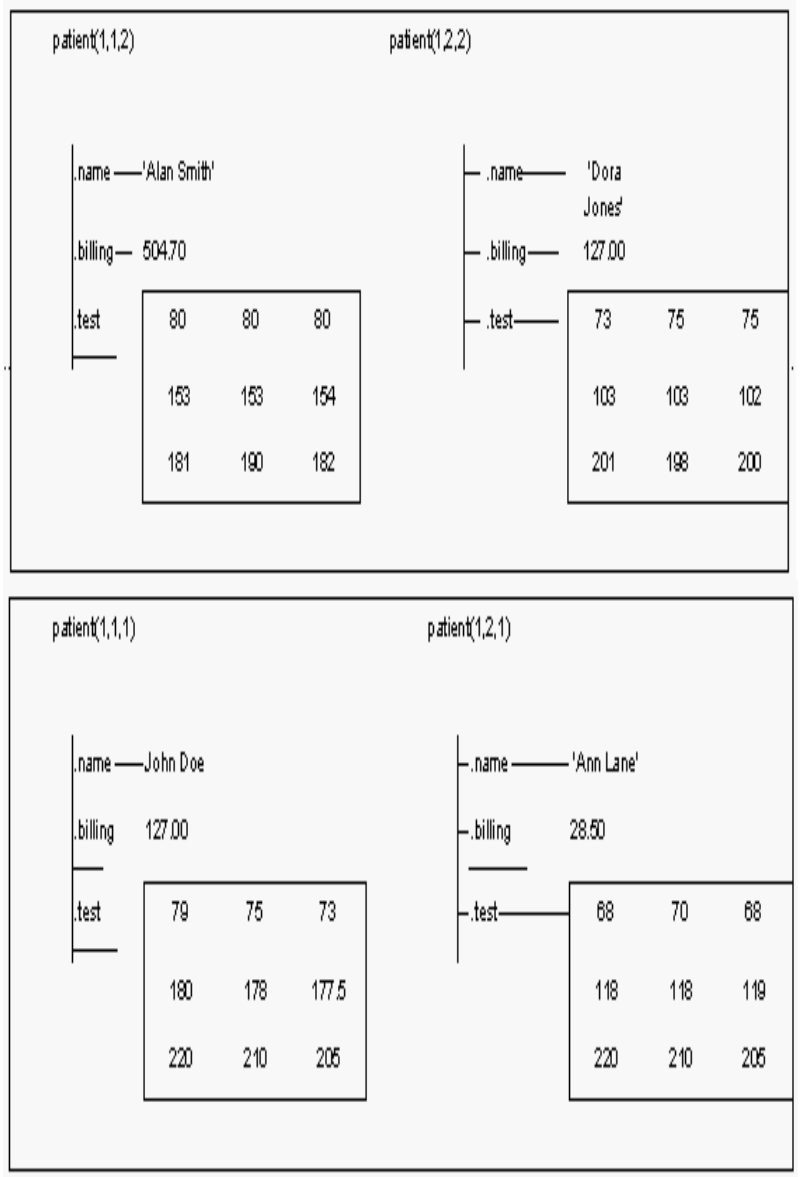


Рисунок 6.10. Многомерный массив структур размера 1x2x2

Для применения функций к многомерным массивам структур надо использовать индексный подход, чтобы получить доступ к полям записи и элементам полей.

Пример. Вычислить сумму значений столбцов массива **test** для структуры **patient(1, 1, 2):**

```
sum((patient(1, 1, 2).test))
ans =      414    423    416
```

Суммировать все поля **billing** 3-мерного массива структур **patient:**

```
total = sum([patient.billing])
total =      1834.10
```

6.5 Функции для работы с массивами записей

STRUCT - Создать массив записей (структуру)

Синтаксис:

S = struct('<имя_поля1>',< значение>, '<имя_поля2>',< значение>, ...)

Описание. Функция **S = struct('<имя_поля1>', <значение>, '<имя_поля2>',<значение>, ...)** создает массив записей (структуру) с заданными именами и значениями полей.

Пример. Воспользуемся функцией **struct**, чтобы создать структуру **patient** размера 1x1:

```
patient = struct('name',' John Doe', 'billing', 127.00,... 'test', [79 75 73; 180 178 177.5; 220 210 205])
patient =
    name: ' John Doe' billing: 127
    test: [3x3 double
```

Сопутствующие функции: **CLASS, CELL, GETFIELD, SETFIELD, RMFIELD, FIELDNAMES**.

FIELDNAMES - Получить имена полей

Синтаксис:

names = fieldnames(S)

Описание:

Функция **names = fieldnames(S)** возвращает имена полей структуры **S** в виде строк массива ячеек.

Пример. Задана следующая структура **A** размера 1x2:

```
A(1).data = [3 4 7; 8 0 1];
A(1).nest.testnum = 'Test 1';
```

```
A(1).nest.xdata = [4 2 8];
A(1).nest.ydata = [7 1 6];
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]
```

Определим имена ее полей, используя функцию **fieldnames**:

```
fieldnames(A)
ans =
    'data'
    'nest'
```

Сопутствующие функции: **GETFIELD, SETFIELD**.

GETFIELD - Получить содержимое поля

Синтаксис:

F = getfield(s, '<имя_поля>')
F = getfield(S, {i, j}, '<имя_поля>', {k})

Описание:

Функция **F = getfield(s, '<имя_поля>'),** где элемент структуры или структура **s** должны иметь размер **s**, возвращает содержимое указанного поля.

Функция **F = getfield(S, {i, j}, '<имя_поля>', {k})** равносильна следующему оператору присваивания **F = S(i, j).<имя_поля>(k).** Все индексы передаются как массивы ячеек и заключаются в фигурные скобки; имена полей передаются как строки.

Пример. Задана следующая структура **A** размера 1x2:

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

Определим содержимое поля **A(1).nest**:

```
getfield(A(1), 'nest')
ans =
    testnum: 'Test 1'
    xdata: [4 2 8]
    ydata: [7 1 6]
```

Это также равносильно следующему оператору

```
getfield(A, {1}, 'nest')
ans =
```

testnum: 'Test 1'
xdata: [4 2 8]
ydata: [7 1 6]

Сравните эти результаты с обращением к оператору **A.nest**:

A.nest
ans =
testnum: 'Test 1'
xdata: [4 2 8]
ydata: [7 1 6]
ans =
testnum: 'Test 2'
xdata: [3 4 2]
ydata: [5 0 9]

Сопутствующие функции: **SETFIELD, FIELDNAMES**.

SETFIELD - Установить содержимое поля

Синтаксис:

s = setfield(s, '<имя_поля>', V)
s = setfield(S, {i, j}, '<имя_поля>', {k}, V)

Описание:

Функция **s = setfield(s, '<имя_поля>', V),** где элемент структуры или структура **s** должны иметь размер 1x1, присваивает указанному полю значение **V**.

Функция **s = setfield(S, {i, j}, '<имя_поля>', {k}, V)** равносильна следующему оператору присваивания **S(i, j).<имя_поля>(k) = V**. Все индексы передаются как массивы ячеек и заключаются в фигурные скобки; имена полей передаются как строки.

Пример. Задана следующая структура **A** размера 1x2:

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

Присвоить новое значение полю **A(1).nest.xdata**:

A = setfield(A(1), 'nest.xdata', [5 3 9]);
getfield(A(1), 'nest')
ans =
testnum: 'Test 1'
xdata: [5 3 9]
ydata: [7 1 6]

Это также равносильно следующему оператору

```
A = setfield(A, {1}, 'nest.xdata', [5 3 9]);
getfield(A, {1}, 'nest.xdata')
ans =
    testnum: 'Test 1'
    xdata: [5 3 9]
    ydata: [7 1 6]
```

Сопутствующие функции: **GETFIELD, FIELDNAMES**.

RMFIELD - Удалить поле

Синтаксис:

S = rmfield(S, '<имя_поля>')

S = rmfield(S, F)

Описание:

Функция **S = rmfield(S, '<имя_поля>')** удаляет указанное поле из структуры.

Функция **S = rmfield(S, F)**, где **F** - символьный массив имен полей или массив ячеек соответствующих строк, удаляет все указанные поля из структуры. Замечание: Удалить таким способом все поля из структуры нельзя.

Пример. Задана следующая структура **A** размера 1x2:

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

Удалить **A(1).data**:

```
B=rmfield(A,'data')
B =
   1x2 struct array with fields:
       nest
B.nest
ans =
   testnum: 'Test 1'
     xdata: [4 2 8]
     ydata: [7 1 6]
ans =
   testnum: 'Test 2'
     xdata: [3 4 2]
     ydata: [5 0 9]
```

Попытка удалить поле **nest** приводит к сообщению об ошибке

B = rmfield(B, 'nest')

??? To RESHAPE the number of elements must not change.

Для выполнения функции **RESHAPE** должно быть изменено количество элементов

Error in ==> d:\matlab5\toolbox\matlab\datatypes\rmfield.m

On line 43 ==> t = reshape(t,size(s));

Ошибка в ==> d:\matlab5\toolbox\matlab\datatypes\rmfield.m

В строке 43 ==> t = reshape(t,size(s));

Сопутствующие функции: **SETFIELD, GETFIELD, FIELDNAMES, STRVCAT**.

ISFIELD - Логическая проверка поля

Синтаксис:

k = isfield(S, '<имя_поля>')

Описание:

Функция **k = isfield(S, '<имя_поля>')** возвращает 1 (логическое **TRUE**), если указанное имя действительно является именем поля данной структуры.

Пример. Задана следующая структура **A** размера 1x2:

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

Проверить, является ли поля **'data', 'nest', 'nest.xdata'** полями структуры **A**:

```
isfield(A,'data')
ans =    1
isfield(A,'nest')
ans =    1
isfield(A,'nest.xdata')
ans =    0
```

Сопутствующие функции: **SETFIELD, GETFIELD, FIELDNAMES**.

ISSTRUCT - Логическая проверка структуры

Синтаксис:

k = isstruct(S)

Описание:

Функция **k = isstruct(S)** возвращает 1 (логическое **TRUE**), если указанное имя действительно является именем структуры и 0 - в противном случае.

Пример. Задана следующая структура **A** размера 1x2:

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

Проверить, является ли объект **A** структурой:

```
isstruct(A)
 ans = 1
```

Сопутствующие функции: **STRUCT, ISCELL, ISNUMERIC, ISOBJECT**.

7 Массивы ячеек

В систему MATLAB 5 впервые включен специальный тип массивов ячеек, элементы которого сами, в свою очередь, являются массивами . Поддержаны следующие функции при работе с массивами ячеек:

Функция	***Описание***
cell	Создать массив ячеек.
celldisp	Показать содержимое массива ячеек.
cellplot	Показать графическую структуру массива ячеек.
num2cell	Преобразовать числовой массив в массив ячеек.
deal	Обмен данными между любыми классами масси-

	вов.
cell2struct	Преобразовать массив ячеек в структуру.
struct2cell	Преобразовать структуру в массив ячеек.
iscell	Истинно, если это массив ячеек.

Пользователь может расширить состав этих функций, создавая специальные М-файлы для обработки конкретных данных.

Определение массива ячеек. Массив ячеек - это массив, в котором элементами являются ячейки, которые могут содержать любой тип массива, в том числе и массив ячеек. Массивы ячеек позволяют хранить массивы с элементами разных типов и разных размерностей. К примеру, одна из ячеек может содержать действительную матрицу, другая массив текстовых строк, третья - вектор комплексных чисел (рисунок 7.1).

Можно строить массивы ячеек любых размеров и любой структуры, включая и многомерные.

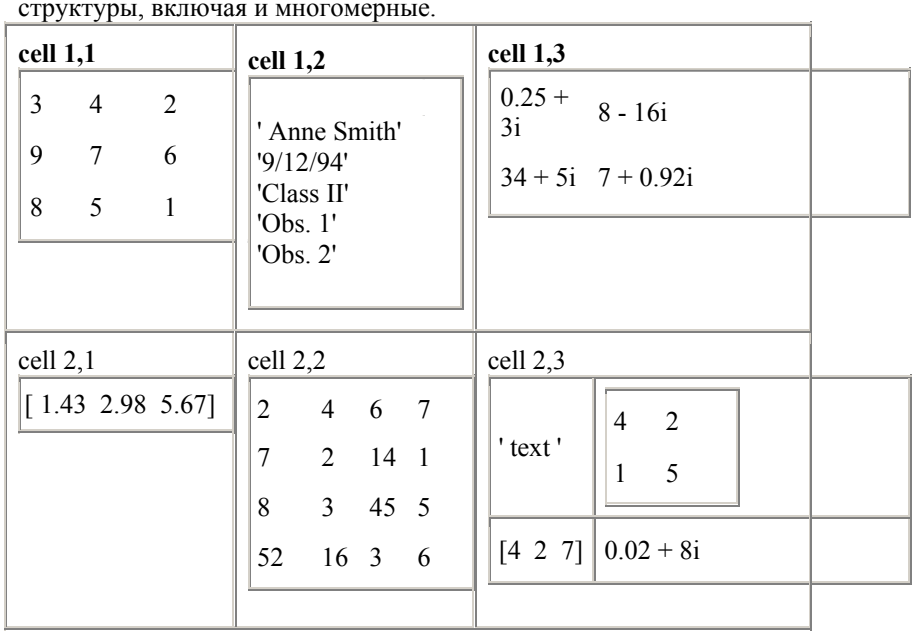


Рисунок 7.1

7.1 Создание массивов ячеек. Применение операторов присваивания

Создать массивы ячеек можно двумя способами:

- используя операторы присваивания;
- используя функцию **cell**, которая позволяет предварительно разместить массив, а затем присвоить данные ячейкам

Применение операторов присваивания

Можно построить массив ячеек, присваивая данные отдельным ячейкам; система MATLAB автоматически строит массив по мере ввода данных. Существует два способа присвоить данные отдельным ячейкам.

