ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
"ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ"

Н. В. Курбатова

“Компьютерное моделирование в Матлаб”

(семестровый курс лекций)

Ростов-на-Дону
2008 г.

Курбатова Н. В. Компьютерное моделирование в Матлаб. Семестровый курс лекций. Ростов-на-Дону, 2008. 114 с.

Курс лекций “Компьютерное моделирование в Матлаб” предусматривает освоение возможностей пакета MatLab, объектов языка, дескрипторной графики, приемов эффективного программирования, создания интерфейсов пользователя. На их основе рассмотрены алгоритмы создания пакетов программ и независимых приложений для решения серии проблемных задач. Предлагаются схемы и примеры использования инструментариев “статистик”, “нейронные сети”, “решение уравнений в частных производных”, которые являются востребованными при решении научно-исследовательских задач.

Оглавление

1 Введение

Пакет MatLab – не только пакет, предназначенный для учебного процесса, инструмент научного исследования, но и пакет решения коммерческих задач, широко используемый в мировой практике. Уровень использование пакета, инструментария модулей, охватывающих множество предметных областей, полностью определяется подготовленностью пользователя. Размах его заключается в пределах от использования пакета в качестве банального калькулятора до мощного вычислительного ресурса, способного решать оптимизационные, статистические, модельные задачи и т.п. В настоящий курс включены в том числе схемы использования модулей Statistics, PDETool, NeuralNetworks.

Язык программирования $MATLAB$ ($Matrix\ Laboratory$) разрабатывался Кливом Моулером ($Cleve\ Moler$) в конце 1970-х годов (Нью-Мексико). Основной его целью было получить максимально простой язык, "заточенный" на матричные вычисления. Традиционным языком инженерных и научных исследований того времени, снабженным библиотеками, в том числе $Linpack$ – библиотекой пакетов линейной алгебры, был $FORTRAN$. На его основе и появилась упрощенная модификация, приведшая к удобному языку матричных вычислений, так необходимому при численной дискретизации широкого класса задач. Основной его особенностью была направленность на контейнерные структуры с ориентацией на комплексную арифметику и двойную точность вычислений. Дальнейшее развитие языка привело к синтезу возможностей языков $C, C++$ и их библиотек, развитию модульной структуры, каждый модуль которой ориентировался на конкретную предметную научную или инженерную область. Параллельно осуществлялось развитие дружественного интерфейса пакета и возможностей динамического использования справочной системы, когда коды, предлагаемые в качестве примеров доступны к реализа-

ции не только в специальных демонстрационных примерах. Стоит также отметить, что развитый пакет символьных вычислений, является ядром "Maple" и "заимствован"у фирмы на коммерческой основе.

Глава 1

Структура MatLab–интерфейса

Первый раздел объединяет материал, который помогает пользователю MatLab легко ориентироваться в многомодульной архитектуре пакета, освоить функции, способствующие интерактивно и эффективно осуществлять диалог со средой программирования и добиваться заметных успехов в режиме интерпретатора.

1 Лекция 1. Структурирование пакета MatLab

Язык $MATLAB$ является высокоуровневым интерпретируемым языком программирования, любой элемент которого является массивом, он создан в следствие модификация языка программирования $Fortran$. В $MATLAB$ поддерживаются объектно–ориентированные принципы и программируются интерфейсы к программам, написанным на других языках программирования.

Основной особенностью языка являются широкие возможности работы с матрицами, которые предполагают матричное мышление при написании кодов программ, для этого поддерживаются векторные операции и предусмотрена функциональная векторизация операций.

Пакет ориентирован на комплексную арифметику; в следствие этого для хранения любого объекта резервируются вещественная и мнимая части, да-

же если в соответствии с логикой создания объекта, например строк, мнимая часть отсутствует, в таком случае она является нулевой. Для выполнения аналитических вычислений в *MatLab* используется ядро символьных вычислений *Maple*. В пакете предусмотрена возможность совместного использования библиотек языков высокого уровня, а также создания независимых приложений.

Популярность пакета обусловила необходимость его адаптации к различным платформам, в том числе, *Windows* (95, 98, 2000, *XP*), *Sun Solaris N*, *Macintosh*, *Linnux*.

Удобное функционирование пакета *MatLab* обеспечивает оконный интерфейс, (рис. 1.1), о котором пойдет речь далее.

- В окне *LaunchPad* представлено содержание загружаемых пакетов (*Toolboxes*), предназначенных для исследований в различных предметных областях
- *Help* – развитая справочная система
- *Command Window* – окно команд предназначено для выполнения программных инструкций в интерактивном режиме, а также для осуществления обратной связи программист $\leftrightarrows$ система (при выполнении скриптов, процедур, функционировании моделей)
- *Workspace* – окно, содержащее информацию о переменных среды (рабочего пространства)

Назначение модулей (*Toolbox*)

Объединяет функции, предназначенные для решения задач, предусмотренные выбираемой предметной областью. Так в окне *LaunchPad* отображаются загружаемые пакеты:

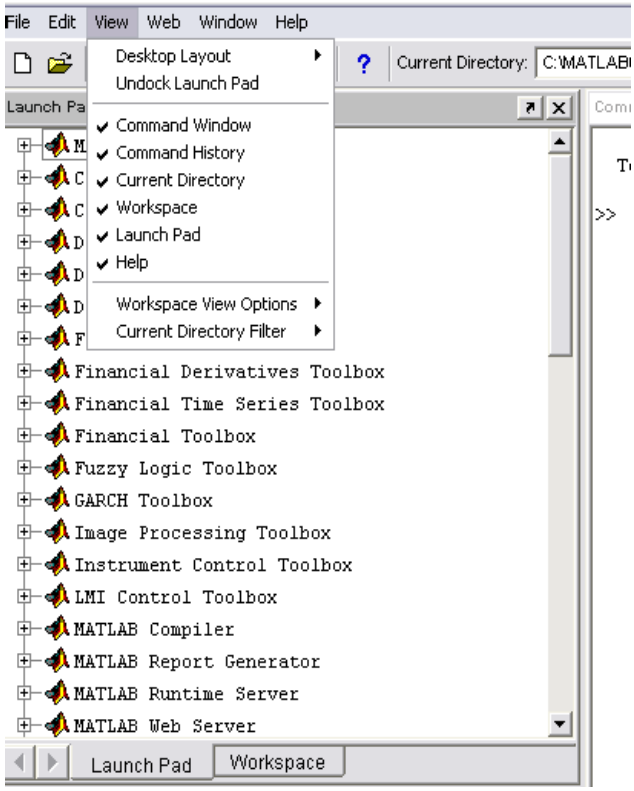


Рис. 1.1: Оконный интерфейс. Выбор доступа

MatLab – раздел, в котором описываются основные структуры среды программирования, языковые конструкции, управляющие повторяющимися операциями, потоками, специальные возможности и функции

Financial – пакет предназначен для анализа финансовых процессов, состояния финансовых портфелей, прогноза, анализа временных рядов

PDE – модуль решения дифференциальных уравнений в частных производных; в том числе с помощью удобного интерфейса, который позволяет строить геометрию области, задавать граничные условия, выбирать тип и параметры уравнений, триангулировать, а также получать векторы искомых полей и их характеристик

Statistics с его помощью проводится не только первичный статистический анализ основных статистик, но также осуществляется (фиттинг) подбор подходящих распределений, использования критериальных стати-

стик для оценки в том числе качества выбранных распределений или иных характеристик центральной симметрии, а также проведения дисперсионного анализа

WaveLet (WL) Преобразования, являющие модификацией преобразований Фурье, в котором в качестве базисных функции используются вейвлеты – функции "маленькой" волны (всплески). С их помощью обрабатываются сигналы, изображения; WL также применяются для аппроксимации функций, дискретизации дифференциальных уравнений и их решения ...

Spline модуль, ориентированный на интерполяцию с использованием возможностей, предлагаемых в сплайнах широких классов функций

Symbolic Символьные вычисления, основанные на функциях ядра Maple, позволяющие эффективно проводить аналитические преобразования

Simulink модуль, с помощью функций которого строятся модели и осуществляется интерактивная возможность регулирования параметрами построенных моделей; реализация элементов управления. Построение моделей реализуется с помощью удобного интерфейс. Результирующая модель генерируется системой в виде скрипта.

☼ **.** *Заметим, что предлагаемый список далеко не исчерпывает содержание пакета. Следует отметить, что разработчики пакета расширяют возможности MatLab, в том числе, покупая готовые модули для новых предметных областей.*

Такая стратегия активно стимулирует вовлечение программистов всего мирового сообщества в модификацию и совершенствование этого программного продукта.

Help – справочная система

На рис.1.2 предлагается структура справочного окна, назначение функций которого перечислено ниже:

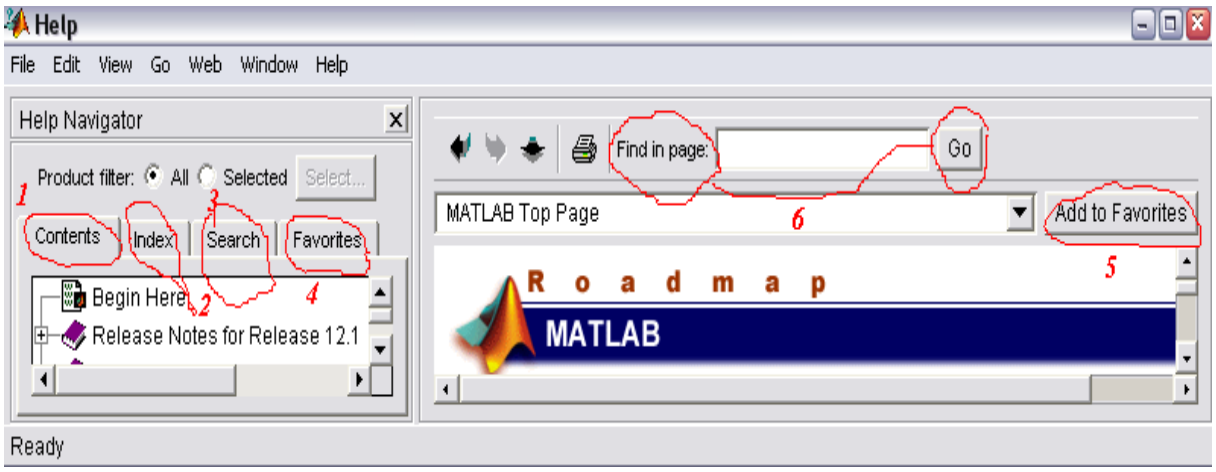


Рис. 1.2: Справочное окно

1. Представлены модули (содержание)
2. Осуществляется поиск по справке в алфавитном порядке
3. Поиск в заданном файле
4. Выбор и $\leftrightsquigarrow$
5. добавление "любимой" справки
6. Контекстный поиск (хранятся последние темы)

Справочная система является "дружественной" по отношению к пользователю и содержит значительное количество демонстрационных файлов. Предусматривается последовательное выполнение инструкций, их отображение в командном окне, а также результата выполнения таких инструкций. Содержание демо–файлов предоставляется и отображается в соответствии с рисунком (1.3)

Контекстный поиск также может быть выполнен из командной строки с помощью команды:

$$> lookfor\ keywords$$

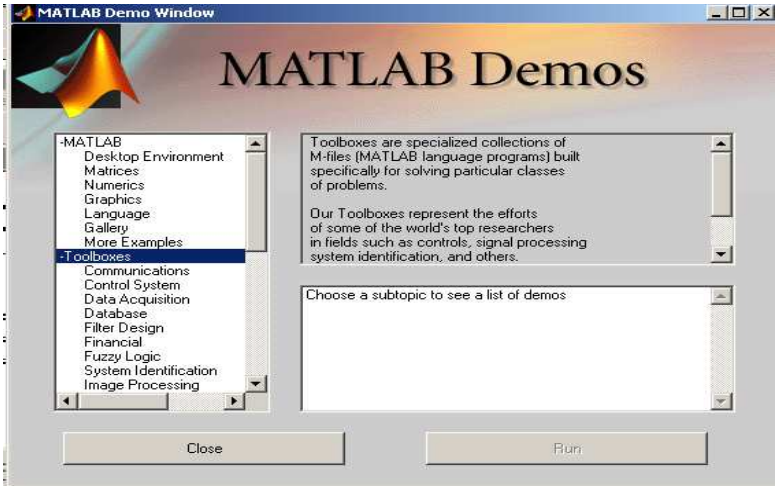


Рис. 1.3: Демонстрационные файлы

а тематическая справка – по команде:

$$> help\ helpgroup \text{ или } \quad > help\ namefunc$$

далее на рисунке представлена таблица, которая в левой колонке содержит "имена" *helpgroup*:

Группы функций	
general	команды общего назначения
ops	операторы и специальные символы
lang	конструкции языка
elmat	элементарные матрицы и действия с матрицами
specmat	специальные матрицы
elfun	элементарные математические функции
specfun	специальные математические функции
matfun	матричные функции - численные методы линейной алгебры
datafun	обработка данных и преобразование Фурье
polyfun	полиномиальные функции и интерполяция
sparfun	работа с разреженными матрицами
plotxy	двумерная графика
plotxyz	трехмерная графика
graphics	графические функции общего назначения
color	управление цветом и освещением
sounds	функции работы со звуком
strfun	функции работы со строковыми переменными
iofun	функции ввода/вывода низкого уровня
demos	демонстрации и примеры

Рис. 1.4: Группы функций *MatLab*

↺ . *Последовательность команд, инструкций, предлагаемых в справоч-*

ных текстах, можно выполнить непосредственно из справки; для этого их следует выделить курсором, щелкнуть правой кнопкой мыши, и в выплывающем локальном меню выбрать функцию execute–выполнить.

2 Лекция 2. Оконный интерфейс и функционирование MatLab

Окно команд. Root

На языке объектно-ориентированного программирования окно команд–*Root* является "родителем" для всех графических объектов, с нулевым описателем–дескриптором. По своей природе он является структурой. Все свойства-поля структуры с визуализацией всех возможных свойств и по умолчанию можно получить по команде (или текущего значения выбранного $anyproperty$ свойства)

$$> set(0), \quad (\quad > get(0, anyproperty) \quad)$$

однако для удобства непосвященного в нюансы свойств объектов пользователя существует набор команд высокого уровня, с помощью которых меняются свойства командного окна. К их числу относятся:

clc – очистка экрана

echo $> echo\ [name\ m-file]$ – команда (и свойство поля структуры $root$), on – отображаются команды скрипта с именем $name\ m-file$ и результат их выполнения в командном окне, off – нет

diary создается текстовый файл всех команд сессии с именем $diary.txt$

format по следующей команде вы получите все предусмотренные форматы (отображения данных, тогда как все операции производятся с двойной точностью):

$$> set(0,'format')$$

в результате получаются

$$[short|long|shortE|longE|shortG|longG|hex|bank| + |rational]$$

данные с фиксированной точкой; для короткой формы - четыре цифры после запятой; с плавающей точкой, шестнадцатиричные, финансовые форматы, рациональная форма представления данных – очевидно определяются соответствия.

$\circlearrowleft$. *Команды, выполняемые в командном окне, помещаются в стек, выбрать их в строку ввода можно с помощью клавиш* $\uparrow$ *и* $\downarrow$

Командное окно, помимо того, что является удобным интерфейсом, еще и мощный функциональный "калькулятор" – интерпретатор. Оно состоит из *строки ввода* – редактируемой последовательности команд, отделяемой запятой (не превышает 256 символов, перенос строки ввода осуществляется троеточием ...); и *строки вывода* – нередактируемой части, результата вычислений, который подавляется точкой с запятой (;). Если результат выполнения операции не присваивается идентификатору то присваивается системной переменной *ans*. На рис. 1.5 представлено командное окно.

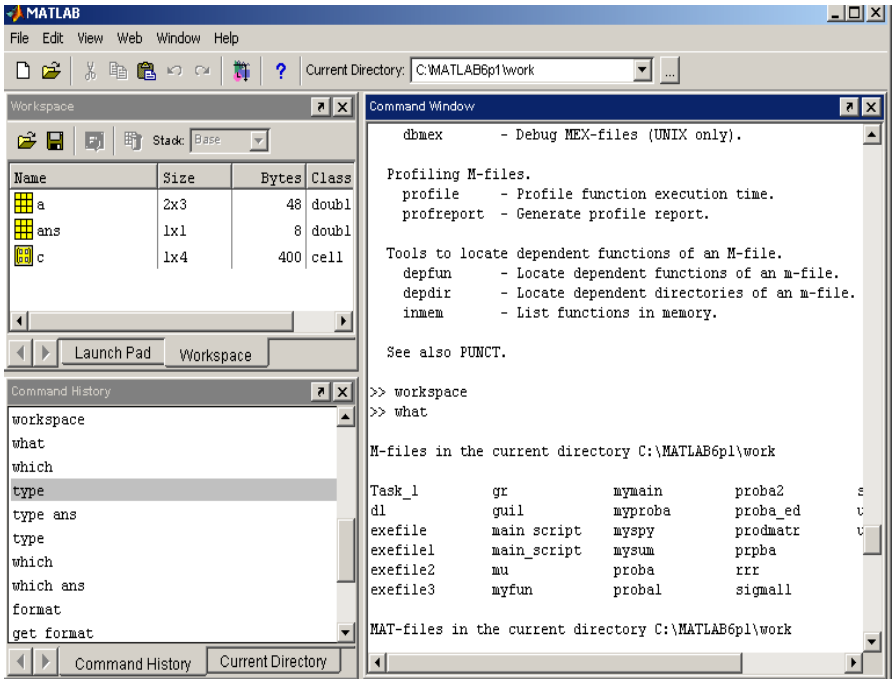


Рис. 1.5: Командное окно и информация о переменных среды

Workspace

Переменная MatLab – идентификатор, неограниченной длины (с различением регистра), начинается с буквы, не содержит знаков операций и точки (только для указания поля структуры); при этом система различает 31 знак имени.

Следующие идентификаторы резервируют системные переменные: inf – результат деления на ноль, $i,\ j$ – мнимые единицы, $realmin\ -(2^{-1022})$; $realmax\ -(2^{1023})$; pi – π; eps – малое, по "чувствительности" совпадает с машинным нулем; NaN $(not\ anumber)$ – описывает неопределенность.

Переменные среды пакета $MatLab$ имеют нестрогую типизацию. Тип переменных идентифицируется системой в момент создания, все переменные являются глобальными. Это обстоятельство является благом и злом, ибо программист должен во избежание ошибочных ситуаций осуществлять дополнительный контроль типов.

Workspace – область распределения памяти переменных сессии, она содержит переменные и информацию о типе, размере; в этом окне существует возможность непосредственного редактирования небольших массивов (двойным щелчком по идентификатору).

Редактирование и распознование свойств переменных среды осуществляется и с помощью специальных команд, к их числу относятся следующие команды

save [nf [nv1 nv2 nv3]] сохранить переменные (в квадратных скобках указываются необязательные параметры); без параметров — все переменные записываются в двоичном файле; $load$ – аналогично, загрузить; $matlab.mat$; nf – имя файла пользователя; $nv1\ \ldots$ – имя переменных;

clear [nv1 nv2 nv3] – удаление всех или части переменных;

$> who,\ > whos$ по этим командам отображаются информация о переменных рабочего пространства и подробная информация об их свойствах (типе, размере, и т.п.);

Все упомянутые предшествующие (и в первой лекции) команды относятся к командам **общего назначения** *general*.

$\circlearrowleft$ **.** *В Help Index набранное is∗ в окне поиска влечет отображение всех функций, позволяющих выявить тип анализируемых переменных.*

Основные операции $MatLab$

Язык программирования среды $MatLab$ унаследовал все операции, конструирующие выражения в $Fortran$, к их числу относятся **арифметические операции** ($+\ -\ *\ ^\wedge\ \ /\ \backslash$) – правое и левое матричное деление (для прямоугольных матриц выполняется деление в смысле МНК), а операции (.* .^) – применяются покоординатно.

Перечень всех операций доступен из справочной системы по команде

$$> help\ ops$$

результирующая таблица содержит функции, сведенные в логические группы, а именно,

- Арифметические операции
- Операции отношения (eq (==), $ne(\sim=)$, lt (<), gt (>), le (<=), ge (>=)) Результат выполнения - массив совпадающей размерности с объектами отношения, состоящий из нулей, в позициях, где отношения ложны, и из единиц – истины
- Логические операции $and(\&)$, $or(|)$, $not(\sim)$, xor, any, all все традиционно, за исключением последних двух команд; так результат выполнения all – вектор-строка, единица в котором означает "истину т.е. что

в текущем столбце все элементы ненулевые. которых покоординатно проверяется на

- Операции над множествами: $union$, $unique$, $intersect$, $setdiff$, $setxor$, $ismember$ весьма полезная последняя команда, которая является новой; она проверяет принадлежит ли объект указанному множеству.

Специальные пунктуационные обозначения

- Исключительно важным является двоеточие (:), с его помощью задаются диапазоны элементов; так, например, $r = [1 : 6]$ – вектор-строка (r' – вектор столбец) шести натуральных чисел;
- Круглые скобки указывают номера элементов (индексы) ($r(2 : 4)$ 2 3 4 и отделяют аргументы функций
- Фигурные скобки {} используются для массивов ячеек при выборе индекса и его создании
- @ – задает функцию
- Точка – десятичная точка или разделитель полей структуры
- ... – символ разделения строки выражения
- , – отделяет один оператор от другого, а также осуществляет конкатенацию вдоль строк
- ; – отделяет один оператор от другого и подавляет вывод результата, а также осуществляет конкатенацию вдоль столбцов
- % – знак последующего коментария
- операция $'$ – транспонирования

Приведем пример использования элементарных арифметических операций, из которого следует, что программист при реализации матричных вычислений избавлен от применения циклов и реализации методов решения систем в $MatLab$.

```
>> A=rand(3)
 A =
     0.4447    0.9218    0.4057
     0.6154    0.7382    0.9355
     0.7919    0.1763    0.9169
>> x=[1 1 1]
x =
     1     1     1
 >> x=x'
x =
     1
     1
     1
:>> b=A*x
b =
    1.7722
    2.2891
    1.8851
 >> xx=A\b
xx =
     1.0000
     1.0000
     1.0000
```

Здесь случайным образом сгенерирована матрица, выбран единичный вектор-строка, транспонирован, чтобы выполнялись правила матричного умножения, получен результат их произведения. А далее с помощью левого деления решена система (методом Гаусса).

3 Лекция 3. Дескрипторная графика. Функциональный дуализм

В пакете реализованы объектно-ориентированные принципы функционирования всех типов графических объектов. Описателями графических объектов являются дескрипторы, которые генерируются системой и "кодируют" все их свойства. Пользователь также может назначить "вручную"(handle) имена-идентификатры графическим объектам. Иерархия, отношения "родители-потомки" с сохранением принципов наследования представлены на следующей схеме: В текущей лекции остановимся подроб-

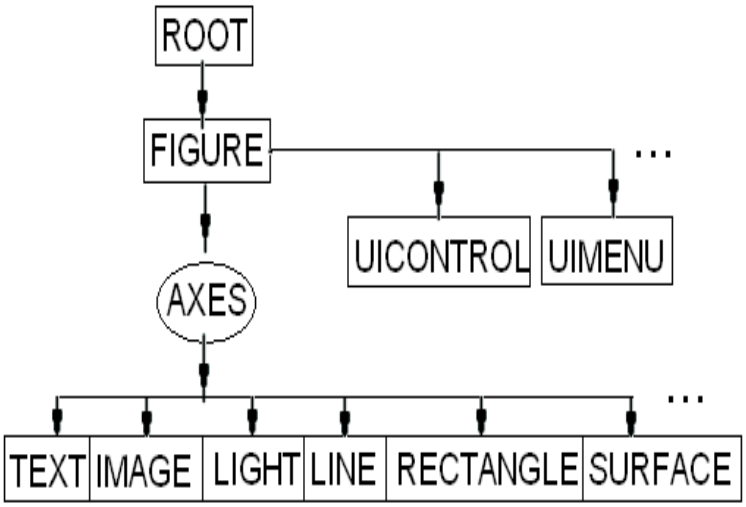


Рис. 1.6: Иерархия графических объектов в *MatLab*

нее на свойствах объектов *root*, *figure* и *axes*.

Root

Для *Command Window*–окна были ранее рассмотрены команды высоко уровня, управляющие свойствами объекта *root*. Как объект дескрипторной графики, он является структурой с регулируемыми свойствами (полей структуры) и

- командное окно – графический объект, имеющий нулевой дескриптор; все графические объекты являются его потомками

- командное окно – структура, свойства которой регулируются параметрами его полей

как было сказано ранее, *root* имеет нулевой дескриптор; информация о его свойствах доступна в результате выполнения команд, в соответствии с которыми

get(0) – получаем информацию о текущих параметрах полей; get(0,namefield) – позволяет отобразить текущий параметр поля

set(0) – получаем информацию о текущих и возможных параметрах полей; отобразить текущий параметр поля

v=get(0,'ScreenSize') – получаем четырехкомпонентный вектор – размер экрана, в единицах которые выбраны в поле *UNITS*, по умолчанию это *PIXEL*; они характеризуют размер экрана ($v(1:2)$ – координаты левого нижнего угла экрана, $v(3:4)$ – правого); указанная информация представляется очень важной, в частности, при программировании интерфейса пользователя (с целью обеспечить независимость GUI относительно текущего дисплея)

set(0,namefield, varfield) – можно изменить текущий параметр некоторого поля $namefield$, являющегося строкой, на параметр, представленный идентификатором $varfield$ (функцией или самим значением).

$\circlearrowleft$ **.** *Заметим, что регулирование параметров окна осуществляется и командами высокого уровня, например, из ранее описанных diary off, echo on или* $set(0,'format','long')$; в этом заключается известный дуализм команд высокого уровня, которые являются традиционными, и принципов объектно-ориентированного подхода при структурировании объектов в среде программирования ML, в том числе и графических.