Индексация ячеек. Заключить индексы ячейки в круглые скобки, используя стандартные обозначения для массива. Заключить содержимое ячейки в правой части оператора присваивания в фигурные скобки { }.

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

Обозначение { } соответствует пустому массиву ячеек точно также, как **[ ]** соответствует пустому числовому массиву.

Индексация содержимого. Для того чтобы индексировать массив ячеек, надо в левой части оператора присваивания указать элемент ячейки в виде индексов в фигурных скобках по аналогии с элементами обычного массива, а также указать содержимое ячейки в правой части оператора присваивания, как это показано на следующем ппримере.

Пример.

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

Здесь не делается попытки отдать предпочтение одной из форм (индексация ячейки или индексация содержимого), а лишь приводится их описание. Пользователь может сам выбрать, какая форма ему более подходит.

Замечание. Если существует числовой массив с некоторым именем, не пытайтесь создавать массив ячеек с тем же именем, не удалив числовой массив. В этом случае система MATLAB генерирует ошибку. Точно также система MATLAB не очищает массив ячеек при выполнении оператора присваивания. Если в каких-либо примерах возникают непредсказуемые результаты, надо прежде всего удалить массив ячеек из рабочей области и повторить операцию.

Система MATLAB отображает массив ячеек в сжатой форме

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

Для отображения содержимого ячеек следует использовать функцию **celldisp**:

```
celldisp(A)
A{1, 1} =
     1     4     3
     0     5     8
     7     2     9

A{2, 1} = 3.0000+ 7.0000i
A{1,2} = Anne Smith
A{2, 2} =
  -3.1416   -2.8274   -2.5133   -2.1991   -1.8850   -1.5708   -1.2566
  -0.9425   -0.6283   -0.3142         0    0.3142    0.6283    0.9425
   1.2566    1.5708    1.8850    2.1991    2.5133    2.8274    3.1416
```

Для отображения структуры массива ячеек в виде графического изображения предназначена функция **cellplot**:

```
cellplot(A)
```

На рисунке 7.2 приведено графическое изображение этого массива ячеек.

Если данные присваиваются ячейке, которая находится вне пределов текущего массива, MATLAB автоматически расширяет мас-

сив ячеек. При этом ячейки, которым не присвоено значений, заполняются пустыми массивами.

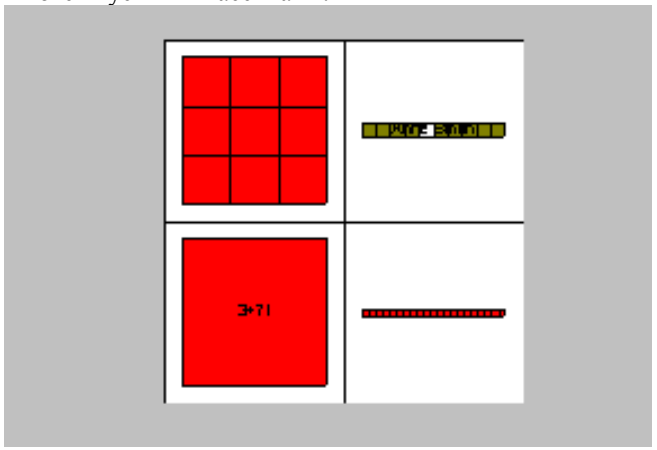


Рисунок 7.2. Массив ячеек размера 2x2

Пример. Добавление ячейки {3, 3} к определенному ранее массиву ячеек **A** размера 2x2 превращает его в массив размера 3x3, оставляя 4 дополнительных ячейки пустыми:

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

На рисунке 7.3, а показано отображение массива ячеек с помощью функции графического вывода **cellplo**t, а на рисунке 7.3, б - способ представления, принятый в данной книге.

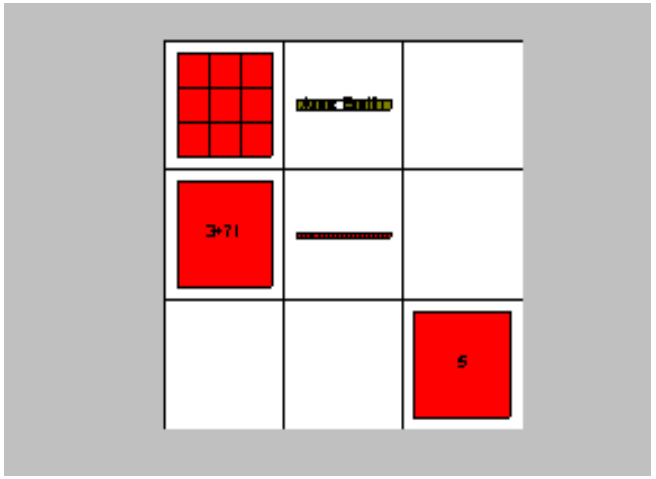
Использование скобок. Фигурные скобки **{ }** являются конструктором массива ячеек, а квадратные **[ ]** - конструктором числового массива. Фигурные скобки аналогичны квадратным скобкам, за исключением того, что они могут быть еще и вложенными.

Пример. Оператор присваивания

C = {[1 2], [3 4]; [5 6], [7 8]}

формирует следующий массив ячеек

cell 1,1 **[1 2]**	**cell 1,2** **[3 4]**
cell 2,1 **[5 6]**	**cell 2,2** **[7 8]**



а)

cell 1,1 **3 4 2** **9 7 6** **8 5 1**	**cell 1,2** **'Anne Smith'**	**cell 1,3** **[ ]**
cell 2,1 **3 + 7i**	**cell 2,2** **[-3.14 ... 3.14]**	**cell 2,3** **[ ]**
cell 3,1 **[ ]**	**cell 3,2** **[ ]**	**cell 3,3** **5**

б)

Рисунок 7.3. Массив ячеек размера 3x3:
а) - графический образ **cellplot**;
б) - изображение, принятое в этой книге

Квадратные скобки используются для объединения нескольких ячеек, как это делается для числовых массивов.

Применение функции cell. Функция **cell** позволяет создать шаблон массива ячеек, заполняя его пустыми ячейками.

Пример. Создать пустой массив ячеек размера 2x3

```
B = cell(2, 3)
 B =
```

[ ] [ ] [ ]

[ ] [ ] [ ]

Используя оператор присваивания, заполним одну из ячеек массива

```
B(1, 3) = {1:3};
 B =
     [ ]    [ ]    [1x3 double]
     [ ]    [ ]    [ ]
```

7.2 Извлечение данных

Существует два способа извлечь данные из массива ячеек для передачи их либо в некоторый числовой массив, либо в новый массив ячеек:

- доступ к содержимому ячейки, используя индексацию содержимого;
- доступ к подмножеству ячеек, используя индексацию ячеек .

Доступ к содержимому ячеек (индексация содержимого). Используя индексирование содержимого в правой части оператора присваивания можно получить доступ к некоторым или всем данным в одной ячейке. Определить переменную в левой части оператора присваивания, чтобы запомнить содержимое ячейки. Заключить индексное выражение в правой части оператора присваивания в фигурные скобки. Это будет означать, что присваивается содержимое ячеек, а не сами ячейки.

Пример.

Рассмотрим массив ячеек **N** размера 2x2:

```
N{1, 1} = [1 2; 4 5];
N{1, 2} = 'Name';
N{2, 1} = 2-4i;
N{2, 2} = 7;
N =
   [2x2 double] 'Name'
   [2.0000- 4.0000i] [ 7]
```

Строку, находящуюся в ячейке **N{1, 2}** можно извлечь следующим образом:

```
c = N{1, 2}
 c = Name
```

Для того чтобы извлечь содержимое из некоторого подмножества ячеек необходимо использовать конкатенацию индексных выражений.

Пример. Извлечь элемент с индексами (2,2) из числового массива ячейки **N{1, 1}**:

```
d = N{1, 1}(2, 2)
 d = 5
```

Доступ к подмножеству ячеек (индексация ячеек). Используя индексацию ячеек, можно переназначить любой набор ячеек другой переменной для создания нового массива ячеек. Используя двоеточие, можно получить доступ к подмножествам ячеек внутри массива ячеек (смотри рисунок 7.4).

cell 1,1 1	**cell 1,2 2**	**cell 1,3 3**
cell 2,1 4	**cell 2,2 5**	**cell 2,3 6**
cell 3,1 7	**cell 3,2 8**	**cell 3,3 9**

-----B = A(2:3, 2:3)--->

cell 1,1 5	**cell 1,2 6**
cell 2,1 7	**cell 2,2 8**

Рисунок 7.4

Удаление и переопределение массива ячеек. Удаляя ячейки из масиива, можно уменьшить размерность массива, применяя единственный оператор присваивания. По аналогии с удалением обычного массива используйте индексацию вектора при удалении строки или столбца ячеек, присваивая пустую матрицу подмассиву:

```
A(j : k ) = [ ]
```

Таким образом, при удалении ячеек фигурные скобки вообще не применяются в операторах присваивания.

Подобно обычным массивам с помощью функции **reshape** можно переопределять размеры массива ячеек, причем общее количество ячеек должно оставаться неизменным; с помощью функции **reshape** ни удалить, ни добавить ячеек нельзя.

Пример.

```
A = cell(3, 4)
 A =
```

[ ] [ ] [ ] [ ]

[ ] [ ] [ ] [ ]

[ ] [ ] [ ] [ ]

size(A)

ans = 3 4

B = reshape(A, 6, 2)

B =

[ ] [ ]

[ ] [ ]

[ ] [ ]

[ ] [ ]

[ ] [ ]

[ ] [ ]

size(B)

ans = 6 2

Описание списков переменных. Массивы ячеек могут быть использованы для замены следующих списков переменных:

- списков входных переменных;
- списков выходных переменных;
- операций вывода на экран терминала;
- квадратных и фигурных скобок при формировании массивов.

Когда для индексирования многомерного массива ячеек используются двоеточие и фигурные скобки, то система MATLAB обрабатывает содержимое каждой ячейки как отдельную переменную.

Пример. Допустим, существует массив ячеек **T**, в котором каждая ячейка содержит вектор; тогда выражение **T{1:5}** эквивалентно списку векторов из первых 5 ячеек массива **T**.

Рассмотрим следующий 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};
```

Используя функцию **conv**, найдем произведение полиномов, определяемых векторами (свертку векторов) из ячеек **C(1)** и **C(2):**

```
d = conv(C{1:2})
d = 1   2   4   2   3
```

Выведим на экран векторы из второй, третьей и четвертой ячеек

```
C{2:4}
ans = 1   0   1
ans = 1   2   3   4   5   6   7   8   9   10
ans = 9   8   7
```

Можно сформировать новый числовой массив, используя следующий оператор присваивания

```
B = [C{1}; C{2}; C{4}]
B =

     1     2     3
     1     0     1
     9     8     7
```

Теперь используя индексацию содержимого в левой части оператора присваивания, можно создать новый массив, каждая ячейка которого представляет отдельный выход:

```
[D{1:2}] = eig(B)
D =
    [3x3 double] [3x3 double]
```

Можно вывести на экран матрицы правых собственных векторов и собственных значений, используя ячейки **D{1}** и **D{2},** соответственно.

```
D{1}
ans =
   0.3088    0.7071    0.5585
   0.1148    0.0000   -0.8091
   0.9442   -0.7071    0.1827

D{2}
ans =
   10.9161         0         0
         0   -2.0000         0
         0         0   -0.9161
```

Замечание. Списки входов и выходов **varargin** и **varargout** позволяют использовать переменное количество входных и выходных аргументов. Эти списки являются массивами ячеек, что позволяет поддерживать разные размеры и типы данных.

7.3 Организация данных

Массивы ячеек используются для объединения массивов данных разных типов и размеров. Массивы ячеек предпочтительнее массивов записей (структур) при следующих обстоятельствах:

- когда требуется доступ одновременно к нескольким полям;
- когда требуется доступ к подмножествам данных в виде списка переменных;
- когда количество полей не определено;
- когда вам требуется удаление полей из структуры.

Списки значений. Извлечение множественных данных из массивов записей и массивов ячеек осуществляется с помощью списков значений.

Список значений для массива записей - это объединение одноименных полей **S.name = [S(1).name S(2).name … S(end).name].**

Список значений для массива ячеек - это объединение ячеек **C{:} = [C{1} C{2} … C{end}].**

Конструкции вида **S(m:n).name, C{m:n**} также представляют собой списки значений.

Формы использования списков значений в различных конструкциях языка обобщены в следующей таблице:

Конструкция языка	***Массив записей***	***Массив ячеек***
Командная строка	**S.name**	**C{:}**
Список входных аргументов М-функции	**myfun(x, y, S.name)**	**myfun(x, y, C{:})**
Операция конкатенации	**[S.name]**	**[C{:}]**
Список выходных аргументов М-функции	**[S.name] = myfun**	**[C{:}] = myfun**

Составляющая массива ячеек	**{S.name}**	**{C{:}}**

Пример. Рассмотрим некоторые примеры использования списков значений:

формирование массива ячеек:

```
C = {1 2 3 4}
 C = [1]   [2]   [3]   [4]
```

преобразование массива ячеек в числовой массив:

```
A = [C{:}]
 A = 1   2   3   4
```

преобразование массива ячеек в трехмерный массив:

```
B = cat(3, C{:})
 B(:, :, 1) =   1
 B(:, :, 2) =   2
 B(:, :, 3) =   3
 B(:, :, 4) =   4
```

присвоение значений одному из полей массива записей:

```
[S(1:3).FIELD] = deal(5)
 S =
 3x1 struct array with fields:
       FIELD
 S(:).FIELD
 ans =   5
 ans =   5
 ans =   5
```

Списки значений играют важную роль при задании входных и выходных аргументов М-функций переменной длины, а также в операциях преобразования массивов записей и массивов ячеек друг в друга и числовые массивы.

Применение функций и операторов. Чтобы применить функции и операторы к содержимому ячеек, необходимо использовать индексацию и списки значений.

Пример. Допустим, что имеются следующие данные:

- массив размера 3x4, содержащий экспериментальные данные;
- строка из 15 символов, содержащая имя экспериментатора;
- массив размера 3x4x5, содержащий измерения последних 5 экспериментов.

Для многих приложений наилучшей организацией данных является структура. Однако в тех случаях, когда, как правило, требуется доступ только к части полей. более предпочтительной может оказаться организация данных в виде массива ячеек, к которой проще применять индексирование:

- Если данные организованы в виде структуры TEST, то для обращения к двум полям требуется два оператора присваивания:

  newdata = TEST.measure
  name = TEST.name
- Если данные организованы в виде массива ячеек TEST, то для обращения к двум полям требуется всего один оператор установления соответствия **deal**:

  [newdata, name] = deal(TEST{1:2})

Сформируем числовой массив A размера 3x3:

A = [0 1 2; 4 0 7; 3 1 2]

A =

0	**1**	**2**
4	**0**	**7**
3	**1**	**2**

Оценим 2-норму матрицы **A**, используя функцию **normest** и запишем результат в 2 отдельные ячейки массива **B**:

B = cell(1, 2);
[B{1:2}] = normest(A)

B =

[8.8826] [4]

В первой ячейке **B(1)** содержится оценка нормы; во второй **B(2)** - количество итераций, потребовавшихся для вычислений.

Пример. Сформируем массив ячеек **A** размера 1x3. Первая ячейка содержит матрицу размера 2x2; вторая - массив случайных чисел, распределенных по нормальному закону, размера 3x3, сформированный с помощью функции **randn**; третья - вектор-строку.

A{1, 1} = [1 2; 3 4];
A{1, 2} = randn(3, 3);
A{1, 3} = 1:5

A =

[2x2 double]	**[3x3 double]**	**[1x5 double]**	**[]**
[]	**[]**	**[]**	**[]**
[]	**[]**	**[]**	**[]**

Теперь применим функцию суммирования к массиву **A{1, 1}**

B = sum(A{1, 1})

B = 4 6

Чтобы применить функцию **sum** к содержимому ячеек, надо использовать цикл:

```
for i = 1:length(A)
   M{i} = sum(A{1, i});
end
M
M = [1x2 double] [1x3 double] [15] [0]
celldisp(M)
M{1} = 4   6
M{2} = -1.9728   0.3321   1.4788
M{3} = 15
M{4} = 0
```

7.4 Вложенные массивы ячеек

Допускается, что ячейка может содержать массив ячеек и даже массив массивов ячеек. Массивы, составленные из таких ячеек, называются вложенными. Ячейки, которые содержат данные, отличные от массива ячеек, называются листьями, а содержащие массивы ячеек - ветвями.

Сформировать вложенные массивы ячеек можно с помощью последовательности фигурных скобок, функции **cell** или операторов присваивания. Для уже сформированных массивов можно получить доступ и манипулировать отдельными ячейками, подмассивами ячеек или элементами самих ячеек.

Применение фигурных скобок. Для создания вложенных массивов ячеек можно применять фигурные скобки.

Пример.

```
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 []}}}
cellplot(A)
```

Заметим, что в правой части последнего оператора присваивания использовано 3 пары фигурных скобок: первая пара определяет ячейку **A(1, 2)** массива **A**, вторая задает внутренний массив ячеек размера 2x2, который, в свою очередь, содержит ячейку {17 [ ]} (смотри рисунок 7.5).

Применение функции cell. Для формирования вложенного массива ячеек с помощью функции cell выполним следующие операции:

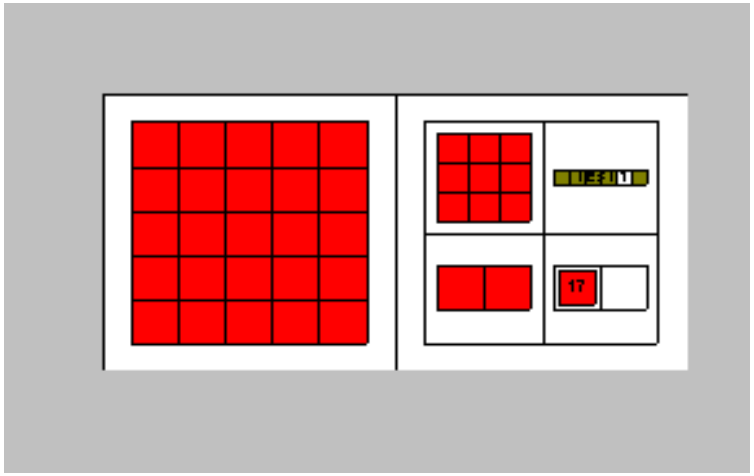


Рисунок 7.5

1. Создадим пустой массив ячеек размера 1x2:

A = cell(1, 2);

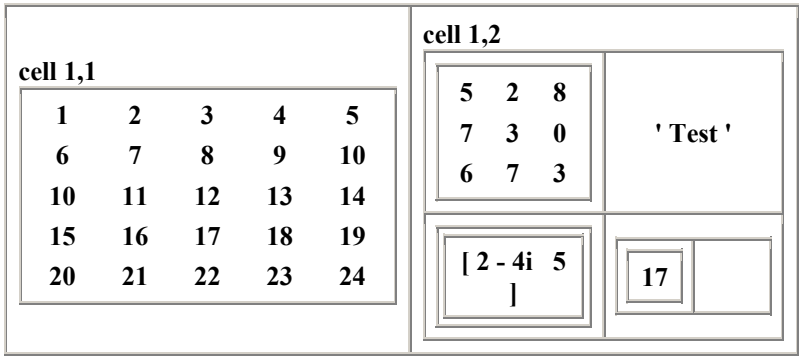
2. Создадим пустой массив ячеек **A(1, 2)** размера 2x2 внутри массива **A**:

A(1, 2) = {cell(2, 2)};

3. Заполним массив **A**, включая вложенный массив, с помощью операторов присваивания:

```
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};
```

Обратите внимание на использование фигурных скобок для последнего уровня вложенности. Это обусловлено тем, что необходимо обратиться к содержанию ячейки внутри массива ячеек. И наконец, можно сформировать вложенные массивы ячеек простым присваиванием значений его элементам, как это сделано выше на шаге 3.



Индексирование вложенных ячеек. Для того чтобы проиндексировать вложенные ячейки, необходимо использовать объединение индексов. Первое множество индексов определяет доступ к верхнему уровню ячеек, а последующие индексные выражения, заключенные в фигурные скобки, задают доступ к низшим уровням.

Пример. Приведенный выше массив ячеек **A** имеет 3 уровня вложенности:

- для доступа к числовому массиву размера 5x5 в ячейке (1, 1) надо использовать обращение - **A{1, 1}**;
- для доступа к числовому массиву размера 3x3 в позиции (1, 1) ячейки (1, 2) надо использовать обращение - **A{1, 2}{1, 1};**
- для доступа к элементу (2, 2) предыдущего числового массива надо использовать обращение **- A{1, 2}{1, 1}(2, 2);**
- для доступа к массиву ячеек размера 2x2 в ячейке (1, 2) надо использовать обращение - **A{1, 2};**
- для доступа к пустой ячейке в позиции (1, 2) ячейки (2, 2), вложенной в ячейку A(1, 2), надо использовать обращение - **A{1,2}{2,2}{1,2}.**

7.5 Работа с массивами различных типов

Преобразование массивов ячеек в многомерные массивы. Для преобразования массивов ячеек в многомерные числовые массивы необходимо использовать циклы **for.**

Пример. Сформируем 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];
F =
```

[2x2 double] **[2x2 double]**

[2x2 double] **[2x2 double]**

Теперь требуется 3 цикла **for**, чтобы преобразовать содержимое массива ячеек **F** в трехмерный 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
NUM
NUM(:, :, 1) =

     1     2
     3     4

NUM(:, :, 2) =

     7     8
     4     1

NUM(:, :, 3) =

    -1     0
     0     1

NUM(:, :, 4) =
```

0 + 4.0000I	**3.0000 + 2.0000I**
1.0 - 8.0000I	**5.0000**

Точно также необходимо использовать циклы **for**, чтобы разместить содержимое числового массива в ячейках:

```
G = cell(1,16);
for m = 1:16
  G{m} = NUM(m);
end
G
G =
  Columns 1 through 11
  [1] [3] [2] [4] [7] [4] [8] [1] [-1] [0] [0]
  Columns 12 through 16
  [1] [0+ 4.0000I] [1.0000- 8.0000I] [3.0000+ 2.0000I] [5]
```

Массивы ячеек, содержащих структуры. Для того чтобы объединить структуры с разными архитектурами полей, удобно использовать массивы ячеек.

```
Пример.          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];
celldisp(c_str)
c_str{1} =
    label: '12/2/94 - 12/5/94'
    obs: [2x4 double]
c_str{2} =
    xdata: [-0.0300 0.4100 1.9800 2.1200 17.1100]
    ydata: [-3 5 18 0 9]
    zdata: [0.6000 0.8000 1 2.2000 3.4000]
```

Ячейка **{1}** массива **c_str** содержит структуру из двух полей: поле **label** - строка, поле **obs** - числовой массив размера 2x4: ячейка **{2}** - три поля с числовыми векторами.

При построении массивов ячеек, включающих структуры, необходимо использовать контекстную индексацию. Точно также кон-

текстная индексация требуется для доступа к содержимому структур (массивов записей) внутри ячеек.

Синтаксис контекстной индексации имеет форму

cell_array{index}.field

Пример. Чтобы получить доступ к полю **label** ячейки **{1}** следующей структуры (рисунок 7.6), необходимо использовать обращение **c_str{1}.label**.

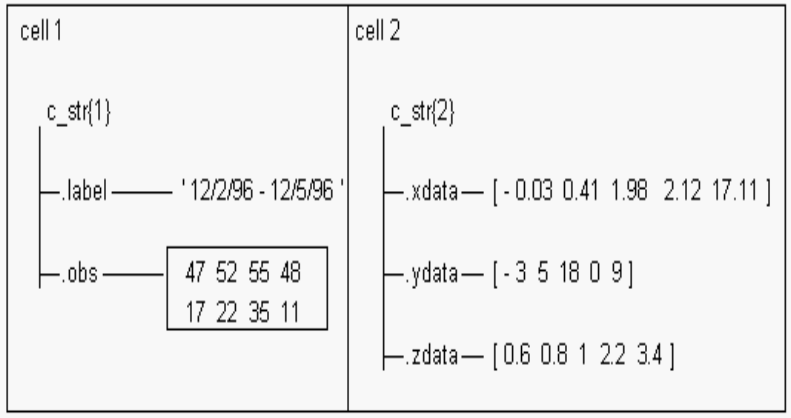


Рисунок 7.6

Многомерные массивы ячеек. Как и в случае числовых массивов, многомерная конструкция массива ячеек является расширением его двумерного аналога. Для формирования многомерного массива ячеек можно просто использовать функцию **cat**.

Пример. Сформируем следующий 3-мерный массив ячеек **C**, объединяющий 2-мерные массивы ячеек **A** и **B:**