Следующая таблица демонстрирует пример такого дуализма:

Дуализм управления Root

	функции	поля структуры
1	»hold on	set(1,'NextPlot','add')
2	»diary on	set(0,'diary','on')
3	»format long	set(0,'format','long')

На рисунке 1.7 показаны текущие свойства корневого объекта в момент визуализации графика.

```
>> x=[0:0.1:2*pi]'; y=sin(x); r=plot(x,y);
>> hold on, plot(x,z,'-r*')
>> get(0)
    CallbackObject = []
    CommandWindowSize = [130 57]
    CurrentFigure = [1]
    Diary = off
    DiaryFile = diary
    Echo = off
    ErrorMessage = Error: Expected a variable, function, or constant, found "=".
    FixedWidthFontName = Courier
    Format = short
    FormatSpacing = loose
    Language = russian
    More = off
    PointerLocation = [618 802]
    PointerWindow = [0]
    RecursionLimit = [500]
    ScreenDepth = [32]
    ScreenSize = [1 1 1280 1024]
    ShowHiddenHandles = off
    Units = pixels

    BeingDeleted = off
    ButtonDownFcn =
    Children = [1]
    Clipping = on
    CreateFcn =
    DeleteFcn =
    BusyAction = queue
    HandleVisibility = on
    HitTest = on
    Interruptible = on
    Parent = []
```

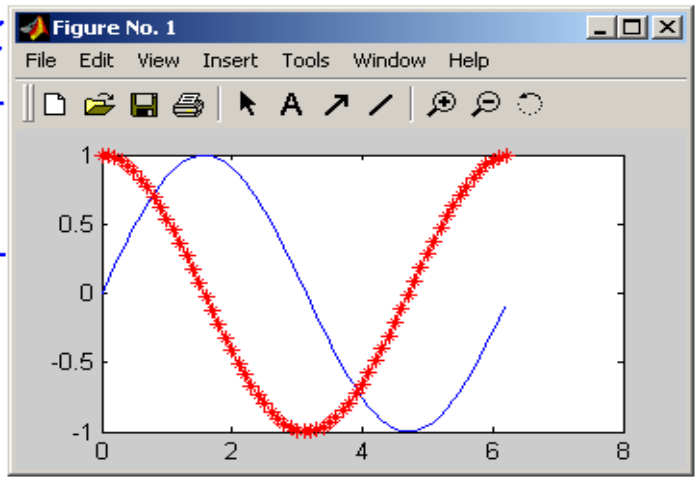


Рис. 1.7: Текущие свойства корневого объекта

Подчеркнуты имена и текущие свойства полей, некоторые из них разъяснены выше; из рисунка также следует, что графическое окно является потомком *root*, в то время как сам объект не имеет родительских. Остано-

вимся на редактировании важного свойства окна команд $Units$. Получив информацию о текущих и возможных его свойствах с помощью

- $>>\ set(0)$ – позволяет получить все возможные значения свойств-полей структуры; f.e. для поля $units\ :\ [inches|centimeters\ |normalized|points\ |\mathbf{pixels}|characters]$;

выясняем размер текущего монитора

- $>>\ get(0,'ScreenSize')$ по умолчанию $('unit' \longrightarrow 'pixel')$ получаем физические размеры монитора, f.e. [1 1 1280 1024];

и понимаем, что хотим "работать в среде безразмерных дисплеев"

- $>> set(0,'unit','normalized')$;
  $>> get(0,'ScreenSize')$

  получаем [0 0 1 1] – безразмерные параметры монитора;

дальнейшее моделирование графических объектов, являющихся элементами интерфейса пользователя, осуществляется в безразмерных величинах; это обеспечивает большую свободу функционирования программ (GUI) от выбора конкретного ПК.

Figure

Это графическое окно является потомком корневого графического объекта; следует выделить, что

- оно создается системой в момент создания любого из объектов более низкого уровня или пользователем с помощью

$$>> figure\ number$$

- $figure$ – структура имеет целый дескриптор ($number$), равный порядковому номеру окна;

- по умолчанию каждый новый график помещается текущее окно осей (регулируется параметром структуры поля $NextPlot$: $[\{add\} \mid replace \mid replacechildren\ ]$ это соответствует режиму $hold$ с параметром – on, режим – off замещает текущий график ;

- выбор окна с дескриптором m (и текущего) окна осуществляются соответственно по команде

$$>> figure\ m, \qquad >> figure\ gcf$$

здесь gcf–$graphic\ current\ figure$ — системный дескриптор текущей фигуры ($figure$). Графическое окно или серия фигур (с первой по текущую:) удаляются с экрана в соответствии с командами

$$>> delete(number), \qquad >> delete(1 : gcf)$$

$\circlearrowleft$. *Заметим, что командой* $delete(descriptor)$ удаляется любой графический объект.

Axes

Оси $Axes$ – это графический объект, который также создается системой при создании графических примитивов, при этом

- дескриптор – вещественного типа, генерируется системой;

- дескриптору текущих осей зарезервирован системный идентификатор gca; так же как и для предыдущих объектов, свойство осей доступны по команде $>> set(gca)$ (информация о параметрах полей, о координатах точки обзора, фонтах, о цвете поля, о типах линий, надписей, наличии или отсутствии сетки, названии осей, ориентации осей, единицах измерения и т.п.);

- очистка содержимого осей осуществляется по команде cla;

- все параметры можно изменять описанным ранее способом или интерактивно: в окне $Figure$

$$Figure \ \mapsto \ Edit \ \mapsto \ AxesProperties$$

- пользователь создает оси с помощью конструктора $axes$ командой высокого уровня:

$$ha = axes('position', rect)$$

здесь $rect$ задает размеры осей четырехкомпонентным вектором, первые два – координата левого нижнего угла, остальные – длины сторон.

Следующий пример демонстрирует, как следует перейти в шрифт нотации издательского пакета $LaTeX$, озаглавить оси – $xlabel$, $ylabel$; $legend$ – подписать (создать легенду графиков); озаглавить – $title$ окно.

Справа представлен график с выполненными приведенными командами:

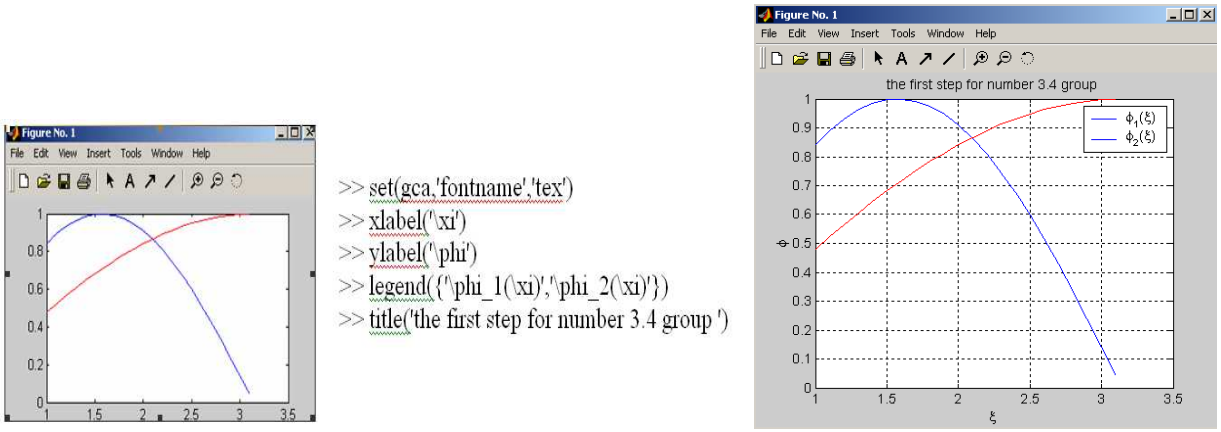


Рис. 1.8: Пример серии инструкций

В случае, если в оси требуется поместить фотографию, или иной объект, не нуждающийся в подписях и "засечках" на осях, нужно изменить параметры $XTick$, $YTick$ на пустой вектор [ ], а также серию параметров из

CameraPosition = [2.25 0.5 17.3205] CameraViewAngle = [6.60861] **Lot of Camera properties** Color = [1 1 1] - белое поле FontName = tex FontSize = [10] LineWidth = [0.5] NextPlot = add Projection = orthographic Position = [0.1 0.1 0.7 0.8] TickLength = [0.01 0.025] Title = [105] Units = normalized View = [0 90]	XGrid = on XLabel = [102.001] XAxisLocation = bottom XLim = [1 3.5] XScale = linear XTick = [1 1.5 2 2.5 3 3.5] Y. . . = Z. . . = Children = [(3 by 1) double array] Parent = [1] Selected = off SelectionHighlight = on Tag = Type = axes Visible = on

Таблица 1.1: свойств объекта *Axes*

таблицы 1.1, которые в этом случае не соответствуют логике изображения, $XGrid, \ldots$ – разметка сетки, подписи на осях и т.д.

$\circlearrowleft$. *Самый эффективный способ изучения возможностей, свойств среды программирования и ее элементов – метод "проб и ошибок"* :)).

4 *Вопросы для самопроверки:*

Точность вычислений связана с выбираемым форматом?

- Нет. Все вычисления производятся с двойной точностью
- Количество значащих цифр и точность соответствует, предусмотренному в форматах *long*, *short*, *bank*
- Точность вычислений может быть задана пользователем

Объект Root является

- командным окном?
- структурой с модифицируемыми свойствами полей?
- объектом дескрипторной графики с нулевым дескриптором?

Минимальной формой организации переменных является

- вещественный массив?
- скаляр?
- массив, каждый элемент которого состоит из вещественной и мнимой частей?

Проектное задание

Заключается в организации среды программирования. Изучении тематических справок (создание favorite), формирования глоссария по новой терминологии.

Глава 2

Классы MatLab

Современное программирование последние годы претерпело заметные изменения, связанные с объектно-ориентированным подходом при конструировании языка. Этот процесс затронул и пакет MatLab. Данный раздел посвящен описанию классов, методов и развитию идеологии матричного мышления при программировании кода программ.

1 Лекция 4. Объекты языка. Массивы классов Double, Sparse, Char. Связь с Maple

MATLAB позволяет реализовать концепцию объектно-ориентированного программирования, с ее помощью можно создавать и манипулировать объектами, которые скрывают реальные данные за контролируемыми методами работы с ними, допускают переопределение операций и дают возможность наследовать их свойства производным объектам-потомкам.

Однако, последний принцип объектно-ориентированного программирования, ***полиморфизм*** (возможность вызывать методы классов-потомков через интерфейсы базовых классов), в MATLAB не реализован. Конкретнее, в MATLAB нет ничего подобного механизму виртуальных функций в языке программирования C++.

Все основные классы типов MATLAB (числовые и разреженные масси-

вы, массивы ячеек, структуры и текстовые строки) представляют собой встроенные классы, а переменные – объекты этих классов. Пользователь имеет возможность вводить свои классы, а также переопределять и доопределять методы всех существующих классов.

Для создания нового класса объектов нужно спроектировать структуру MATLAB, которая будет хранить данные, принадлежащие объекту, и определить функции-методы работы с этими объектами. Эти функции определяются в обычных M-файлах, которые должны быть помещены в специальную папку, имя которой начинается с символа '@', а в остальном должно совпадать с именем структуры (класса), причем эта папка должна входить в папку, определенную в пути MATLAB-path.

Саму папку-контейнер методов добавлять в путь MATLAB не нужно.

Язык программирования MATLAB не имеет деклараций, в том числе деклараций новых классов и типов. Поэтому любой объект - представитель некоторого класса создается в момент вызова функции-конструктора этого класса. Следовательно, для создания объекта нужно создать хотя бы один метод (*конструктор*) в упомянутой *папке-контейнере* его класса. Все поля структуры, хранящей данные класса являются скрытыми (private), то есть их поля доступны только из методов данного класса, *напрямую в выражениях их использовать нельзя*. Таким образом для переменных $MatLab$ справедливы следующие положения

- В Matlab реализован ООП (объектно–ориентированный подход);
- Родителем любого объекта является массив;
- Отсутствует строгая типизация данных;
- Система идентифицирует тип объекта в момент создания;

и принципы объектно-ориентированного программирования:

наследование: возможность создания родительских и дочерних объектов; при наследовании (множественном наследовании) работают все методы родительских объектов – основа создания новых типов данных

инкапсуляция: объединение данных (и программ осуществляется через входные-выходные параметры функций), в результате чего образуются новые элементы программирования, объекты

полиморфизм: присвоение некоторому действию одного имени, которое работает по всей цепочке новые элементы программирования – в полной мере не реализован

агрегирование: объединение частей объектов или ряда объектов в одно целое – присуще только MatLab

users class создание пользовательских классов объектов MatLab с помощью конструктора классов (системные средства)

Иерархия классов отражена на рисунке 2.1 следующей схемой:

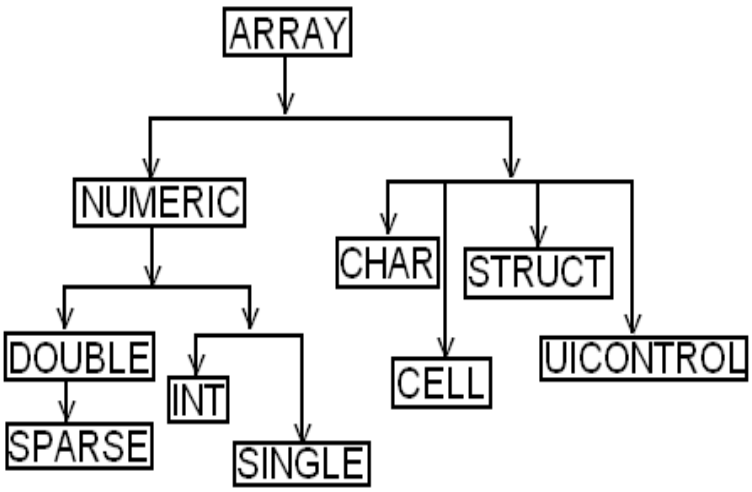


Рис. 2.1: Иерархия объектов

Класс *array* – "законодатель" класса массивов, в котором разработаны методы функционирования всех дочерних объектов.

Методы класса Array

Хранение массивов в ML осуществляется в векторной форме последовательно по столбцам; основой операций с массивами является сопряжение размерностей. Получить такую информацию можно с помощью перечисленных ниже функций

ndims $n = ndims(a)$ – возвращает количество размерностей многомерного массива

size – определяет вектор $v = size(a)$ размерностей;

так для $n = 2,\ [m, p] = size(a)$;

length – определяет длину вдоль большей размерности *двойная нумерация:* $a_{ij} : a(i, j) \longleftrightarrow a(m * (j - 1) + i)$

end – последний индекс массива

numel $numel(a)(= prod(v)$ – количество элементов

disp(display) – визуализация объектов, не подавляется знаком (;)

isnan, isinf определяет нечисловые элементы массива

f.e. $a = [1\ 2\ 3],\ b = [0\ 4\ 6],\ c = isnan(a./b)$ в результате c – массив, состоящий из логических нулей ($c = isinf(a./b)$) $c = [1\ 0\ 0]$); однако, если $a = [0\ 2\ 3],\ b = [0\ 4\ 6],\ c = isnan(a./b)$ тогда в результате $c = [1\ 0\ 0]$.

isnumeric, islogical, isequal, isempty: результатом применения являются – булевские массивы, определяющие числовой, логический тип объектов, их совпадение или проверку на наличие элементов.

Класс Double

Исторически пакет ориентирован на матричные вычисления двойной точности с элементами класса *Double*. В силу свойств наследования

- скаляр – тоже массив, минимальный элемент размера (1,1)
- в памяти матрица хранится как вектор по столбцам $\longrightarrow$ поддерживается двойная нумерация.
- Диапазон вещественных чисел $[-2^{1022}, 2^{2023}] \sim [realmin, realmax]$.

Рассмотрим способы создания, хранения визуализации, оптизации, основанные на векторизации операций:

создание:

- *перечислением*

  $> a = [1; 2; 3]$ – вектор–столбец;

  $> a = [123]$ или $> a = [123]$ – вектор–строка

  $> a = [1 : 3]$ индексированный вектор

- *с помощью*

  специальных матриц

  $> a = rand(size(b))$ – равномерно распределенные

  на отрезке [0,1];

  $> a = rand(m, n)$ – нормально распределенные

  с нулевым средним и единичной дисперсией

  $> a = ones(n), b = eye(n)$ из всех единиц, единичная матрица

  $> a = zeros(n)$ из всех нулей

- *импортированием* с помошью $File \longrightarrow ImportData$, либо программными средствами $xlsread('pathname.filename')$

  !!! n*amespreadsheet* - на английском

  $load'pathname.filename'$!!! данные могут быть разных типов, но иметь форму матрицы (таблицы), имя переменной массива создается по имени файла

- *с помощью форматированного ввода*

  $> fid = fopen('pathname.filename','permission')$

  −открытие и форма доступа; $'permission' = [r, w, a]$; $fid = 0$

  $> a = fscanf(fid, format, size)$,

  здесь$format = [\%g\ \%f\ \%e\ \%s\ \%c\ \%d]$,

  $size = [m, n]$ m − количество элементов в строке;

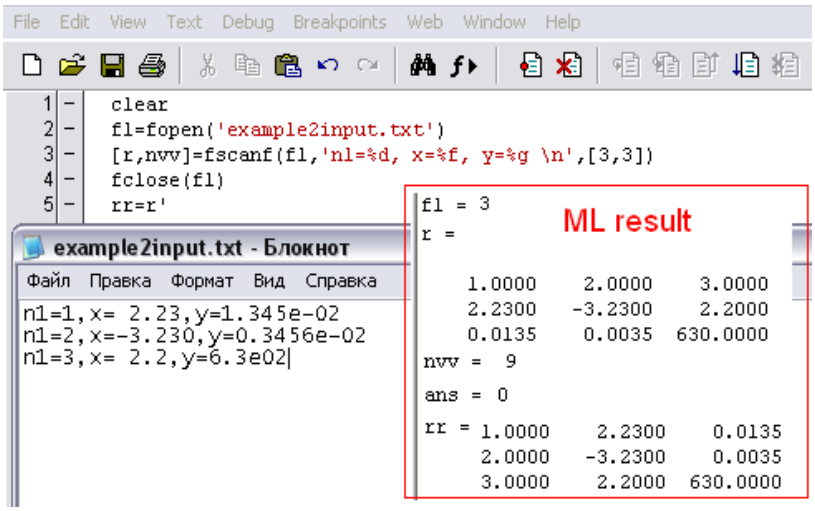
  n − количество элементов в столбце

  $n = inf$если

  $> fprintf(fid, format, namevar)$

  $> fclose(fid)$

Здесь приводится пример внешнего текстового файла, инструкций и результата их выполнения.



- *репликацией и конкатенацией*

  $> repmat(A, 2, 3)$

  $> [m, n] = size(a);\ b = reshape(a),\ [k, r] = size(b)$

  необходимо выполнение $m * n = k * r$

  $> cat(q, A, B)\ (\simeq [A, B],\ q = 2),\ (\simeq [A; B],\ q = 1)$

- *редактированием*

  удаление строк матриц

  $> a(1 : 2 : end, :) = [\ ]$

  выбор частей матриц (верхне-, нижнетреугольных матриц,

  диагонали или восстановления по ней диагональной матрицы):

  $> triu(a), tril(a), diag(a)$

  редактирование блоков, обнуление

  $> a(n : m, p : q) = 0$

  $> a(2 : 4, 3 : end) = zeros(size(a(2 : 4, 3 : end)))$

Рассмотрим пример оптимизации кода. Воспользуемся не столько векторными операциями, сколько навыками "векторного мышления".

Пример векторизации кода.

Традиционный код, использующий циклы,

```
tic
ww = w (ones(nf,1),:);
A = b.*ww*b';
toc
```

выполняется значительно медленнее:

```
t0 = clock;
for i = 1:size(b,1)
for j = i:size(b,1)
A (i, j) = sum (b (i,:).*b (j,:).*w);
end
for j = 1:i
A (i, j) = A (j, i);
end
end
etime(clock,t0)
```

Здесь в первом случае время измеряется секундомером, запускается-*tic*, останавливается-*toc*; во втором *clock* - измеряет текущее время, *etime* - определяет разницу.

Некоторые методы и операции линейной алгебры

Широко используются для осмысленного проведения прцесса обучения или научного исследования, так

качественный анализ элементов матриц sort - упорядочивание, find - поиск ненулевых элементов, min, max, sum, - определение минимальных, максимальных элементов массива и суммирование (следует отметить, что если не указано вдоль какой размерности осуществляется анализ, производится по столбцам - вдоль строк)

специальные функции с помощью *eig* - определяются собственные значения и векторы; *inv*, *conj* - обращение и комплексное сопряжение

факторизация матриц и решение систем $lu(a)$ - факторизация a на верхне- и нижнетреугольную матрицы; $qr(a)$ ($[Q, R] = qr(a, b),\ a = Q * R$), R - ортогональная матрица; в случае симметричной матрицы реализован метод Холецкого; $R = chol(a), x = R_{\backslash}(R * R' = a)$; с

помощью $linsolve(a, b)$ получается решение линейной системы, тот же результат достигается: $(Ax = b, \quad x = A\backslash b)$.

Класс Sparse - разреженных матриц

В результате дискретизации краевых задач получаются конечномерные аналоги, которые представляют собой сильно разреженные системы линейных алгебраических уравнений. Точность решения сеточных или конечно-элементных методов увеличивается (замедляясь в результате накопления ошибки вычислений) с ростом порядка системы. Для таких объектов разработаны эффективные способы хранения матриц и решения систем. С этой целью в $MatLab$ разработан класс $sparse$, он же конструктор.

Для матрицы A – полной, определяется массив, который хранит индексы ненулевых элементов и сами числа с помощью

$$[i, j, [v]] = find(A);$$

при этом, если число ненулевых элементов $(nnz(A)) < prod(size(A)/2)$ мало, то целесообразно преобразовать полную матрицу в разреженную:

$$s = sparse(a).$$

Визуально оценить количество ненулевых элементов можно посредством функции $spy(s)$; на рисунке 2.2 демонстрируется такая визуализация и предлагаются инструкции; результат выполнения которых поясняет сущность элементов описываемого класса.

Некоторые процедуры, предусмотренные для разреженных форматов, начинаются sp и указывают на принадлежность классу:
$spalloc$ – выделяется динамически память, $spones$ – строится матрица из единиц, $speye$ – единичная, $sprand$ – случайная (равномерное распределение); $sprand$ – случайная (нормальное распределение); $spconvert$, $full$ –

```
r =
    0.2920    0.3567    0.1133    0.6700    0.0595
    0.8580    0.4983    0.8983    0.2009    0.0890
    0.3358    0.4344    0.7546    0.2731    0.2713
    0.6802    0.5625    0.7911    0.6262    0.4091
    0.0534    0.6166    0.8150    0.5369    0.4740
>> r=randint(5)
r =  1     0     1     0     1
     1     1     1     0     0
     0     1     1     1     1
     0     1     0     1     1
     1     0     0     0     0
>> spy(r)
>> t=sparse(r)

t =

   (1,1)        1
   (2,1)        1
   (5,1)        1
   (2,2)        1
   (3,2)        1
   (4,2)        1
   (1,3)        1
   (2,3)        1
   (3,3)        1
   (3,4)        1
   (4,4)        1
   (1,5)        1
   (3,5)        1
   (4,5)        1
```

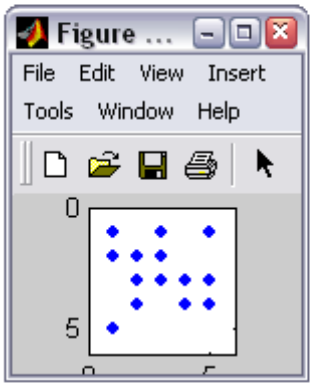


Рис. 2.2: Разреженные матрицы. Связь с полными.

возвращает полную из разреженной, справка по команде

$$>> help\ sparfun$$

предоставит все методы класса.

Класс Char

Строка - любой набор знаков, заключенных в кавычки, например,

$> a =' it\ is\ ''true'''$

$> b =' it\ is\ ''false'''$

$> c = [a, b] -$ возможно, $c = [a, b]$несовпадение строк

Каждый элемент строки занимает два байта (+ зарезервированная мнимая часть), наследуются все методы $array$ ($size$, $length$, $ndims \dots$), а также правила редактирования и сопряжения размерности.

Некоторые специальные функции

Сравнение массивов строк осуществляется по команде $strcmp('Matlab','Maple')$ – результатом является логический ноль или единица.

Контекстный поиск меньшей строки в большей осуществляется по команде $res = findstr(str1, str2)$,
res - вектор адресов начал подстроки либо пустой вектор.

Конвертирование массивов строк в элемент $Numeric$ осуществляется по команд $str2num(str)$, $str =' 1.23456' \sim str2double(str)$, $str = 1.2356e-5$ – связь с $double$ $str2mat(str1, str2, \dots, strn)$ – строки преобразуются в матрицу с согласованием размерностей.

Выполнить строку $eval(varChar)$, здесь varChar–строковая переменная, как правило, выражение в синтаксисе языка $MatLAb$; $feval(nameFile)$ выполнить файл с именем $nameFile$.

Редактирование аналогично *double*.

Преобразование аналитики осуществляется командами: $simplify$, $subs$, $factor$ – так же, как и в $Maple$.

$\circlearrowleft$. *Заметим, что в varChar в случае необходимости следует произвести векторизацию операций, например, с помощью процедуры vectorize, исходный код преобразуется к виду* $eval(vectorize(varChar))$.

Связь с классами MatLab

В соответствии со следующими конвертерами осуществляется преобразование строк в объекты "смежных" классов.

- $double(r) \mapsto$ 120 121 122 - конвертирование строк в коды таблицы $ASCII$
- sym(str1) - конвертирование строки в символы
- char(sym1) - конвертирование символов в строки

Объекты ядра Maple

Ядро символьных преобразований – коммерческое, основано на иных принципах функционирования (в чужой монастырь со своим уставом). Так символьная арифметика в пакете является декларируемой:

$$syms\ x,\ b = sin(x) * x^2,\ char(b)$$

Все команды ядра Maple выполняются в формате:

$$maple('n.f., arg_1, arg_2, ..., arg'_n)$$

Пример использования функции ядра:

$$> syms\ x$$

$$> a = (x+1)^2$$

$$> maple('coeff', a, x)$$

Существует ряд функций, аналогичных Maple, которые написаны впоследствии для строк переменных и не требуют использования специального обращения к ядру.

$$simplify,\ factor,\ expand,\ subs, \ldots$$

Методы, которые предусмотрены для класса символьных объектов, доступны по команде

$$methods(sym,)$$

это тригонометрические функции, специальные (Бесселя, Эри...), преобразования Фурье, Лапласса, интегрирование, дифференцирования, поиск конкретного символа, операции над матрицами, операции линейной алгебры, конвертирование в строковый массив и т.п. Выделим лишь функцию $findsym$, с помощью которой определяются имена неизвестных в уравнении, а также $solve$ – функцию решения систем и уравнений.

Построение символьной графики осуществляется командами $ezplot,\ ezsurf,\ ez\ldots$, при этом графические объекты не являются структурами, как в MatLab.

Практически все методы, работающие для элементов ядра, имеют такие же имена и применяются для объектов $Double\ Sparse$ и т.п.

2 Лекция 5. Массивы ячеек. Структуры

Массивы ячеек

Массив ячеек – класс, который является специфическим классом для языков программирования, поскольку этот класс позволяет объединять разнородные объекты, которые принадлежат разным типам данных, и для каждого из которых поддерживаются свои методы.

Cell – конструктор класса, однако с его помощью только задается размер массива разнородных объектов - ячеек. Фигурные скобки используются для перечисления его элементов, а также для указания индексов при оперировании его объектами.

Способы созданиия делятся на декларативные, описательные, конвертационные. Так, по команде *cell* задается размер массива ячеек, выделяется "псевдопамять"–как форма организации – декларирования размера. К числу описательных – относятся способы перечисления элементов, непосредственного присвоения; конвертационных – способы применения функций связи, позволяющих конвертировать элементы иных классов в массивы ячеек. Далее приводятся примеры, дающие представления об указанных приемах.