```
A{1, 1} = 'Name';
A{1, 2} = [4 2; 1 5];
A{2, 1} = 2-4i;
A{2, 2} = 7;
B{1, 1} = 'Name2';
B{1, 2} = [ 3 5 ]';
B{2, 1} = 0:1:3;
B{2, 2} = 3;
C = cat(3, A, B);
```

Сформированный массив ячеек показан на рисунке 7.7 .

	cell 1,1,2 **'Name2'**	**cell 1,2,2** **3** **5**
	cell 2,1,2 **[0 1 2 3]**	**cell 2,2,2** **3**
cell 1,1,1 **'Name'**	**cell1,2,1** **4 2** **1 5**	
cell 2,1,1 **2 - 4i**	**cell 2,2,1** **7**	

Рисунок 7.7. Массив ячеек размера 2x2x2

7.6 Функции и команды обработки массивов ячеек

CELL - Создать массив ячеек

Синтаксис:

c = cell(n)
c = cell(m, n)
c = cell([m n])
c = cell(m, n, p,...)
c = cell([m n p ...])
c = cell(size(A))

Описание. Функция **c = cell(n)** создает массив ячеек, состоящих из пустых матриц, размера **nxn**. Сообщение об ошибке возникает в том случае, если **n** не является скаляром.

Функции **c = cell(m, n)** и **c = cell([m, n])** создают массив ячеек, состоящих из пустых матриц, размера **mxn**. Аргументы **m** и **n** должны быть скалярными.

Функции **c = cell(m, n, p,...)** и **c = cell([m n p ...])** создают многомерный массив ячеек, состоящих из пустых матриц, размера **mxnxp ...** . Аргументы **m, n, p**, ... должны быть скалярными.

Функция **c = cell(size(A))** создает массив ячеек, состоящих из пустых матриц, того же размера, что и массив **A**.

Пример.

```
A = ones(2, 2)
A =

     1   1

     1   1

c = cell(size(A))
c =

    [ ]    [ ]

    [ ]    [ ]
```

Сопутствующие функции. **ONES, RAND, RANDN, ZEROS**.

CELLDISP - Вывести на экран содержимое массива ячеек

Синтаксис:

celcelldisp(C)

Описание. Команда **celldisp(C)** выводит на экран содержимое массива ячеек.

Пример. Выведем на экран содержимое следующего массива ячеек размера 2x3:

```
C = {[1 2] 'Tony' 3+4i; [1 2; 3 4] -5 'abc'};
celldisp(C)
C{1,1} =   1   2
C{2,1} =

     1     2

     3     4

  C{1,2} =Tony
C{2,2} =    -5
C{1,3} = 3.0000+ 4.0000i
C{2,3} = abc
```

Сопутствующие функции. **CELLPLOT.**

CELLPLOT - Вывести на экран графическую структуру массива ячеек

Синтаксис:

cellplot(C)
cellplot(C, 'legend')
handles = cellplot(...)

Описание. Команда **cellplot(C)** выводит в графическое окно содержимое массива ячеек **C**. Закрашенные прмоугольники соответствуют векторам и массивам, скалярные величины и короткие строки символов выводятся в виде текста.

Команда **cellplot(C, 'legend')** кроме графического изображения выводит описание цветов для различных типов данных.

Функция **handles = cellplot(C)** выводит в графическое окно содержимое массива ячеек **C** и возвращает вектор поддержек.

Ограничение. Команда **cellplot** может выводить только графическую струтктуру для двумерных массивов ячеек.

Пример. Выведем на экран содержимое следующего массива ячеек размера 2x3 (рисунок 7.8):

```
c{1, 1} = '2-by-2';
c{1, 2} = 'eigenvalues of eye(2)';
c{2, 1} = eye(2);
c{2, 2} = eig(eye(2));
cellplot(c, 'legend')
```

Сопутствующие функции. **CELLDISP**.

CELLSTR - Преобразовать массив строк в массив символьных ячеек

Синтаксис:

C = cellstr(S)

Описание. Функция **C = cellstr(S)** преобразует массив строк **S** в массив символьных ячеек **C**.

Пример. Задан следующий массив строк **S** размера 3x4:

```
S = ['abc ' ;
     'defg' ;
     'hi ']
S = abc
    defg
     hi
```

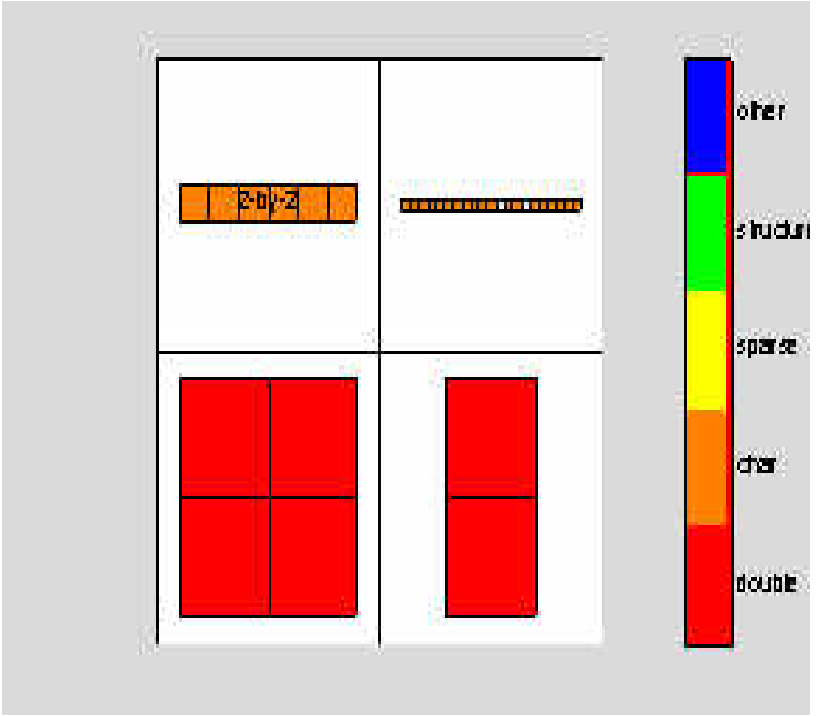


Рисунок 7.8. Отображение массива ячеек размером 2x3

Функция **C = cellstr(S)** возвращает массив ячеек размера 3x1:

```
 C = cellstr(S)
  C = 'abc'
      'defg'
      'hi'
```

Сопутствующие функции. **ISCELLSTR, STRINGS**.

DEAL - Установить соответствие между входами и выходами

Синтаксис:

```
[Y1, Y2, Y3,...] = deal(X)
[Y1, Y2, Y3,...] = deal(X1, X2, X3,...)
```

Описание. Функция **[Y1, Y2, Y3,...] = deal(X)** копирует единственный вход на все выходы, реализуя следующее соответствие **Y1 = X, Y2 = X, Y3 = X, …** .

Функция **[Y1, Y2, Y3,...] = deal(X1, X2, X3,...)** устанавливает следующее соответствие между входами и выходами **Y1 = X1, Y2 = X2, Y3 = X3, …** .

Замечание. Функция **deal** исключительно полезна при применении в следующих конструкциях при работе с массивами ячеек и массивами записей:

- оператор **[S.field] = deal(X)** присваивает всем полям структуры **S** с именем **field** значение **X**. Если **S** не существует, надо использовать оператор **[S(1:m).field] = deal(X);**
- оператор **[X{ : }] = deal(A.field)** копирует поля структуры **A** с именем **field** в массив ячеек **X**. Если **X** не существует, надо использовать оператор **[X{1:m}] = deal(A.field);**
- оператор **[Y1, Y2, Y3, ...] = deal(X{ : })** копирует содержимое массива ячеек **X** в отдельные переменные **Y1, Y2, Y3, ... ;**
- оператор **[Y1,Y2,Y3,...] = deal(S.field)** копирует поля структуры **S** с именем **field** в отдельные переменные **Y1, Y2, Y3, ...**

Пример. Скопировать содержимое массива ячеек **C** размера 1x4 в 4 выходные переменные:

```
C = {rand(3) ones(3, 1) eye(3) zeros(3, 1)};
[a, b, c, d] = deal(C{ : })
a =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214

b =
     1
     1
     1
c =
     1     0     0
     0     1     0
     0     0     1

d =
     0
     0
     0
```

Скопировать содержимое всех полей **name** структуры **A** размера 1x2 в отдельные переменные:

```
A.name = 'Pat';      A.number = 176554;
A(2).name =          A(2).number =
'Tony';              901325;
```

```
[name1, name2] = deal(A(:).name)
name1 = Pat
name2 = Tony
```

Сопутствующие функции. **VARARGIN, VARARGOUT, CELL2STRUCT**, **STRUCT2CELL, NUM2CELL, CAT.**

ISCELL - Выявление массива ячеек

Синтаксис:

k = iscell(C)

Описаниею. Функция **k = iscell(C)** возвращает логическое **TRUE (1),** если **C** - массив ячеек, и логическое FALSE (0) - в противном случае.

Сопутствующие функции: функции группы **IS*.**

NUM2CELL - Преобразовать массив чисел в массив ячеек

Синтаксис:

C = num2cell(A)

C = num2cell(A, dims)

Описание. Функция **C = num2cell(A)** преобразует массив **A** в массив ячеек, размещая каждый элемент массива **A** в отдельной ячейке. Размер массива ячеек будет совпадать с размерами массива **A**.

Функция **C = num2cell(A, dims)** преобразует только те элементы массива **A** в массив ячеек, которые заданы вторым аргументом.

Пример. Рассмотрим некоторый массив чисел **A = rand(3):**

```
A = rand(3)
A =

    0.4447    0.9218    0.4057
    0.6154    0.7382    0.9355
    0.7919    0.1763    0.9169
```

и применим к нему следующие преобразования:

```
celldisp(num2cell(A, 2))
ans{1} = 0.4447   0.9218   0.4057
ans{2} = 0.6154   0.7382   0.9355
ans{3} = 0.7919   0.1763   0.9169
celldisp(num2cell(A, [1 3]))

ans{1} =    ans{2} =    ans{3} =

    0.4447      0.9218      0.4057

    0.6154      0.7382      0.9355

    0.7919      0.1763      0.9169
```

В последнем случае столбцы размещены по отдельным ячейкам.

Сопутствующие функции: **CAT**.

CELL2STRUCT - Преобразовать массив ячеек в массив записей

Синтаксис. **lS = cell2struct(C, fields, dim)**

Описание. Функция **S = cell2struct(C, fields, dim)** преобразует массив ячеек **C** в массив записей **S** вдоль размерности **dim**, сохраняя размер массива **C** по этой размерности в записи структуры. Аргумент **fields** может быть массивом строк или массивом строковых ячеек.

Пример. Рассмотрим массив ячеек размера 1 по первой размерности и размера 3 по второй

```
c = {'tree', 37.4, 'birch'}
 c =
    'tree'    [37.4000]   'birch'
```

и преобразуем его по второй размерности в структуру с полями **f = {'category', 'height', 'name'}:**

```
f = {'category', 'height', 'name'};
 s = cell2struct(c, f, 2)
 s =
  category: 'tree'
     height: 37.4000
     name: 'birch'
```

Сопутствующие функции: **FIELDNAMES, STRUCT2CELL**.

STRUCT2CELL - Преобразовать массив записей в массив ячеек

Синтаксис:

C = struct2cell(S)

Описание. Функция **C = struct2cell(S)** преобразует массив записей **S** размера **mxn** (с **p** полями) в массив ячеек **C** размера **pxmxn**.

Если массив записей **S** многомерный, то массив ячеек **C** имеет размер **[p size(S)].**

Пример. Следующие операторы

```
clear S,
S.category = 'tree'; S.height = 37.4; S.name = 'birch';
```

создают структуру

```
S =
  category: 'tree'
    height: 37.4000
     name: 'birch
```

Преобразуем эту структуру в массив ячеек

```
C = struct2cell(S)
C =
    'tree'
    [37.4000]
    'birch'
```

Сопутствующие функции: **CELL2STRUCT, FIELDS**.

LISTS - Определение списков значений

Синтаксис:

help lists

Описание. Команда **help lists** выводит на экран следующий комментарий к определению и использованию списков значений при работе с массивами записей и массивами ячеек.

Извлечение множественных данных из массивов записей и массивов ячеек осуществляется с помощью списков значений.

Список значений для массива записей - это объединение одноименных полей **S.name = [S(1).name S(2).name … S(end).name].**

Список значений для массива ячеек - это объединение ячеек **C{:} = [C{1} C{2} … C{end}].**

Конструкции вида **S(m:n).name, C{m:n}** также представляют собой списки значений.

Списки значений используются в следующих случаях:

- в командной строке для вывода значений на экран - **S.name, C{:};**
- при вызове М-функций - **myfun(x, y, S.name), myfun(x, y, C{:});**
- в операциях конкатенации **- [S.name], [C{:}];**
- в списках выходных аргументов функции - **[S.name] = myfun, [C{:}] = myfun**;
- как составляющие массива ячеек - **{S.name}, {C{:}}.**

Пример. Рассмотрим некоторые примеры использования списков значений:

- формирование массива ячеек :

C = {1 2 3 4}

C = [1] [2] [3] [4]

- преобразование массива ячеек в числовой массив :

A = [C{:}]

A = 1 2 3 4

- преобразование массива ячеек в трехмерный массив :

B = cat(3, C{:})

B(:, :, 1) = 1

B(:, :, 2) = 2

B(:, :, 3) = 3

B(:, :, 4) = 4

- присвоение значений одному из полей массива записей :

[S(1:3).FIELD] = deal(5)

S =

3x1 struct array with fields:

FIELD

S(:).FIELD

ans = 5

ans = 5

ans = 5

В результате выполненных операций были сформированы следующие массивы:

Whos

Name	Size	Bytes	Class
A	**1x4**	**32**	**double array**
B	**1x1x4**	**32**	**double array**
C	**1x4**	**400**	**cell array**
S	**3x1**	**332**	**struct array**

Grand total is 22 elements using 796 bytes

Общее количество элементов - 22; используют 796 байтов

Сопутствующие функции: **CAT, CELL2STRUCT, DEAL, NUM2CELL, STRUCT2CELL, VARARGIN, VARARGOUT.**

VARARGIN - Список входных аргументов переменной длины

Синтаксис:

z = function myfun(x, y, varargin)

Описание. Переменная **varargin** позволяет объединить любое количество входных аргументов; она представляет собой массив ячеек, который содержит аргументы-опции вызываемой функции. Эта переменная должна быть последней в списке входов, а ее написание допускается только строчными буквами.

Пример. Рассмотрим М-функцию

```
function myplot(x, varargin)
    plot(x, varargin{:})
```

Она объединяет все входные аргументы, начиная со второго в одну переменную **varargin**. В свою очередь, при обращении к функции **plot** используется список значений **varargin{:},** чтобы передать все задействованные аргументы. Например, при вызове функции **myplot** в форме:

myplot(sin(0:.1:1), 'color', [.5 .7 .3], 'linestyle', ':');

переменная **varargin** - это массив ячеек размера **1x4**, содержащий значения

'color', [.5 .7 .3], 'linestyle', ':'.

Сопутствующие функции: **VARARGOUT, NARGIN, NARGOUT, INPUTNAME, FUNCTION, LISTS.**

VARARGOUT - Список выходных аргументов переменной длины

Синтаксис:

[z, varargout] = myfun(x, y, n)

Описание. Переменная **varargout** позволяет объединить любое количество выходных аргументов; она представляет собой массив ячеек, который содержит аргументы-опции выхода функции. Эта переменная должна быть последней в списке выходов, а ее написание допускается только строчными буквами.

Переменная **varargout** не создается при вызове функции; ее необходимо создать при формировании выходов создаваемой М-функции, используя соответствующие операторы цикла, как это показано в нижеследующем примере

Пример. Рассмотрим М-функцию

```
function [s, varargout] = mysize(x)
    nout = max(nargout, 1)-1;
    s = size(x);
```

```
    for i=1:nout,
        varargout(i) = {s(i)};
    end
```

Здесь с использованием цикла объединуны все выходные аргументы, начиная со второго в одну переменную **varargout.**

Переменная **varargout** - это массив ячеек размера **1xnout**, содержащий значения размерностей выходного многомерного массива. Например, при обращении вида

[s, rows, cols, pages] = mysize(rand(4, 5, 3));

возвращаются значения

s = [4 5 3], rows = 4, cols = 5, pages = 3.

Сопутствующие функции: **VARARGIN, NARGIN, NARGOUT, FUNCTION, LISTS**.

8 Объектно-ориентированное программирование

Введение новых типов данных и операций, определяемых пользователем, характеризует подход, известный как объектно-ориентированное программирование. Эта глава описывает, как добавлять новые типы данных к системе MATLAB, создавая классы. В ней также объясняется, как создавать объекты и управлять ими, если они являются образцами классов MATLAB.

8.1 Объекты и классы

Классы и объекты позволяют добавлять новые типы данных и новые операции. Класс описывает тип переменной и определяет, какие операции и функции могут быть применены к этому типу переменной. Объект - это структура или образец некоторого класса.

В системе MATLAB определены 5 классов объектов:

double	Числовые массивы и матрицы, заданные в арифметике с плавающей точкой в формате удвоенной точности.
sparse	Двумерные действительные или комплексные разреженные матрицы
char	Массивы символов
struct	Массивы записей (структуры)
cell	Массивы ячеек

В свою очередь, пакеты прикладных программ (ППП) семейства продуктов MATLAB дают много примеров определения новых классов объектов. Например, в самой системе MATLAB используется встроенный класс inline, который дает простой способ определения встроенных функций для применения в программах вычисления квадратур, решения дифференциальных уравнений и вычисления нулей функции. ППП **Symbolic Math Toolbox** базируется на классе объектов **sym**, который позволяет выполнять вычисления с символьными переменными и матрицами. ППП **Control System Toolbox** использует класс объектов **lti** и три его подкласса **tf, zpk, ss**, которые поддерживают алгоритмы анализа линейных систем с постоянными параметрами.

Добавление классов осуществляется в рамках операционной среды системы MATLAB, которая обеспечивает возможность хранения созданных объектов и организации каталога М-файлов, которые определяют допустимые методы обработки для данного класса объектов. Каталог класса включает М-функции, которые определяют способ, каким операторы системы MATLAB, включая арифметические, обработки индексов, конкатенации применяются к объекту данного класса. Переопределение встроенных операторов для нового класса объектов в рамках объектно-ориентированного подхода называется переопределением методов.

В языке MATLAB отсутствует механизм объявления переменных. Например, оператор **A = zeros(10, 10)** формирует обычную матрицу размера 10x10, которая является объектом класса **double**. Точно также оператор **s = 'Hello world'** создает объект класса **char**.

То же самое относится и к вновь создаваемым классам. Никаких объявлений переменных или объектов не требуется. Объекты создаются динамически посредством вызова конструктора класса.

Далее в качестве типового рассматривается класс полиномов от одной переменной. Название класса и конструктора класса - **polynom**. В этом классе объект является полиномом, например, вида **p(x) = x^3 - 2x - 5**, который создается в результате вызова конструктора **polynom**, применяемого к вектору коэффициентов

```
p = polynom([1 0 -2 -5])
p = x^3 - 2*x - 5.
```

Структура объекта. Один из первых шагов при проектировании нового класса объектов - это выбор структуры данных для рассматриваемого класса. Объекты класса задаются в виде структур. Поля структуры и операции с полями видимы только внутри методов для данного класса.

Например, для класса **polynom** можно выбрать представление полиномиального объекта в виде вектор-строки, содержащей коэффициенты степеней переменной в убывающем порядке. Так что объект p, принадлежащий классу **polynom**, - это структура с единственным полем **p.c**, содержащим коэффициенты. Это поле доступно только для методов, описанных в подкаталоге **@polynom**.

Однако, это - не единственый способ представить полином. Коэффициенты полинома могут быть упорядочены по возрастанию степеней или представлены в виде вектор-столбца. Можно представить полином с точностью до скалярного множителя, определяя его нулями. Выбор той или иной среди этих альтернативных структур для такого простого объекта как полином не особенно существенен, труден или важен, но для более сложных объектов выбор структуры данных может играть очень важную роль. Например, специальная структура данных **sparse**, выбранная для разреженных матриц, существенно сокращает время выполнения операций над ними.

Каталог класса. М-файлы, определяющие методы для объектов данного класса объединяются в каталог класса, название которого задается как **@<имя_класса>.** Это означает, что М-файлы, определяющие методы для класса **polynom**, должны быть размещены в каталоге **@polynom**.

Каталог класса обязательно является подкаталогом каталога, описанного в пути доступа системы MATLAB, но не самим каталогом. Например, каталог **@inline** - это подкаталог каталога **toolbox/matlab/funfun**, а каталог **@sym** - это подкаталог каталога **toolbox/symbolic**. Новый каталог **@polynom** должен быть подкатало-

гом рабочего каталога системы MATLAB или собственного персонального каталога, который должен быть добавлен к пути доступа.

Конструктор класса. Каталог класса должен обязательно содержать М-файл, называемый конструктором класса. Название конструктора должно совпадать с названиями класса и каталога без префикса **@.** Конструктор создает объекты, используя данные в виде массива записей (структуры) и приписывая им метку класса.

Рассмотрим конструктор полиномов **@polynom/polynom.m**:

```
function  p = polynom(v)
%POLYNOM Конструктор полиномов.
% Функция p = polynom(v) формирует полином, используя вектор v,
% содержащий коэффициенты степеней переменной x, расположенные
% в убывающем порядке.
   if nargin == 0
      p.c = [ ];
      p = class(p, 'polynom');
   elseif isa(a, 'polynom')
      p = v;
   else
      p.c = v(:).';
      p = class(p,'polynom');
   end
```

Система MATLAB позволяет вызывать конструктор без аргументов. В этом случае конструктор должен создать шаблон объекта, как правило, с пустыми полями. Также возможно, что конструктор будет вызываться с аргументом входа, который уже является объектом данного класса. В этом случае конструктор обычно возвращает входной аргумент. Функция **isa** проверяет эту ситуацию. Если аргумент входа существует и не принадлежит классу **polynom**, то он переформируется так, чтобы быть вектором-строкой и присвавается полю .c результата. И наконец, функция **class** используется, чтобы приписать метку результату, которая определяет его как **polynom**.

Следующий оператор является примером использования конструктора **polynom**

```
p = polynom([1 0 -2 -5])
 p = x^3 - 2*x - 5
```

и формирует полином с заданными коэффициентами.

В общем виде функция конструктора удовлетворяет следующим свойствам:

- в отсутствии аргументов возвращается шаблон объекта;

- если вход является объектом данного класса, то он же является выходом;
- преобразует вход к требуемой форме;
- присваивает значения различным полям структуры;
- использует функцию **class** для того, чтобы приписать объекту соответствующую метку.

Функции isa и class. Эти функции используются конструктором, но могут применяться и вне каталога класса.

Функция **isa(a, 'class_name')** проверяет, принадлежит ли объект а данному классу.

Пример. Каждое из следующих выражений истинно

```
isa(pi, 'double')      isa('hello', 'char')      isa(p, 'polynom')

ans = 1                ans = 1                   ans = 1
```

При использовании вне контекста методов функция **class** допускает только один аргумент.

Команда **class(a)** возвращает строку, содержащую имя класса для объекта **a.**

Пример. Последовательность операторов возвращает соответственно

```
class(pi)              class('hello')            class(p)

ans = 'double'         ans ='char'               ans ="polynom'
```

Объекты и массивы. В системе MATLAB5 к основному объекту системы - массиву чисел добавлены новые объекты **struct** и **cell**. Поскольку объект класса описывается структурой, то допустимы следующие способы использования массива в качестве объекта некоторого класса:

- поле объекта - массив;
- объект - массив;
- элементы массива - объекты некоторого класса.

Рассмотрим эту концепцию на примере полиномов Чебышева, алгоритм построения которых описывается следующими рекуррентными соотношениями:

T0 (x) = 1
T1 (x) = x
Tn+1 (x) = 2 x Tn (x) -Tn-1 (x), n > 1

Если поле объекта - массив, то объект **polynom**, описанный выше, уже имеет в своей структуре поле, которое является массивом,

в данном случае вектор-строкой, содержащей полиномиальные коэффициенты. Понятие “полиномиальные” можно обобщить так, чтобы сами коэффициенты также были векторами. Тогда объект **polynom** должен иметь поле **p.c,** которое являлось бы либо двумерным массивом, либо массивом ячеек для размещения векторов коэффициентов. Все методы для такого обобщенного объекта должны обрабатывать такие коэффициенты. Это - возможно не лучший способ представления полиномиальной последовательности, хотя это вполне жизнеспособная структура данных для таких объектов.

Если сам объект - массив, то объект **polynom** - это массив структур. Внутри методов полиномы должны обозначаться как **p(k),** и каждый полином должен иметь собственное поле коэффициентов p(k).c. Каждый метод должен использовать цикл для обработки всех полиномов последовательности. Это усложняет применение методов, но упрощает работу с объектами.

Если используется массив объектов, то объект **polynom** можно использовать без каких-либо изменений. Чтобы сгенерировать последовательность полиномов, надо просто работать с полиномами, хранящимися в массиве ячеек. Это - вероятно, лучший способ генерации полиномов Чебышева.

Рассмотрим реализацию последнего подхода. Введем аргумент **x:**