1) $c = cell(n),\ c = cell(size(A)),$ – **резервирование**

2) **присвоение:** $c = \{zeros(4), 'sin(x)', sparse(rand(4))\}$

3) $c = \{[1]\ [2\ 3\ 4]; [5;\ 9]\ [6\ 7\ 8; 10\ 11\ 12]\}$

4) $c = \{'mainnamefile',\ 'proc1',\ 'sin(x^2)/(3 * pi * x^2)]\}$

5) **как результат конвертирования структуры**

6) **как результат конвертирования массива строк**

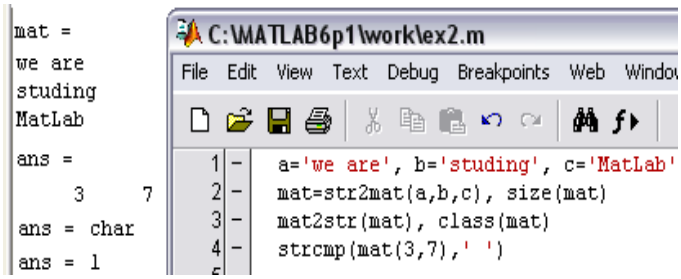
Здесь в пункте (2) задан вектор-ячеек, первый элемент которого нулевая квадратная матрица четвертого порядка; второй – строка; третий – слу-

чайная разреженная матрица четвертого порядка.

↻ . *Следует отметить, что элементы массива ячеек не отображаются в строке вывода, и для визуализации всех элементов массива ячеек предназначена команда*

$$celldisp(c), \quad \textit{здесь c-массив ячеек.}$$

Рассмотрим **Пример 1. связи** *CHAR* **и** *Double Cell*




```
>> num2cell('mat')
ans =  'm'    'a'    't'
>> num2cell(mat)
ans =  'w'    'e'    ' '    'a'    'r'    'e'    ' '
       's'    't'    'u'    'd'    'i'    'n'    'g'
       'M'    'a'    't'    'L'    'a'    'b'    ' '
```

Здесь a, b, c – строки разной длины, которые конвертируются в матрицу из трех строк и семи столбцов; строки дополняются пробелами и в результате имеют равную длину. Конвертирование в массив ячеек приводит к тому, что каждые элемент массива ячеек – строка, состоящая из одной буквы или пробела.

Конвертирование массива ячеек в матрицу *Double* осуществляется по команде:

cell2mat с учетом согласования размерностей.

В третьем пункте приведен пример конвертирования массива ячеек в

матрицу $r = cell2mat(c)$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

Не требует специальных пояснений конвертирование массивов *Double* в *Cell*, суть процедуры следует из предлагаемых примеров с использованием следующих команд:

mat2cell r=mat2cell(a,m,n), здесь m – вектор количества строк подматриц;
n – столбцов подматриц;

num2cell r=num2cell(a,m,n), экспортирование данных типа *Double* и *Char*

Пример 2. *Double* $\longrightarrow$ *Cell*

```
>> a=randint(4,4,4), r=mat2cell(a,[1,3],[2,2])

a =

     1     3     2     1
     3     0     1     2
     1     2     0     3
     0     2     1     1


r =

    [1x2 double]    [1x2 double]
    [3x2 double]    [3x2 double]
```

Пример 3. *Адресации в ячейках.*

$$>> a = cell(1,2)$$

$$>> a\{1,1\} = \{'str', \{1,2\}, eye(2)\},$$

$$>> a\{1,2\} = \{ones(2)\}$$

$$>> r = a\{1,1\}\{1\}(2)(\rightarrow r = t)$$

$$> ischar(ans)(\rightarrow 1)$$

$$>> a\{1,1\}\{2\}\{1\}(\rightarrow 1)$$

$$>> a\{1,1\}\{1\}(3)(\rightarrow r)$$

$$>> a\{1,1\}\{3\}(4)(\rightarrow 1)$$

$$>> a\{1,1\}\{3\}(\rightarrow [1\ 0; 0\ 1])$$

Контроль элементов структуры

В том случае, если логика программирования требует проверки типов элементов массива ячеек, такая проверка осуществляется с помощью оператора (хотя ее назначение существенно шире)

$$D = cellfun(FUN, C), \quad D = cellfun('size', C, K)$$

здесь результат является истинным, если тип всех элементов массива ячеек C сравним с FUN; далее приведены возможные значения FUN: $'isreal'$ – истина для вещественных элементов; $'isempty'$ – для пустого массива; $'islogical'$ – для логического массива элементов; $'length'$ – длина массива ячеек; $'ndims'$ – размерности; $'prodofsize'$ – количество элементов. Реально функция FUN применяется к структуре C.

Структуры пользователя

Массив структур – класс, который также позволяет объединять разнородные объекты с помощью поименованных полей, точкой они отделяются от имени структуры.

Struct – конструктор класса, с помощью и создается структура. В изучаемом курсе мы имели дело с системными структурами, такими как командное окно–*Root*, $Figure$ и т.п.; каждое поле структуры объединяет однородные объекты, тогда как элементы всех полей представляют разнородные объекты.

Далее приводятся примеры создания пустой, начальной и элементарной структур, в которых $'namefield\_i', value\_i$ – имя i–го поля и i–го значения соответственно:

struct – конструктор

$s = struct('namefield\_1', \{\}, \ldots, 'namefield\_n', \{\},)$

– создание пустой структуры;

$s = struct('namefield\_1', value\_1, \ldots, \ldots$

$'namefield\_n', value\_n)$

– создание начальной структуры;

$s = struct('namefield\_1', eye(3), 'namefield\_2', \{'sin'\})$

– создание элементарной структуры

Из определения структур и ячеек следует родственная природа данных объектов. Если в структурах озаботиться названиями полей и задать их, то очевидна возможность их конвертирования в структуры; равно как и в случае отсутствия нужды в смысловом распознавании полей структуры, целесообразно ее конвертировать в массив ячеек; далее приводятся примеры, поясняющие специфику такого преобразования:

cell2struct >s=cell2struct(a,strnamefield,dim) здесь a - массив ячеек из примера 3

>s=cell2struct(a{1,2},'matrix',1) ($\longrightarrow$ [1 1; 1 1])

>s=cell2struct(a{1,1},'nonhom',3) ($\longrightarrow$ s.nonhom состоит из трех разнородных элементов: слова str, массива ячеек из двух чисел, и единичной матрицы)

struct2cell >c=struct2cell(s) здесь s - структура.

Редактирование массива структур

Рассмотрим процедуру редактирования структуры. Так текущему полю $field$ присваивается значение v следующими способами:

редактирование полей структуры

>s = setfield(s, 'field', v)

>s = setfield(s, {i,j}, 'field', {k},v)

>s(i,j).field(k)=v

удаление полей структуры >s = rmfield(s, 'field' – по этой команде удаляются все элементы поля и само поле

получение содержимого поля осуществляется по команде:

>f = getfield(s,'field')

>f = getfield(s,{i,j},'field',{k}) или s(i,j).field(k)

>f = getfield(s,{1,1},'r',{2}) ($\longrightarrow$ t)

получение всех имен структуры –

>names = fieldnames(s)

$\circlearrowleft$ **.** *Заметим, что names – имена полей является массивом ячеек, элементы которого – строки*

Конвертирование

```
s = 1x3 struct array with fields:
        mycell
res =   1     0
        0     1
s1 = mydouble: [2x2 double]
r1 = 1
r2 = r
ans = 1
ans = 2
```

C:\MATLAB6p1\work\cellstr.m

File Edit View Text Debug Breakpoints Web Window Help

```
1 - clear
2 - a{1,1}={'str',{1,2},eye(2)},  a{1,2}={ones(2)}
3 - s=cell2struct(a{1,1},'mycell',[3,1])
4 - res=getfield(s,{1,3},'mycell')
5 - s1=cell2struct(a{1,2},'mydouble',[2,2])
6 - r1=s1.mydouble(2,2)
7 - r2=s(1,1).mycell(3)
8 - s(1,2).mycell{1:2}
```

Здесь приведена последовательность программного кода и результат выполнения (конвертирования массива ячеек в серию структур). Рассмотрим далее пример структуры и конвертирование ее в массив ячеек:

Пример 4. (1×3 *struct array with fields : name age status*)

$> s = struct('name', \{'kim','tan','pak'\},'age', \ldots$

$\{8, 20, 28\},'status', \ldots,$

$\{'schoolboy','student','worker'\})$

$> names = fieldnames(s)$ $(names -)$

$> getfield(s, \{1, 1\})$

kim 8 schoolboy

$> getfield(s, \{1, 1\},'name', \{1\})$

k

$> s = setfield(s, \{1, 1\},'name','rik')$

$> getfield(s, \{1, 1\},'name', \{1\})$

rik

структура - аналог многомерного разнородного массива!

Обратное конвертирование (в *cell*) осуществляется по команде

$$> c = struc2cell(s)$$

в результате структура примера 4. преобразуется к виду

$$c\{1,1,1\} \to rik$$

$$c\{2,1,1\} \to 8$$

$$c\{3,1,1\} \to schoolboy$$

$$c\{1,1,2\} \to tan$$

$$c\{2,1,2\} \to 20$$

$$c\{3,1,2\} \to student$$

$$c\{1,1,3\} \to pak$$

$$c\{2,1,3\} \to 28$$

$$c\{3,1,3\} \to worker$$

Каждые новый “лист” содержит полную информацию о текущей персоналии.

3 Лекция 6. Конструкции языка. Тенденции векторизации

Условный оператор

Условный оператор – практически обязательная лексема языков программирования, полная форма которой имеет вид:

```
if expression1
   statements1
elseif expression2
   statements2
else
   statements3
end
```

здесь $expression1, \ldots$ – выражения, связанные операцией отношения из

```
expression   rel_op  expression

rel_op: ==, <, >, <=, >=, or (|) ,~=, and (&)
```

указанного списка. Последовательность кода после $*if$ выполняется в случае, если результат отношения – логическая единица ($true$) в противном случае выполняются операторы, следующие после $else$. Короткая форма ограничивается одним условием и совокупностью последующих инструкций. Из приведенной далее таблицы следует правило генерирования результата сравнения в операциях отношения.

Очевидно, что операции отношения также оптимизированы и “заточены” под матричные операции, это освобождает от использования традиционных циклов.

A = **B =**

1 0 **1 1**

2 3 **3 4**

Expression	Evaluates As	Because
`A < B`	`false`	`A(1,1) < B(1,1).`
`A < (B + 1)`	`true`	$A_{ij} < B_{ij} + 1$
`A & B`	`false`	`A(1,2) & B(1,2)` is false.
`B < 5`	`true`	$B_{ij} < 5.$

Циклы с известным числом повторений

```
for variable = expression
   statement
   ...
   statement
end
```

```
for index = start:step:end
   statements
end

for index = start: end
   statements
end
```

start – начальное значение индексной переменной (и.п.);

step – шаг изменения индекса; *end* – конечное значение и.п.

По умолчанию $step = 1$.

```
>> for e = eye(n), e, end
```

На i–м шаге вычисляется вектор-столбец $e(:, i)$,

$$for\ s = 1.0 :\ -0.1 :\ 0.0, ...,\ end$$

– данный пример иллюстрирует цикл с отрицательным шагом цикла.

Циклы с бесконечным числом повторений

```
while expression
  statements
end
```

```
2  while (A & B)

3  while (A | B)

4  while (b ~= 0) & (a/b > 18.5)
   if exist('myfun.m') & (myfun(x) >= y)
   if iscell(A) & all(cellfun('isreal', A))

5  eps = 1;
   while (1+eps) > 1
       eps = eps/2;
   end
   eps = eps*2
```

- В примере **2.** если матрица A – вся из единиц, то матрица B – не вычисляется, т.к. результат для любого B – истина
- В примере **3.** если выражение (матрица) A – нулев(ая)ое, то матрица B – не вычисляется, т.к. результат для любого B – ложный;
- В примере **4.** бесконечный цикл выполняется пока справедлив $a/b > 18.5$, при этом при этом исключается деление на ноль; контролируется величина вычисляемой функции (обеспечивается контроль ее существования); условным оператором также контролируется вещественность элементов массива ячеек
- в примере **5.** Здесь на первый взгляд – бесконечный цикл, однако он прекратится по достижении машинного нуля;

Переключатели

Традиционной конструкцией языка является переключатель. Его синтаксис расширен возможностью использовать массивы ячеек в качестве индикаторов реализуемых "веток", это отражено в форме оператора *switch_expr* – вычисляемое выражение; результат сравнивается со значе-

```
switch switch_expr
 case case_expr
   statement,...,statement
 case {case_expr1,case_expr2,case_expr3,...}
   statement,...,statement
...
 otherwise
   statement,...,statement
end
```

ниями отдельных случаев *case_expr* или контейнеров – ячеек, в случае совпадения выполняются инструкции таким образом "помеченной" ветки, в противном – *otherwise*.

```
method = 'Bilinear';
switch lower(method)
  case {'linear','bilinear'}
    disp('Method is linear')
  case 'cubic'
    disp('Method is cubic')
  case 'nearest'
    disp('Method is nearest')
  otherwise
    disp('Unknown method.')
end
```

Оператор перехвата

Очень важной конструкцией языка является оператор перехвата– “перехватчик”, как правило, ошибок. В рамках нестрогой типизации, программист “тащит” тяжелый “воз” проблем, связанный с обеспечением принудительной проверки типов данных. В острых ситуациях, когда ошибка представляется возможной и обусловлена логикой процесса, а возможные сообщения об ошибке исчерпываются некоторым разумным списком таких “ошибочных” сообщений, можно избежать несанкционированного прекращения работы программы.