```
x = polynom([1 0])
 x = x
```

Сгенерируем и сохраним в массиве ячеек **T** 10 первых полиномов Чебышева:

```
T{1} = 1;
T{2} = x;
for n = 2:11
  T{n+1} = 2*x*T{n} - T{n-1};
end
```

В результате будут сгенерированы следующие полиномы:

```
for n = 2:11
   T{n}
end
ans = x
ans = 2*x^2 - 1
ans = 4*x^3 - 3*x
ans = 8*x^4 - 8*x^2 + 1
ans = 16*x^5 - 20*x^3 + 5*x
ans = 32*x^6 - 48*x^4 + 18*x^2 - 1
ans = 64*x^7 - 112*x^5 + 56*x^3 - 7*x
ans = 128*x^8 - 256*x^6 + 160*x^4 - 32*x^2 + 1
```

ans = 256*x^9 - 576*x^7 + 432*x^5 - 120*x^3 + 9*x

ans = 512*x^10 - 1280*x^8 + 1120*x^6 - 400*x^4 + 50*x^2 - 1

Построение последнего из них показывает, что на интервале **[-1 1]** он напоминает гребень и характеризуется быстрым ростом вне этого интервала (смотри рисунок 8.1).

plot(T{11})

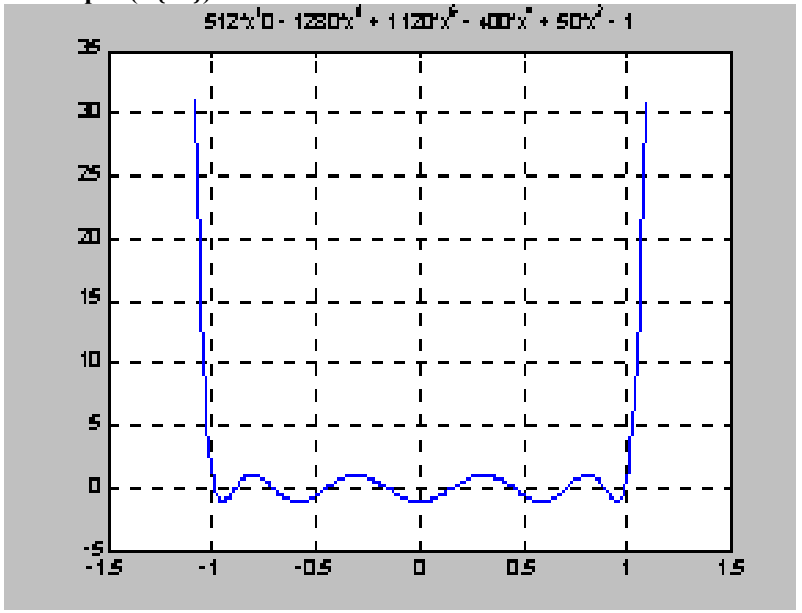


Рисунок 8.1

Преобразование классов. Вызов функции преобразования класса имеет вид

b = class_name(a),

где **a** - объект некоторого класса, отличного от **class_name**. В этом случае система MATLAB ищет метод с именем **class_**name в каталоге классов для объекта a. Такой метод преобразовывает объект одного класса в объект другого класса. Если входной объект уже является объектом класса **class_name**, то система MATLAB вызывает функцию конструктора, который просто возвращает этот вход.

Две из наиболее важных функций преобразования классов - это **double** и **char**. Преобразование к классу **double** создает традиционный массив системы MATLAB, хотя это может и не отражать тре-

буемого соответствия для некоторых классов. Преобразование к классу **char** полезно для вывода на печать.

Для класса объектов **polynom** функция преобразования к классу **double** - это очень простой М-файл **@polynom/double.m**, который восстанавливает вектор коэффициентов:

```
function c = double(p)
  % POLYNOM/DOUBLE Преобразование объекта polynom в вектор
  % коэффициентов.
  % Функция c = DOUBLE(p) преобразовывает объект polynom в вектор c,
  % содержащий коэффициенты полинома по степеням переменной x
  % в убываающем порядке.
c = p.c;
```

Для ранее рассмотренного полинома функция возвращает следующий вектор коэффициентов

```
double(p)
ans = 1 0 -2 -5
```

Преобразование к классу **char** - это ключевой метод, потому что он формирует строку символов, состоящую из степеней независимой переменной x. Фактически, как только переменной **x** присвоено значение, строка становится синтаксически правильным выражением системы MATLAB. В данном случае это метод **@polynom/char.m.**

```
function s = char(p)
% POLYNOM/CHAR CHAR(p) формирует строковое представление для полинома
p. c = p.c;
if all(c == 0)
   s = '0';
 else
   d = length(p.c)-1;
   s = [ ];
   for a = c;
     if a ~= 0;
       if ~isempty(s)
         if a > 0
           s = [s ' + '];
         else
           s = [s ' - '];
           a = -a;
         end
       end
```

```
            if a ~= 1 | d == 0
              s = [s num2str(a)];
              if d > 0
                s = [s '*'];
              end
            end
            if d >= 2
              s = [s 'x^' int2str(d)];
            elseif d == 1
              s = [s 'x'];
            end
          end
          d = d - 1;
        end
      end
```

Для ранее рассмотренного полинома функция

char(p)

возвращает результат

ans = x^3 - 2*x - 5.

Вывод на терминал. Метод с именем **display** вызывается всякий раз, когда оказывается, что исполняемый оператор не заканчивается точкой с запятой. Для многих классов метод **display** может просто выводить на экран имя переменной, а затем использовать конвертор **char**, чтобы печатать содержимое или значение переменной. Для этого служит метод **@polynom/display.m.** Тело этой функции может быть без изменения использовано в каталогах других методов.

```
function display(p)
% POLYNOM/DISPLAY Вывести на экран терминала объект класса polynom.
disp(' ');
disp([inputname(1),' = '])
disp(' ');
disp([' ' char(p)]) disp(' ');
```

Пример.

```
display(p)
 p = x^3 - 2*x - 5
```

При работе с полиномами полезно иметь объект **x**, который представляет собой независимую переменную **x**. Это реализуется оператором

```
x = polynom([1 0])
 x = x
```

8.2 Вызов методов

При работе с объектами и методами MATLAB использует специальное множество правил, чтобы гарантировать вызов требуемой функции. Если, по крайней мере, один из аргументов - объект, MATLAB рассматривает список параметров слева направо, чтобы определить их старшинство. (Для операторов равного старшинства выбирается крайний левый.) Затем к нему применяются следующие правила:

1. Если имя вызываемой функции совпадает с именем встроенной функции, то проверяется, существует ли переопределенная версия этой функции для этого класса, а затем - для родительского. Если ни один из этих случаев не имеет место, выдается ошибка.
2. Если имя функции совпадает с названием каталога классов, проверяется, не является ли эта функция конвертором и если да, то вызывает ее. В противном случае вызывается конструктор класса.
3. Если оба случая не подходят, то анализируются следующие возможности:

- если есть метод соответствующего типа, то вызывается он;
- если есть метод родительского класса, то вызывается он;
- если есть функция с таким именем в пути доступа, то вызывается она;
- генерируется ошибка.

Частные методы и функции. Каталоги классов могут иметь связанные с ними частные каталоги. Такие каталоги могут содержать как частные методы, которые работают с объектами данного класса, так и частные функции, которые не работают с объектами, но выполняют общие вычисления. Можно устанавливать частный каталог под каталогом класса точно также, как создается любой частный каталог, то есть просто создать каталог, именуемый **private,** внутри каталога **@class_name**.

Отладка методов. Можно использовать команды отладки для методов точно так же, как они используются при работе с М-файлами. Просто надо включить название каталога класса перед названием метода. Например, **dbstop class/method**. Заметим, что при использовании отладчика, заданная по умолчанию форма команды может видеть частные функции и методы внутри каталога класса.

8.3 Переопределение классов

Во многих случаях можно изменить поведение операторов и функций системы MATLAB, когда в качестве аргумента выступает объект. Это осуществляется путем переопределения соответствующих функций. Переопределение класса открывает возможность обработки с помощью этой функции различных типов данных при произвольном количестве входных аргументов.

Переопределение арифметических операций. Каждый встроенный оператор в системе MATLAB имеет имя. Поэтому любой оператор может быть переопределен путем создания M-файла с соответствующим названием в каталоге классов. Например, если **p** или **q** - полиномы, то выражение вида **p + q** задает вызов метода **@polynom/plus.m**, если он существует.
В данном случае это M-файл вида:

```
function r = plus(p, q)
% POLYNOM/PLUS Реализовать операцию p + q для объектов polynom..
p = polynom(p);
q = polynom(q);
k = length(q.c) - length(p.c);
r = polynom([zeros(1, k) p.c] + [zeros(1, -k) q.c]);
```

Прежде всего M-функция метода преобразует оба аргумента входа к классу **polynom**. Это гарантирует, что выражения типа **p + 1**, которые включают как объект типа **polynom**, так и объект типа **double**, будут вычисляться правильно.

Функция затем обращается к двум векторам коэффициентов и в случае необходимости, дополняет один из них с нулями, чтобы выравнять их длины. Фактическое сложение - просто сумма двух векторов коэффициентов, к которой применяется в третий раз конструктор **polynom**, чтобы сформировать правильный тип результата.

Другой пример - это метод **@polynom/mtimes.m**, который вычисляет произведение полиномов **p*q**. Буква **m** в начале имени функции обусловлена тем, что это есть переопределение функции умножения матриц. Умножение двух многочленов - просто свертка их векторов коэффициентов.

```
function r = mtimes(p, q)
% POLYNOM/MTIMES Реализует операцию умножения p * q для объектов
% polynom.
```

p = polynom(p);
q = polynom(q);
r = polynom(conv(p.c, q.c));

Операторы

q = p + 1
r = p*q,

используя описанные функции, дают следующие результаты

q = x^3 - 2*x - 4
r = x^6 - 4*x^4 - 9*x^3 + 4*x^2 + 18*x + 20.

Переопределение операторов. В нижеприведённой таблице указаны символьные имена для большинства встроенных операторов системы MATLAB'.

Оператор	***Имя М-файла***	***Описание***
a + b	**plus(a,b)**	Двоичное сложение
a - b	**minus(a, b)**	Двоичное вычитание
-a	**uminus(a)**	Унарное вычитание
+a	**uplus(a)**	Унарное сложение
a.*b	**times(a, b)**	Поэлементное умноже-ние
a*b	**mtimes(a, b)**	Умножение матриц
a./b	**rdivide(a, b)**	Правое поэлементное деление
a.\b	**ldivide(a, b)**	Левое поэлементное деление
a/b	**mrdivide(a, b)**	Правое деление матриц
a\b	**mldivide(a, b)**	Левое деление матриц
a.^b	**power(a, b)**	Поэлементное возведе-ние в степень
a^b	**mpower(a, b)**	Возведение матрицы в степеннь
a b	**lt(a, b)**	Меньше

a b	**gt(a, b)**	Больше
a = b	**le(a, b)**	Меньше или равно
a = b	**ge(a, b)**	Больше или равно
a ~= b	**ne(a, b)**	Не равно
a == b	**eq(a, b)**	Тождественно
a & b	**and(a, b)**	Логическое И
a | b	**or(a, b)**	Логическое ИЛИ
~a	**not(a, b)**	Логическое НЕТ
a:d:b **a:b**	**colon(a, d, b)** **colon(a, b)**	Формирование вектора
a'	**ctranspose(a)**	Транспонирование матрицы
a.'	**transpose(a)**	Транспонирование массива
command window output	**display(a)**	Вывод на терминал
[a b]	**horzcat(a, b, ...)**	Объединение в строку
[a; b]	**vertcat(a, b, ...)**	Объединение в столбец
a(s1,s2,...sn)	**subsref(a, s)**	Индексная ссылка
a(s1,...,sn) = b	**subsasgn(a, s, b)**	Индексное выражение
b(a)	**subsindex(a, b)**	Индекс подмассива

Переопределение функций. Можно переопределить любую М-функцию, создавая функцию с тем же именем в каталоге класса. Когда функция применяется к объекту, MATLAB прежде всего просматривает каталог соответствующего класса, а уже потом другие пути доступа. Чтобы переопределить функцию **plot** для некоторого класса, надо просто разместить М-файл **plot.m** в соответствующем каталоге класса. Далее приведено несколько примеров, относящихся к объектам из класса **polynom**.

В состав ядра системы MATLAB включены функции для работы с полиномами, описываемыми векторами их коэффициентов. При создании новых полиномиальных объектов эти функции должны быть переопределены. Во многих случаях переопределяемая функция может просто применять исходную функцию к полю соответствующей структуры. Например, имеется метод **@polynom/roots.m:**

```
function r = roots(p)
% POLYNOM/ROOTS. ROOTS(p) - это вектор, содержащий корни полинома p.
r = roots(p.c);
```

Оператор **roots(p)** дает следующий результат

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

Функция **polyval** вычисляет полином для заданного множества точек. В данном случае это метод **@polynom/polyval.m.** Он основан на методе Горнера.

```
function y = polyval(p,x)
% POLYNOM/POLYVAL POLYVAL(p,x) вычисляет значение полинома p в точке x
y = 0;
for a = p.c
  y = y.*x + a;
end
```

Обе эти функции используются при переопределении функции **plot**. Область изменения независимой переменной выбирается, чтобы быть немного большей, чем интервал, содержащий все действительные корни. Затем используется функция **polyval**, чтобы вычислить значения полинома в нескольких сотнях точек области.

Результирующий метод **@polynom/plot.m** имеет вид:

```
function    plot(p)
% POLYNOM/PLOT
% Функция PLOT(p) строит график полиномиального
% объекта p.
        r = max(abs(roots(p)));
        x = (-1.1:.01:1.1)*r;
        y = polyval(p,x);
        plot(x,y);
```

```
title(char(p))
grid on
```

Наконец, рассмотрим метод **@polynom/diff.m,** который позволяет дифференцировать полиномы путем умножения коэффициентов на соответствующие степени переменной:

```
function q = diff(p)
% POLYNOM/DIFF
% Функция DIFF(p) - производная полинома p.
c = p.c;
d = length(c) - 1; % степень полинома
q = polynom(p.c(1:d).*(d:-1:1));
```

Функция **methods('polynom')** или в форме команды **methods polynom** выводит на экран все методы для данного класса в следующем виде:

```
methods polynom
 Methods for class polynom:
 char display minus plot polynom roots
 diff double mtimes plus polyval
```

Большинство этих методов будет вызвано при выполнении операторов

```
p = polynom([1 0 -2 -5]);
 plot(diff(p*p + 10*p + 20*x) - 20)
```

8.4 Иерархия объектов. Индексация объектов

Правило старшинства устанавливает иерархию объектов в системе MATLAB. Это позволяет управлять последовательностью обработки выражений, составленных из объектов. В системе MATLAB принято, что объекты имеют один приоритет и при выполнении выражения вызывается метод, ассоциированный с крайним левым объектом. Если установлено соотношение старшинства, то вызывается метод для класса с самым высоким приоритетом. Для установления иерархии объектов служат функции конструктора **inferiorto** и **superiorto**.

Функция **superiorto('class_A')** устанавливает более высокий приоритет объектов других классов по отношению к объектам класса, указанного в качестве аргумента.

Допустим, что мы решили добавить класс объектов **rational,** и тогда ожидается появление смешанных выражений, которые включают полиномы и рациональные функции. Причем класс **rational** должен находиться выше в иерархии классов, чем класс **polynom**. Это может быть реализовано указанием в конструкторе **@rational/rational.m** оператора **superiorto('polynom').** При этом в методы класса **polynom** никаких изменений не вносится.

Тогда выражения типа **p + r, r + p, p*r и r*p**, включающие полином **p** и рациональную функцию **r** будут использовать методы, определенные в конструкторе.

Функция **inferiorto('class_A')** устанавливает более низкий приоритет объектов других классов по отношению к объектам класса, указанного в качестве аргумента; иначе говоря, объекты класса **class_A** имеют наивысший приоритет.

Индексация объектов

Общее правило состоит в том, что индексации объектов аналогична индексации структур.

Индексная ссылка. Использование индекса или указателя поля в правой части оператора присваивания называется индексной ссылкой. В этих случаях вызывается метод **subsref**, реализованный в виде встроенной функции. Соответствующие выражения могут иметь следующий вид **A(I), A{I}, A.field**. Каждый из них приводит к вызову метода **subsref** в форме

B = subsref(A, S)

Второй аргумент **S** является структурой с двумя полями. Поле **S.type** - строка, содержащая символы **'( )', '{ }'** или **'.'**, которые определяют тип индекса. Круглые скобки соответствуют числовому массиву; фигурные - массиву ячеек; точка - структуре. Поле **S.subs** - массив ячеек или строка, содержащая фактические индексы. Двоеточие, используемое как индекс, соответствует строковой переменной **':'**. Например, выражение **A(1:2, :)** вызывает метод **subsref(A, S)**, где **S** - структура размера **1x1** вида

S.type = '( )'
S.subs = {1:2, ':'}

Выражение **A{1:2}** вызывает метод **subsref(A, S),** где **S** - структура вида

S.type ='{ }'
S.subs = {1:2}.

Выражение **A.field** вызывает метод **subsref(A, S),** где **S** - структура вида

S.type = '.'
S.subs = 'field'.

Эти простые обращения могут быть объединены в более сложные индексные выражения. В этом случае **length(S)** определяет количество уровней индексации. Например, выражение **A(1, 2).name(3:4)** вызывает метод subsref(A, S), где **S** - структура размера **3x1** со следующими значениями полей

S(1).type = '( )'	**S(2).type = '.'**	**S(3).type = '( )'**
S(1).subs = '{1, 2}'	**S(2).subs = 'name'**	**S(3).subs = '{3:4}'**

Индексное присваивание. Использование индекса или указателя поля в левой части оператора присваивания называется индексным присваиванием. В этих случаях MATLAB вызывает метод **subsasgn**, реализованный в виде встроенной функции. Соответствующие выражения могут иметь следующий вид

A(I) = B
A{I} = B
A.field = B

Каждый из них приводит к вызову метода **subsasgn** в форме

A = subsasgn(A, S, B)

Поля структуры **S** аналогичны полям структуры в случае индексной ссылки.

Функция **subsasgn** различает присваивание вида **A(i) = [ ]** и **A(i) = B**, где **B** - пустой массив.

Если правая часть - символ пустого массива **[ ]**, а не переменная, то третий аргумент входа функции **subsasgn** является строкой **'[ ]'**. Следовательно, при разработке собственного метода типа **subsasgn** следует различать эти два случая.

8.5 Наследование

Процедура, когда объекты одного класса приобретают свойства объектов другого класса или классов, называется процессом наследования. Если некоторый объект (дочерний) наследует свойства другого (родителя), дочерний объект включает все поля родительского объекта и может вызывать соответствующие методы.

Наследование - главное свойство объектно-ориентированного программирования. Оно позволяет многократно применять программы, разработанные для родительских объектов, к дочерним. Методы родительского класса имеют доступ только к тем полям, которые унаследованы от родительского класса, но не к полям дочернего класса.

Существует два вида наследования:

- простое наследование, когда дочерний объект наследует характеристики от одного родительского класса;
- множественное наследование, когда дочерний объект наследует характеристики более чем одного родительского класса.

Простое наследование. В случае простого наследования класс, который наследует атрибуты другого класса и добавляет собственные атрибуты. Наследование подразумевает, что объекты, принадлежащие к дочернему классу, имеют те же поля, что и объекты родительского класса, и как правило, дополнительные собственные поля. Именно поэтому методы родительского класса могут применяться к объектам дочернего класса. Методы дочернего класса однако не могут применяться к объектам родительского класса.

Функция конструктора для класса, который наследует поведение другого класса, имеет две специальных характеристики:

- она обычно вызывает функцию конструктора родительского класса, чтобы создать “наследованные” поля;
- вызов функции такого класса несколько отличается от стандартного, поскольку учитывает как новый, так и родительский класс.

Примеры простого наследования представлены объектами ППП **Control System Toolbox**, используемыми для решения задач анализа линейных стационарных систем (**Linear, Time-Invariant systems - LTI)**. Родительский класс так и называется **lti**. Имеются три дочерних класса, или подкласса, которые соответствуют трем различным представлениям **LTI**-систем:.

- **tf**-- Передаточная функция (**Transfer function**)
- **zpk**-- Нули-полюса-коэффициент передачи (**Zero, pole, gain**)
- **ss**-- Пространство состояний (**State space**)

Объект **lti** включает информацию, которая не зависит от частного вида системы (непрерывная или дискретная), а также от имен входов и выходов. Дочерние объекты зависят от модели представления. Объект класса **tf** характеризуется векторами коэффициентов числителя и знаменателя рациональной передаточной функции. Объект

класса **zpk** характеризуется векторами, содержащими нули, полюса и коэффициент передачи системы. Объект класса **ss** определяется четверкой матриц, дающих описание динамической системы в пространстве состояний.

Оператор создания объекта в классе **lti**

L = lti(1, 1)

создает скелет **LTI**-объекта с нулевым тактом дискретности и пустыми именами входов и выходов.

Оператор вида

T = tf(1,[1 0 -2 -5])

создает объект в классе **tf,** который представляет собой непрерывную передаточную функцию

Transfer function:

1

s^3 - 2 s - 5

Более полно объект **T** характеризуется четырьмя полями:

- **T.num** -- Поле вектора числителя, 1
- **T.den** -- Поле вектора знаменателя, [1 0 -2 -5]
- **T.variable** -- Поле переменной **'s'**
- **T.lti** -- Поле, наследованное от **lti**-класса

Для объекта **tf** поле **lti** наследовано от его родителя **LTI**-объекта. В данном случае оно совпадает со скелетом **L**.

Если для ППП **Control System Toolbox** переопределить функции **set** и **get,** то их можно использовать для анализа свойств объекта.

Пример:

```
get(T)
    num = {[0 0 0 1]}
    den = {[1 0 -2 -5]}
    Variable = 's'
    Ts = 0
    Td = 0
   InputName = {''}
    OutputName = {''}
    Notes = {}
    UserData = []
```

Диаграмма на рисунке 8.2 иллюстрирует соотношения наследования

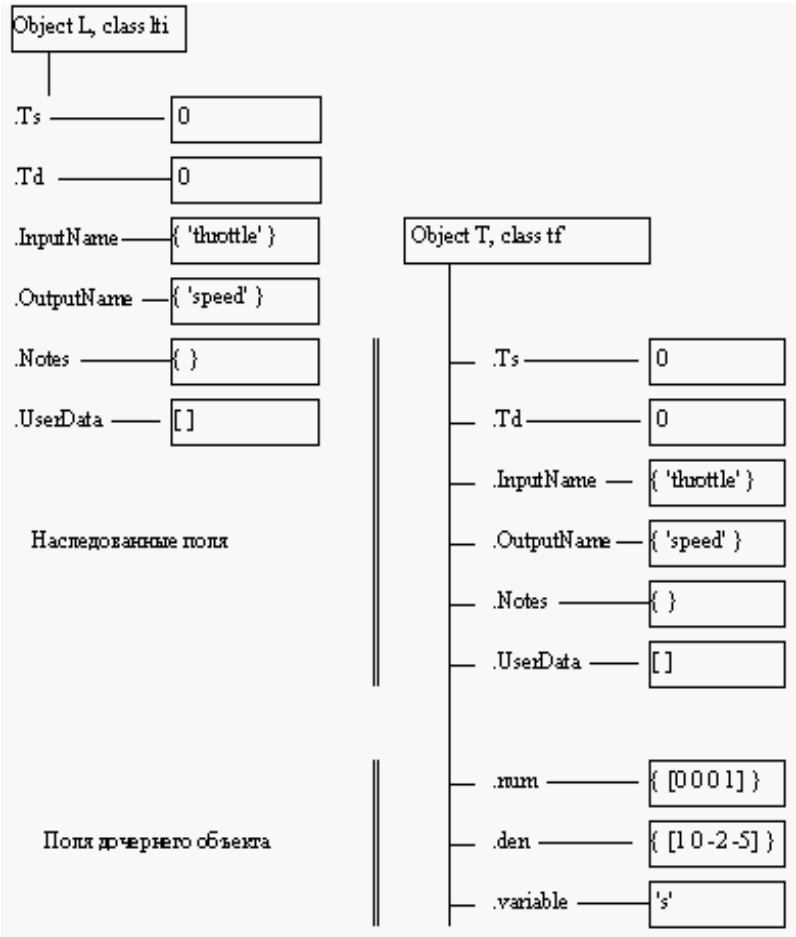


Рисунок 8.2

Функции **S = ss(T)** и **Z = zpk(T)** преобразовывают представление в виде передаточной функции в другие представления, сохраняя общее поле **Iti**-объекта для всех трех представлений:

S=ss(T)

a =

x1 x2 x3