Для корректной обработки ошибки служит оператор

```
try,
statement,
...,
statement,
catch,
statement,
...,
statement,
end
```

Системная переменная $lasterr$ хранит информацию о последнем сообщении об ошибке.

На этапе $try - catch$ возможна ошибочная ситуация; на этапе $catch - end$ происходит обработка ошибки, например, с помощью функции $findstr(lasterr, listerrori)$, в зависимости от конкретики возможны варианты “сознательного регулирования потоков”.

Таким образом, синтаксис языковых конструкций совпадает с традици-

онным – для других языков программирования Если в операциях отношения, результат обусловлен значением одного выражения, то другое не системой вычисляется. Циклы выполняются в случае крайней нужды, эффективнее векторизация (имманентная или оптимизационная)

На следующей – приводится пример использования конструкций языка и различных типов данных; пример создания случайной матрицы, на ее основе – разреженной; пример построения графиков функций, редактиро-

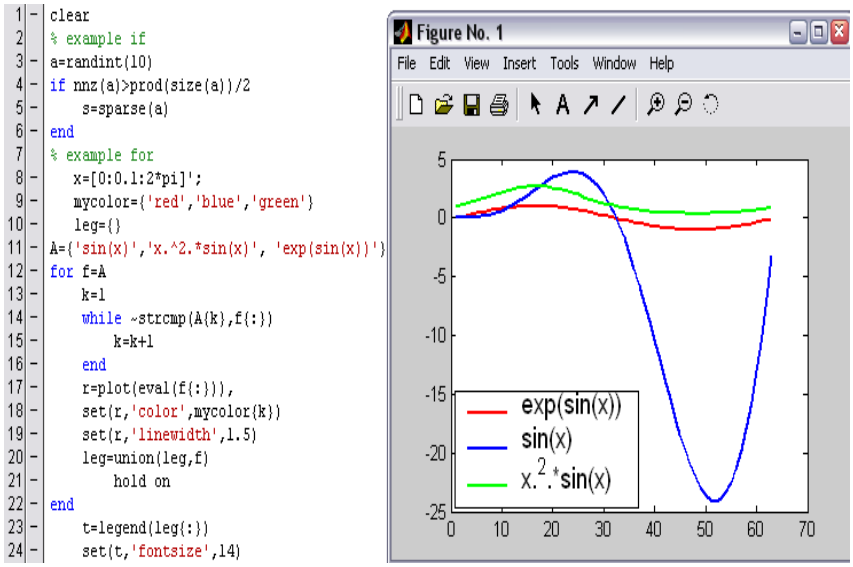


Рис. 2.3: Редактирование графических объектов

вание свойств структур графических объектов (линий).

Пример векторизации кода

Матричное мышление – это процесс, опирающийся на знание синтаксиса языка, опыт и развивающийся в рамках неустанного использования метода “проб и ошибок”. Далее приводятся код,

```
ww = w (ones(nf,1),:);
A = b.*ww*b';
```

который выполняется значительно быстрее, чем код

```
for i = 1:size(b,1)
```

```
for j = i:size(b,1)
A (i, j) = sum (b (i,:).*b (j,:).*w);
end
for j = 1:i
A (i, j) = A (j, i);
end
end
```

доказывающие преимущества векторизации. Код лаконичный и эффективный. Здесь очевидно вычисление элемента $A(i, j)$ согласно правилам умножения матриц может быть представлен как скалярное произведение векторов в первом случае и покоординатное произведение и суммирование - во втором. Надо сказать оба варианты представляют формы оптимизированного кода по сравнению с традиционными языками, в которых не поддерживается векторизация операций.

4 *Вопросы для самопроверки:*

Символьные объекты являются

- элементами ядра Maple
- массивами – объектами среды программирования MatLab?
- аналогом строковых массивов в среде ML, поддерживаются методы $CHAR$

Равносилен ли код $b = 1 : n,\ ones(size(b)) * b'$ **коду**

- $sum(1 : n)$
- $b = 1 : n,\ b * ones(n, 1)$
- $b = 1 : n,\ sum(b. * ones(length(b)))$ графики с нулевым дескриптором?

Структура –

- предназначена для объединения разнородных объектов?
- является аналогом массива ячеек с поименованными “размерностями”?
- в рамках одного поля–свойства объединяет разнотипные объединяет разнотипные объекты?
- пользователя создается конструктором *struct*, *structure*?

Массив – ячеек

- предназначена для объединения разнотипных объектов, не нуждающихся в смысловом объединении?
- не отображается в строке вывода, является исключением?
- объединяющий элементы, являющиеся строками, приводит к их конкатенации?
- задается перечислением элементов в фигурных скобках, квадратных, круглых?

Проектное задание

Заключается в выполнении типовых заданий, например, по созданию сложной блочной матрицы, с выполнением серии редактирования по предложенному алгоритму, проверкой ее свойств и визуализацией каждого этапа редактирования в отдельном окне. Выполнение операций линейной алгебры, решение систем, оценка эффективности в случае полных и разреженных матриц.

Глава 3

Графические объекты. GUI как независимое приложение

Пакет является не только мощным средством проблемного программирования, но также предусматривает возможность реализации независимых приложений, оптимизации, получения библиотек, сопряжения модулей, написанных на таких языках программирования как C++, и Fortran. Дополненный функциями создание элементов пользовательского интерфейса, пакет представляет собой мощное средство написания оболочек, предназначенных для функционирования в динамических рамках различных предметных областей. Изучение вышеперечисленных возможностей излагается в текущем разделе.

1 Лекция 7. Графические объекты: Line и Text

Line

Объект класса *Line* создается как конструктором класса объектов, для которого поддерживаются различные синтаксические формы,

$$line(X, Y),\ line(X, Y, Z),\ line(X, Y, Z, 'PropertyName', PropertyValue, ...)$$

так и с помощью команд высокого уровня

$$plot(Y), \quad plot(X1, Y1, ...), \quad plot(X1, Y1, LineSpec, ...),$$

$$line(X, Y, Z, 'PropertyName', PropertyValue, ...),$$

здесь $'PropertyName'$ – имя свойства графической структуры, $'PropertyValue'$ – его значение; свойства линии в $LineSpec$ связаны с характеристиками стиля линии, цветом линии и формой маркеров (точек). В следующей таблицы приведены некоторые стили линий а также формы маркеров:

Свойства линии Line	
LineStyle	***стили*** :-, --, :, -., Default: -
LineWidth	***толщина*** Default: 0.5 points
Marker	***маркер*** Default: none
MarkerSize	***размер маркера*** Default: 6

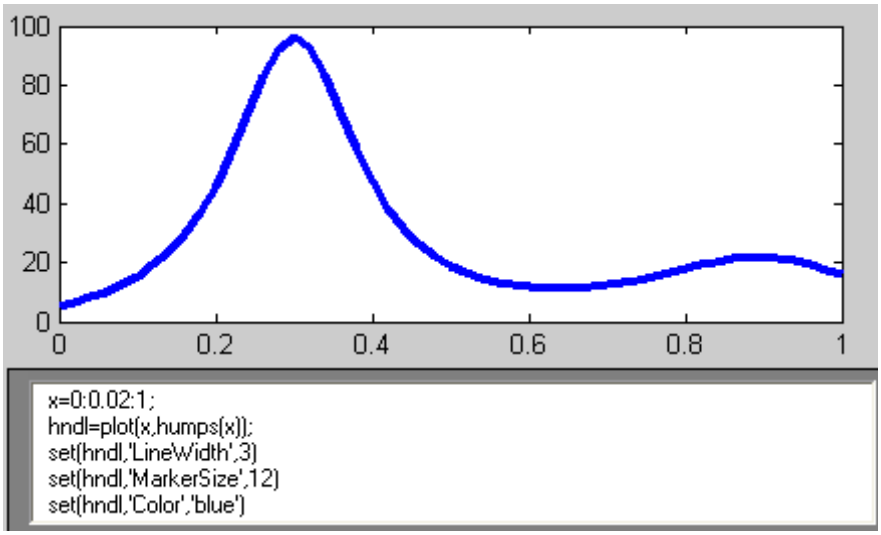
Marker	
+	
o	
*	
.	
x	
s	**квадрат**

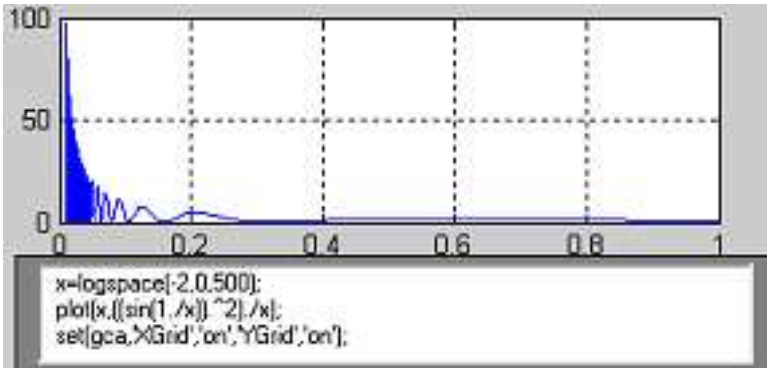
Заметим, что если в *plot* в качестве первого аргумента вектор–столбец, а второго – матрица, в этом случае будут построены графики по числу столбцов матрицы, ординаты которых являются столбцами указанной матрицы.

Далее представлена системная функция *humps*, график которой с дескриптором *hndl* отображен на приведенном рисунке, при этом у структуры *hndl* изменены свойства полей, а значит и свойства линии: ее толщина, размер маркера, а также цвет линии.

Следует отметить что по умолчанию сетка на осях отсутствует. Следующий пример демонстрирует активацию режима нанесения сетки.

В пакете реализована возможность построения линий заданных аналитическими выражениями в виде строк, а также систем, как пересечение


100
80
60
40
20
0
0
0.2
0.4
0.6
0.8
1
x=0:0.02:1;
hndl=plot(x,humps(x));
set(hndl,'LineWidth',3)
set(hndl,'MarkerSize',12)
set(hndl,'Color','blue')


100
50
0
0
0.2
0.4
0.6
0.8
1
x=logspace(-2,0,500);
plot(x,([sin(1./x)].^2)./x);
set(gca,'XGrid','on','YGrid','on');

заданных поверхностей:

$$ezplot(expressionChar)$$

Text

Графические объекты представляемого класса служат для оформления графиков, осей, различных элементов "видеоряда". Конструктор создания текста имеет вид:

$$TEXT(X, Y, 'string'),$$

здесь X, Y – координаты помещаемого текста, $'string'$ – сам текст.

Однако для удобного и быстрого функционирования используются команды высокого уровня:

$$xlabel,\ legend,\ title,\ \ldots$$

↻ . *Заметим, что координаты X, Y – задаются в тех единицах, которые определены в поле $Units$.*

Так, на рисунке 3 предложен код, с помощью которого построены графики и создана легенда. Все описанные возможности конструктора и команд предусматривают создание надписей программными средствами (в отложенном режиме взаимодействия), но существует команда интерактивного помещения текста в визуально требуемую позицию

$$h = gtext('string').$$

При выполнении скрипт-файла или функции, содержащих $gtext$, система останавливается на указанной команде, движение мыши отображает перекрестье "прицела", которое позволяет точно выбрать позицию, по правой кнопке мыши, строка, являющаяся параметром функции отображается на поле осей. Демонстрация диалога с пакетом во время использования $gtext$ представлена на двух следующих пакетах.

```
1  clear
2  namefunc={'sin(x)*cos(2*x)','sin(2*x)/cos(x)','x^2/(x-3^4)'}
3  x=(-pi/2:0.1:pi/2)'
4  y=size(x)
5  y=eval(vectorize(namefunc{1}))
6  for i=2:3
7      y=[y, eval(vectorize(namefunc{i}))]
8  end
9  r=plot(x,y(:,1),'-rs',x,y(:,2),'-b*',x,y(:,3),'-g>')
10 legend(namefunc{:})
```

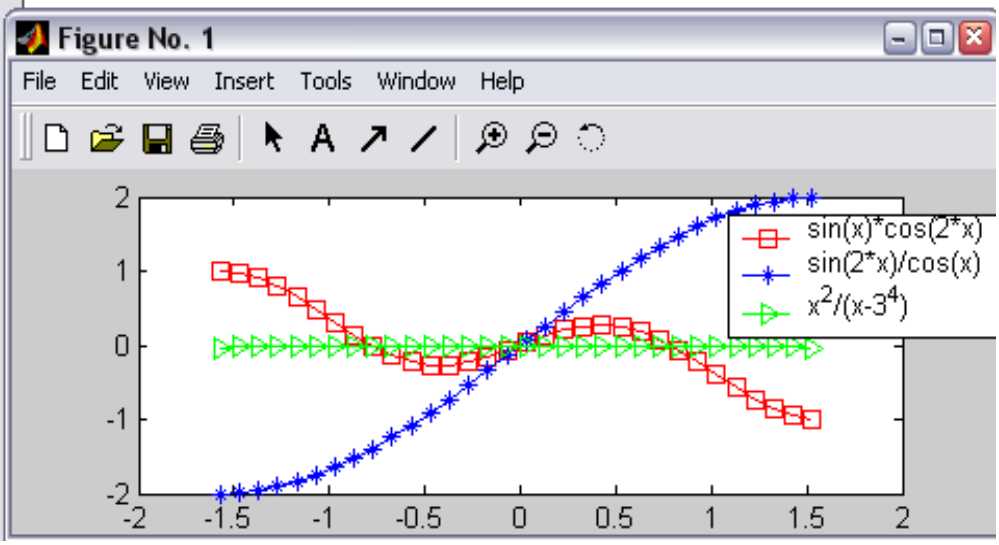
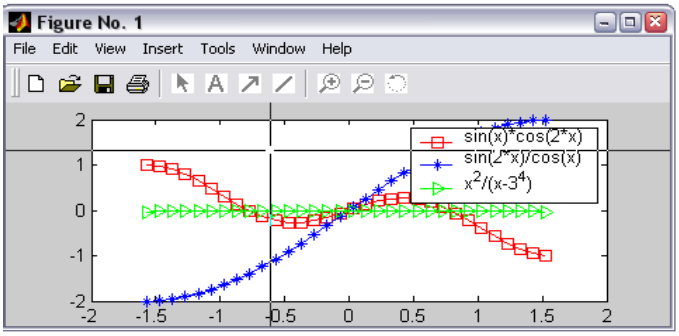
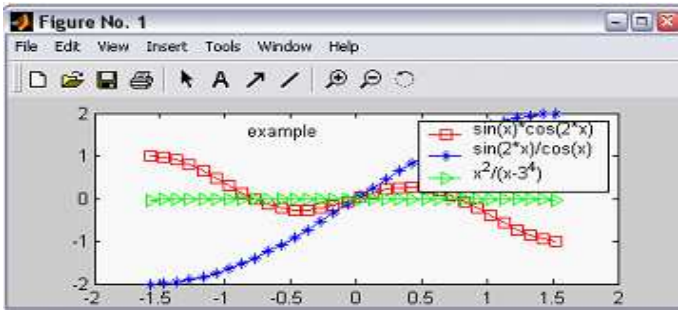


Рис. 3.1: Пример построения серии графиков с легендой

```
gtext('example')
```





Пример 3.2 демонстрирует, как можно подгрузить издательский пакет LaTex, чтобы с его помощью все подписи в случае необходимости имели математическую нотацию. Здесь первые три оператора формируют надписи под осями X, Y, соответственно, и заголовок.