```
x1       0          1.00000    1.25000
x2       2.00000    0          0
x3       0          2.00000    0

b =
                    u1
          x1        0.50000
          x2        0
          x3        0

c =
                x1    x2    x3
       y1       0     0     0.50000

d =
                    u1
          y1        0

Continuous-time system.
      get(S)
         a = [3x3 double]
         b = [3x1 double]
         c = [0 0 0.5]
         d = 0
         e = [ ]
         StateName = {3x1 cell}
         Ts = 0
         Td = 0
         InputName = {''}
         OutputName = {''}
         Notes = {}
         UserData = [ ]
       Z = zpk(T)
       Zero/pole/gain:
```

```
                  1
 -------------------------------
 (s-2.095) (s^2 + 2.095s + 2.387)
get(Z)
    z = {[0x1 double]}
    p = {1x1 cell}
    k = 1
    Variable = 's'
    Ts = 0
    Td = 0
    InputName = {''}
    OutputName = {''}
    Notes = {}
    UserData = [ ]
```

Механизм наследования реализуется в функциях конструктора дочернего класса. Например, конструктор класса **tf** включает оператор

```
L = lti(Ny, Nu, Ts)
```

для создания **lti**-объекта с соответствующими параметрами. Тогда **tf**-объект можно создать, используя встроенный оператор **class**

```
sys = class(sys, 'tf', L)
```

Такое использование оператора **class** с тремя аргументами позволяет присвоить объекту соответствующую метку класса и указать, что наследуется от родительского объекта.

Наследование может порождать более одного поколения, то есть дочерний объект может содержать поля как родительских объектов, так и объектов более старших поколений. В этом случае родительский объект может вызывать прародительские методы, а дочерний - как родительские, так и прародительские .

Множественное наследование. В случае множественного наследования класс наследует атрибуты от более, чем одного родительского класса. Дочерний объект наследует поля, как поля родительских классов, так и поля собственной структуры.

Множественное наследование может объединять более одного поколения. Например, каждый из родительских объектов может наследовать поля от множественных прародительских объектов. Множественное наследование реализовано в конструкторах путем вызова функции class с более, чем тремя аргументами:

```
obj = class(structure, 'class_name', parent1, parent2,...)
```

Можно объединять любое количество аргументов родительского класса в соответствии со списком входных аргументов.

Схема на рисунках 8.3 и 8.4 показывает, как гипотетический объект класса **eco** мог бы наследовать поля от двух родительских классов **weather** и **plant**.

Множественные родительские классы могут присоединять методы с одинаковыми названиями. В этом случае MATLAB использует метод, связанный с родителем, который появляется первым при обращении к функции класса в конструкторе.

Агрегирование объектов. В дополнение к наследованию система MATLAB поддерживает соединение частей в целое, или агрегирование. То есть один объект может включать другой объект в качестве одного из полей структуры. Объект класса **rational** может использовать два объекта из класса **polynom**, например, для задания числителя и знаменателя передаточной функции. Поскольку доступ к полям структуры возможен только изнутри метода, то вызов метода для включенного объекта возможен только изнутри метода для внешнего объекта.

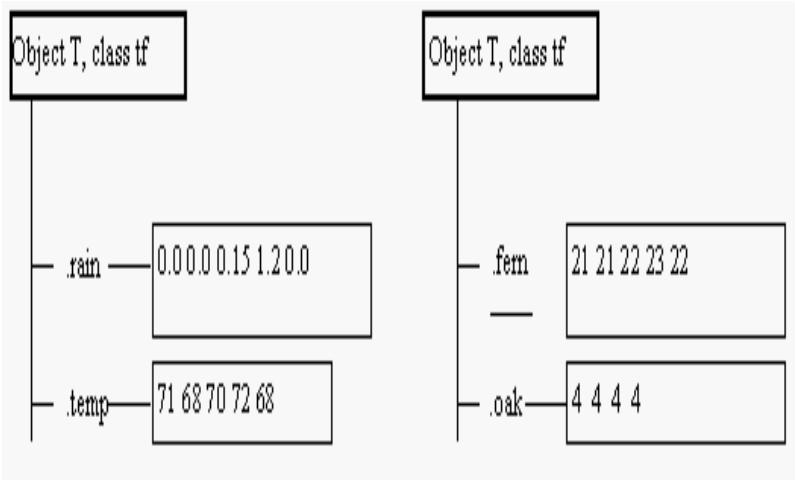


Рисунок 8.3

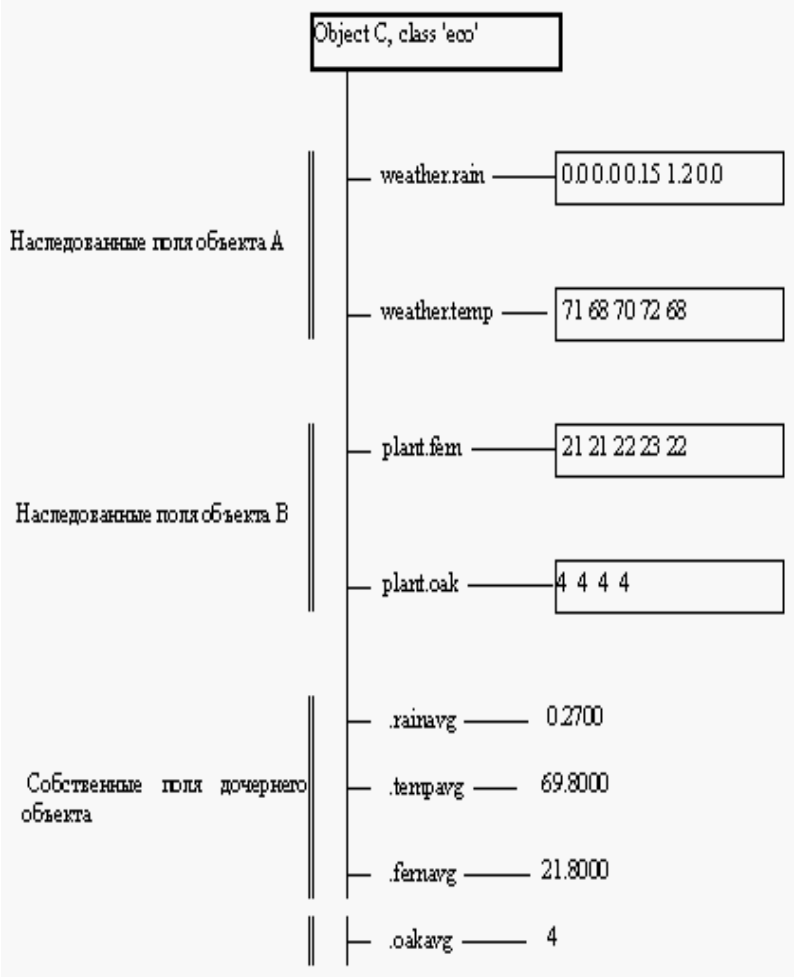


Рисунок 8.4

8.6 Описание функций и команд

CLASS **Определить класс объекта или создать объект**

Синтаксис:

str = class('<имя_объекта>')

obj = class(S, '<имя_класса>')

obj = class(S, '<имя_класса>', <родитель1>, <родитель2> …)

Описание. Функция **str = class('<имя_объекта>')** возвращает строку, содержащую имя класса, соответствующего следующей таблице

Имя класса	***Класс объектов***
Double	Многомерные массивы чисел в арифметике с плавающей точкой в формате удвоенной точности.
Sparse	Двумерные действительные или комплексные разреженные матрицы
Char	Массивы символов
Struct	Массивы записей (структура)
Cell	Массивы ячеек
'<имя_класса>'	Класс, определяемый пользователем

Функция **obj = class(S, '<имя_класса>')** создает объект класса с указанным именем, используя структуру S в качестве шаблона. Это относится только к М-функциям с именем **имя_класса.m**, размещенным в каталоге **@имя_класса**.

Функция **obj = class(S, '<имя_класса>', <родитель1>, <родитель2> …)** создает объект класса с указанным именем, используя структуру **S** в качестве шаблона, а также гарантирует, что вновь создаваемый объект наследует методы и поля родительских объектов, указанных в качестве аргументов.

Сопутствующие операторы: **INERIORTO, ISA, SUPERIORTO**.

ISA **Определить принадлежность объекта к данному классу**

Синтаксис:

K = class(obj, '<имя_класса>')

Описание. Функция **K = class(obj, '<имя_класса>')** возвращает логическое **TRUE** (1), если объект принадлежит данному классу, и логическое **FALSE** (0) - в противном случае.

Аргумент **'<имя_класса>'** - это либо имя класса, определенного пользователем, либо имя одного из предопределенных классов системы MATLAB:

Имя класса	***Класс объектов***
Double	Многомерные массивы чисел в арифметике с плавающей точкой в формате удвоенной точности.
Sparse	Двумерные действительные или комплексные разреженные матрицы
Char	Массивы символов
Struct	Массивы записей (структура)
Cell	Массивы ячеек

Пример. Функция

isa(rand(3, 4), 'double')

истинна и возвращает значение

ans = 1

Сопутствующие операторы: **CLASS**.

ISOBJECT **Выявление объекта некоторого класса**

Синтаксис:

k = isobject(A)

Описание. Функция **k = isobject(A)** возвращает логическое **TRUE** (1), если **A** - объект некоторого класса, и логическое **FALSE** (0) - в противном случае.

Сопутствующие операторы: операторы группы **IS***.

METHODS **Вывести на терминал список методов для данного класса**

Синтаксис:

methods <имя_класса>

s = methods('<имя_класса>')

Описание. Команда **methods <имя_класса>** выводит на терминал список методов для данного класса.

Функция **s = methods('<имя_класса>')** возвращает массив ячеек, содержащих имена методов в виде строковых величин.

Пример. Команда **methods polynom** выводит на экран следующий список методов для класса **polynom**:

```
methods polynom
 Methods for class polynom:
 Методы для класса polynom:
 char display minus plot polynom roots
 diff double mtimes plus polyval
```

Функция **s = methods('polynom')** выводит на экран следующий список методов для класса **polynom** в виде массива ячеек символов:

```
s = methods( 'polynom')
 s =
    'polynom'
    'mtimes'
    'plus'
    'plot'
    'display'
    'diff'
    'char'
    'roots'
    'minus'
    'double'
    'polyval'
```

Сопутствующие операторы: **HELP, WHAT, WHICH.**

INFERIORTO Отношение низшего класса

Синтаксис:

inferiorto('<имя_класса1>', '<имя_класса2>', …)

Описание. Команда **inferiorto('<имя_класса1>', '<имя_класса2>', …),** вызванная внутри конструктора класса (например, **myclass.m),** определяет, что метод **myclass.m** не должен вызываться, если функция вызвана с объектом класса **myclass** и одним или более объектов классов **'<имя_класса1>', '<имя_класса2>', ….**

Пояснение. Допустим, что **A** - это объект класса **'class_a', B** - это объект класса **'class_b'** и **C** - это объект класса **'class_c'**. Допустим также, что конструктор **class_c.m** содержит утверждение

inferiorto('class_a'), что означает низший приоритет класса **'class_c'** по отношению к классу **'class_a'**. Тогда при вызове функций **e = fun(a, c)** или **e = fun(c, a)** вызывается функция **class_a/fun.**

Если функция вызывает 2 объекта, отношение классов которых не определено, то оба объекта имеют одинаковый приоритет и используется метод, относящийся к первому объекту в списке аргументов вызываемой функции. То есть, вызов **fun(b, c)** использует метод **class_b/fun**, а вызов **fun(c, b)** использует метод **class_c/fun.**

Сопутствующие операторы: **SUPERIORTO.**

SUPERIORTO **Отношение высшего класса**

Синтаксис:

superiorto('<имя_класса1>', '<имя_класса2>', …)

Описание. Команда **superiorto('<имя_класса1>', '<имя_класса2>', …),** вызванная внутри конструктора класса (например, **myclass.m**), определяет, что метод **myclass.m** должен быть вызван, если функция вызвана с объектом класса **myclass** и одним или более объектов классов **'<имя_класса1>', '<имя_класса2>', ….**
Пояснение. Допустим, что **A** - это объект класса **'class_a', B** - это объект класса **'class_b'** и **C** - это объект класса **'class_c'**. Допустим также, что конструктор **class_c.m** содержит утверждение **superiorto('class_a'),** что означает высший приоритет класса **'class_c'** по отношению к классу **'class_a'**. Тогда при вызове функций **e = fun(a, c)** или **e = fun(c, a)** вызывается функция **class_c/fun**.

Если функция вызывает 2 объекта, отношение классов которых не определено, то оба объекта имеют одинаковый приоритет и используется метод, относящийся к первому объекту в списке аргументов вызываемой функции. То есть, вызов **fun(b, c)** использует метод **class_b/fun**, а вызов **fun(c, b)** - метод **class_c/fun**.

Сопутствующие операторы: **INFERIORTO**.

Терёхин
Валерий Владимирович

Моделирование в системе MATLAB

Учебное пособие

Редактор……