Следующий за ними оператор – конструктор выбирает все объекты синего цвета ([0 0 1]). В нашем случае "все"– исчерпывается одним графиком и для него с помощью *set* производится редактирование свойств: синий цвет заменяется красным, и толщина линии увеличивается до двух.

```
xlabel('-\pi \leq \Theta \leq \pi')
ylabel('sin(\Theta)')
title('Plot of sin(\Theta)')
text(-pi/4,sin(-pi/4),'\leftarrow sin(-\pi\div4)',...
     'HorizontalAlignment','left')

set(findobj(gca,'Type','line','Color',[0 0 1]),...
    'Color','red',...
    'LineWidth',2)
```

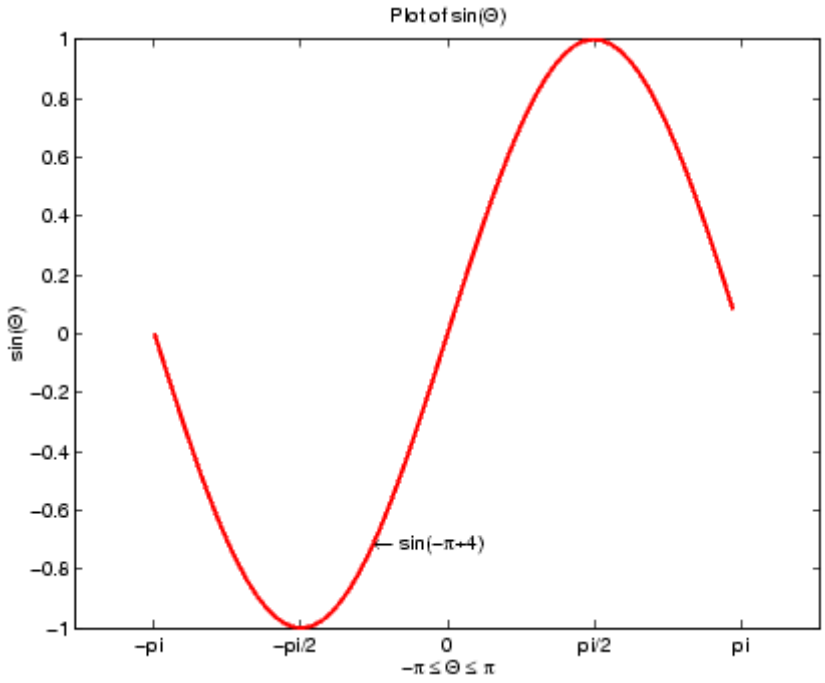


Рис. 3.2: Использование надписей в нотации LaTeX

2 Лекция 8. Поверхности, образы, анимация

MESH

Изображение поверхностей осуществляется с помощью команд

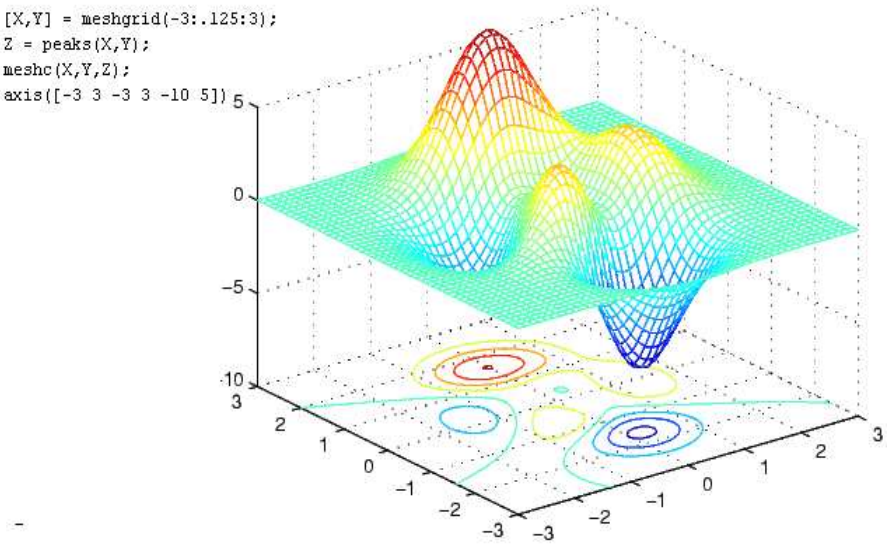
$$surf(Z), \quad surf(X,Y,Z), \quad surf(X,Y,Z) \tag{3.2.1}$$

$$mesh(Z), \quad mesh(X,Y,Z), \tag{3.2.2}$$

здесь в (3.2.3) поверхность строится с окрашиванием сетки, в качестве аргументов могут быть как пространственные координаты узлов самой поверхности Z, так и проекций X, Y на ось $Z = 0$; последний оператор позволяет визуализировать контурные линии (уровня) в плоскости $Z = 0$. Набор функция в (3.2.4) отличается еще и окрашиванием ячеек.

Следует заметить, что для построения поверхностей необходимо предварительно задать сетку, в координатах которой будет вычислен массив, описывающий поверхность. Это выполняется с помощью команды *meshgrid* – генерирования сетки. Далее приведенные примеры иллюстрируют некоторые приемы построения поверхностей:

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(X,Y,Z);
axis([-3 3 -3 3 -10 5])
```



Заметим, что в пакете существует мобильная возможность строить “неудобные” поверхности, которые в заданной области имеют устранимые

особенности. Далее предлагается пример кодов, с помощью которых строится поверхность $sin(x)/x$ (рисунок 3.3):

```
[X,Y] = meshgrid(-2*pi:.5:2*pi);
P = sqrt(X.^2 + Y.^2);
Z(P =0) = sin(P(P =0))./P(P =0);
Z(P==0) = 1;
mesh(X,Y,Z);
```

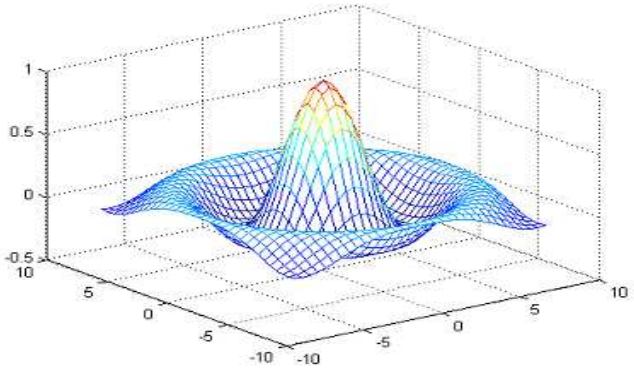


Рис. 3.3: График функции $sin(x)/x$.

IMAGE

В пакете широко используются такие графические объекты как изображения. Для этого предусмотрены функции их визуализации, редактирования, оптимального хранения, конвертирования и др.. Изображения являются не только средством отображения образов, но и объектом исследования таких пакетов, например, как WaveLet, NeuralNetwork, в которых изображения "очищаются" с помощью фильтров, распознаются, кодируются и т.п..

$$image(C), \quad image(x, y, C), \tag{3.2.3}$$

$$mesh(Z), \quad mesh(X, Y, Z), \tag{3.2.4}$$

здесь C – каждый элемент характеризует цвет прямоугольника, соответствующего элемента матрицы C (сегмента образа); $x, y,$ – двумерный вектор, указывающий размах соответствующих осей. На рисунке 3.4 приводится пример построения и визуализации образа–матрицы:

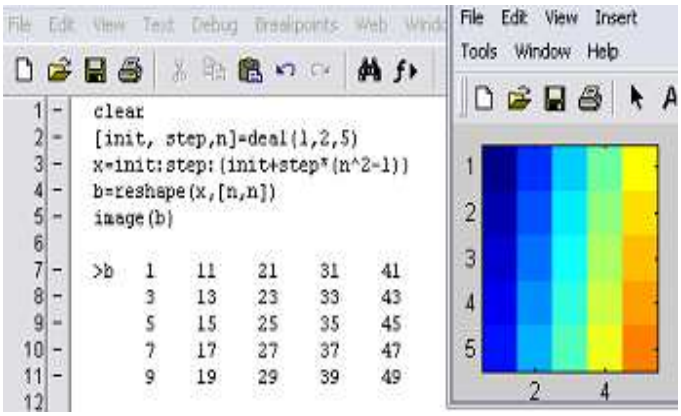


Рис. 3.4: Пример построения IMAGE

Помня о том, что $IMAGE$ – является структурой–элементом дескрипторной графики, приведем конструктор (команду высокого уровня), позволяющий оперативно изменять свойства полей структуры (образа):

$$image(x, y, C, 'PropertyName', PropertyValue, ...) \tag{3.2.5}$$

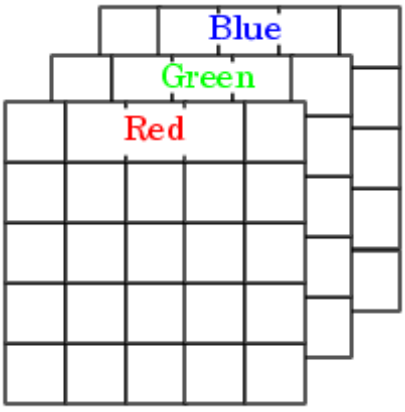


Рис. 3.5: Структура RGB

Цвет такого образа описывает многомерная матрица, каждый "лист" которой представляет интенсивность $RedGreenBlue$ в виде триплета (рису-

нок 3.5).

В MatLab поддерживаются возможности обработки файлов любого графического формата. Чтение осуществляется по команде

$$A = imread(filename, fmt),\ [X, map] = imread(filename, fmt) \quad (3.2.6)$$

некоторые форматы приведены в таблице 3.6. Информация о цвете хра-

'bmp'	'jpg' or 'jpeg'
'tif'	'pcx'
'hdf'	'png'

Рис. 3.6: Таблица форматов Image

нится попиксельно в виде триплетов.

Визуализация импортированного (редактируемого) объекта (фотографии) осуществляется с помощью команды

$$>> imshow(x, y, A, ...),\ \ >> imshow\ filename, \quad (3.2.7)$$

Анимация

Заниматься анимацией на серьезном уровне следует, используя пакеты $3d\ Studio\ MAX$, $Toom\ Boom\ Studio \ldots$, однако и $MatLab$ позволяет легко оживить некоторые процессы. Для этого необходимо создать серию графических объектов, описанными ранее средствами, которые представляют некоторую цепочку изменений исходного образа. Следует озаботиться, чтобы каждый элемент "цепочки" размещался в отдельном окне для этого для свойства поля $nextplot$ объекта оси cga следует задать $replacechildren'$, при этом содержимое текущего фрейма (поля с активным образом) сохраняется с помощью

$$F = getframe.$$

Анимация запускается по команде

$$>> movie(M), \quad >> movie(M, n), \tag{3.2.8}$$

noindent здесь M – совокупность промежуточных состояний типа F, n – количество повторений оживления картинок.

peaks

```
Z = peaks; surf(Z);
axis tight
set(gca,'nextplot','replacechildren');
% Record the movie
for j = 1:20
    surf(sin(2*pi*j/20)*Z,Z)
    F(j) = getframe;
end
% Play the movie twenty times
movie(F,20)
```

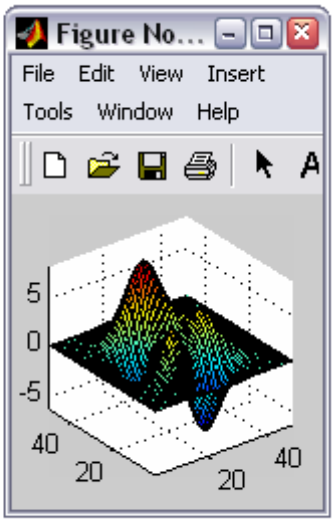


Рис. 3.7: Пример простейшей анимации

На рисунке 3.7 представлен код программы, демонстрирующий анимацию объекта *peaks*, координаты поверхности которого дискретизируются на 20 состояний суперпозицией Z и функции $\sin(2 * \pi * j/20)$, здесь j $(j = 1, \ldots, 20)$ характеризует текущий фрейм.

3 Лекция 9. Скрипты. Свойства отладчика

Скрипт-файлы

Скрипт–файлы это совокупность инструкций языка MatLab, записанных в текстовый файл с расширением $*.m$, идентификатор файла не может начинаться с цифры. Длина каждой строки не должна превышать 256 знаков, знак ... является индикатором продолжением инструкции на следующую строку.

Следует отметить, что скрипт–файлы исполняются с помощью интерпретатора. Если результат выполнения текущей инструкции не подавляется знаком (;), то в командном окне отображается этот результат. Формат отображаемого результата регулируется параметром $format$ объекта $Root, \mathrm{set}(0,\text{'format'})$.

Отображение результата может сочетаться с отображением в командной строке самой инструкции, это может быть полезно при программировании демо-версий учебных программ и достигается изменением свойства командного окна по команде *echo on*

Все переменные скрипта являются глобальными в рамках текущей сессии. Это обстоятельство может повлиять на переопределение переменных и некоторую неразбериху в случае повторных запусков скрипта. Исключить такие ситуации позволяет команда *clear*, использующаяся в качестве первой инструкции скрипт-файла.

Запуск скрипта, например, с именем $myscript.*$ осуществляется из командной строки или из текстового редактора, прилагающегося к пакету и поддерживающего функции отладчика.

Панель инструмента редактора, реализующая отладчик

Рассмотрим механизм функционирования отладчика, встроенного в пакет MatLab. Это текстовый редактор, поддерживающий функции отладки и позволяющий на протяжении всего этапа функционирования программы (от отладки до реализации) отслеживать ее состояние.

Рассмотрим назначение пиктограмм панели, указанные на рисунке. Соответственно номеру приводится интерпретация.

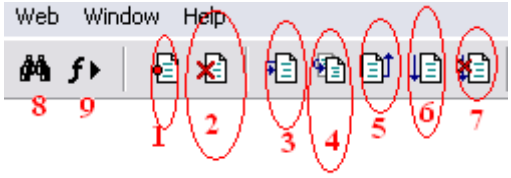
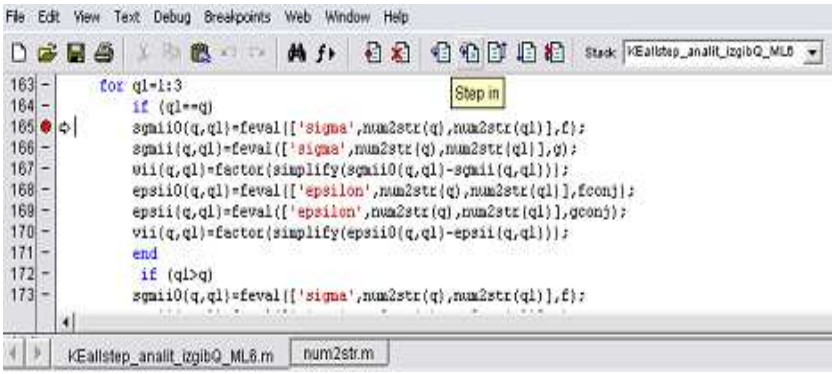


Рис. 3.8: Функции отладчика

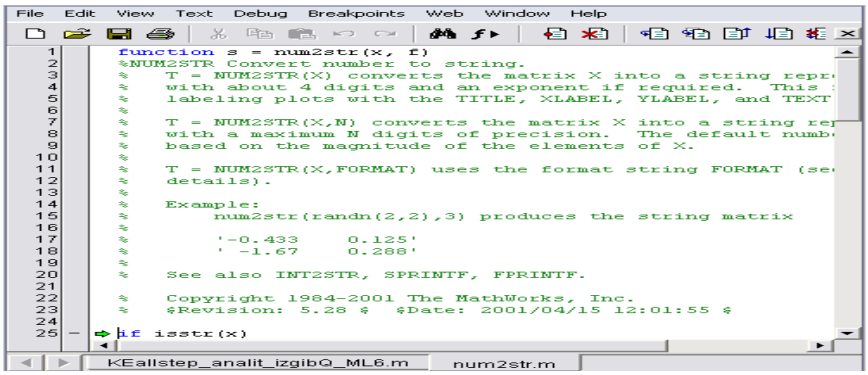
1. Установка точки прерывания осуществляется последовательным щелчком левой кнопкой мыши на пиктограмме, а затем в строке прерывания.
2. Убрать все точки прерывания, установленные в скрипте
3. Пошаговое выполнение кода.
4. Вход в подпрограмму из точки прерывания.
5. Возвращение пошаговое вверх по алгоритму.
6. Запуск программ на выполнение
7. Функция редактора контекстного поиска
8. Список активных процедур

Далее приводится вариант работы отладчика с функцией входа в процедуру из главной программы. Заметим, что установлена точка прерывания и

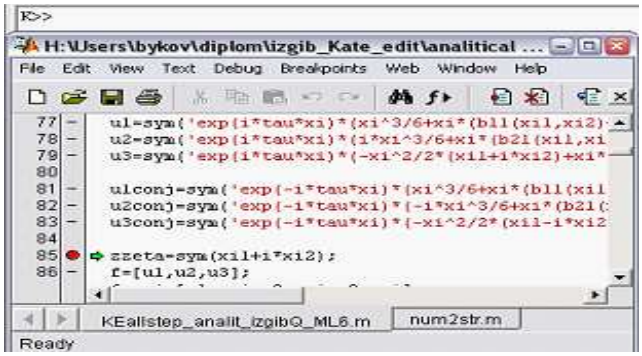
функция $feval(namefun, argument)$ равносильна выполнению процедуры $namefun(argument)$, в процессе формирования имени $namefun$, преобразовывается число в строку, в результате обращения к функции



к процедуре и выборе пиктограммы перехода в процедуру, программист оказывается с помощью редактора в тексте процедуры, в данном случае системной $(num2str)$.



Как только в процессе отладки выполнение инструкций достигает точки прерывания, командная строка претерпевает изменения и указывает K – индикацию режима отладки. В этом режиме командная строка по-прежнему готова к выполнению, возможно, новых инструкций программиста, которые помогут выявить некорректности программы. Более того, программист может выполнить некорректный или интересующий его код текста программы (непоследовательный), для этого требуется с помощью правой кнопки мыши выделить участок кода, находясь на выделенном участке, правой кнопкой активизировать выплывающее меню и выбрать



соответствующую выполнению опцию.

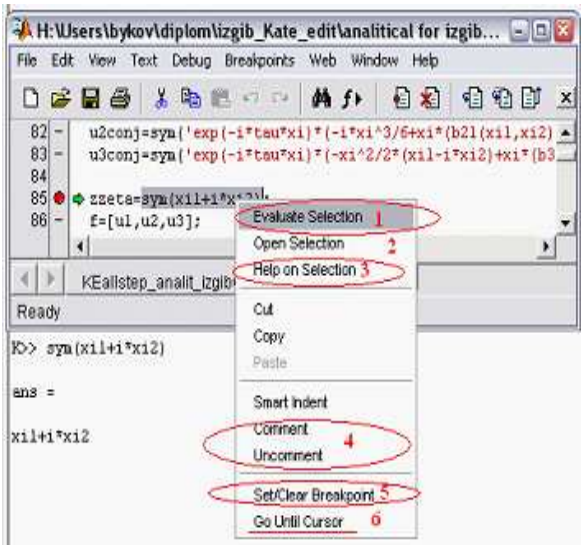


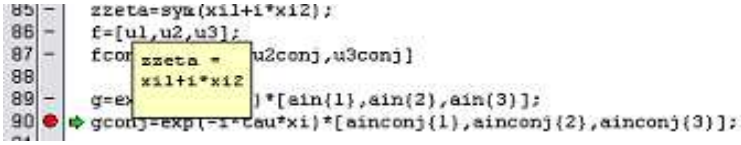
Рис. 3.9: Выплывающее меню

Их назначение в соответствии с указанной нумерацией представлено следующим перечнем:

1. выполнить выделенные инструкции, выражение
2. открыть m–файл с выделенным именем
3. открыть тематическую справку по выделенному идентификатору (имени процедуры, функции)
4. закомментировать выделенные строки; раскомментировать
5. добавить и ли убрать точку прерывания

6. осуществлять выполнение программы до строки, в которой размещен курсор.

Информация о состоянии выполняемой программы может быть получена еще более оперативно, для этого достаточно навести курсор на иденти-



фикатор переменной в той части программы, которая уже выполнялась, чтобы в выплывающем меню увидеть ее текущее значение.

$\circlearrowleft$. *Заметим, что такая визуализация не предусмотрена для массива ячеек, это осуществляется специальным методом celldisp.*

В соответствии с вышесказанным имеем все основания констатировать, что отладка программы с использованием возможностей отладчика – представляет собой функционально удобный процесс.

4 Лекция 10. Процедуры. Создание независимых приложений

Реализация программ с помощью интерпретатора отличается низкой скоростью выполнения. Альтернатива последовательному выполнению кодов скрипт-файлов – использование процедур. Компиляция кодов в виде процедур существенно ускоряет выполнение инструкций.
Процедура состоит из

- заголовка процедуры;
- справочной информации (о назначении процедуры и входных-выходных параметров);
- оператора декларирования глобальных переменных;
- Тела процедуры
- подпроцедур.

Заголовок процедуры: состоит из системного синтаксического определения $function$ и имеет вид:

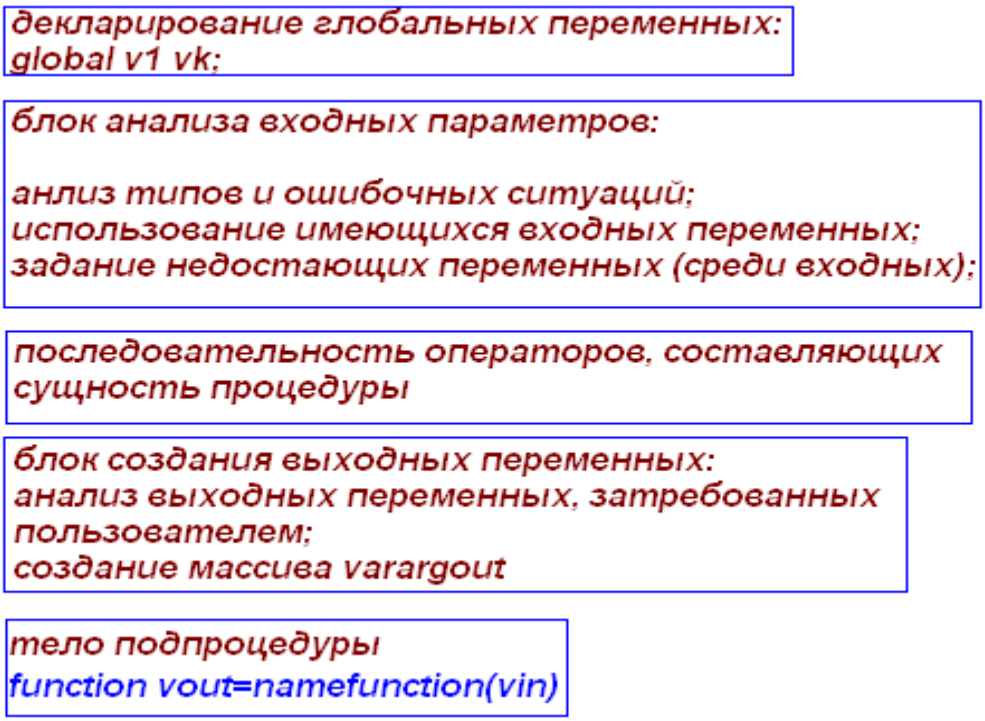
$$function\ [o_1, \ldots, o_n, varargout] = nf(i_1, \ldots, i_k, varargin),$$

здесь $o_1, \ldots, o_n$ – обязательные выходные параметры; $varargout$ – массив ячеек переменной длины, создаваемый **пользователем** для серии необязательных выходных параметров; $i_1, \ldots, i_k$ – обязательные входные параметры, могут отсутствовать; $varargin$ – массив ячеек переменной длины, создаваемый **системой** для серии необязательных входных параметров. С помощью функций $nargin$ определяется число входных параметров процедуры ($nargin = k + length(varargin)$); а $nargout$ – число выходных параметров ($nargout = n + length(varargout)$).

$\circlearrowleft$. *Заметим, что имя процедуры должно совпадать с именем файла* $nf.m$, *в котором записан код программы.*

Справочная информация процедуры пользователя должна располагаться после заголовка процедуры, и содержать информацию о назначении процедуры, формальных параметров. Вторая и третья строка доступны в качеств е справки по команде $helpname\_func$, $name\_func$ – имя процедуры; контекстный поиск, как и в общем случае, по команде $lookfor$, контекстная переменная выбирается так, чтобы процесс поиска ограничился разумным временем.

Структура и содержание процедуры схематически может быть представлено следующей таблицей: подробно анализ и обработка входных

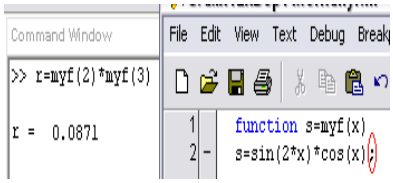


переменных, которые передаются системой при обращении к процедуре, будет проиллюстрирована на примере итерационных методов решения систем, равно как и приемы формирования списка выходных переменных, в том числе переменной длины.

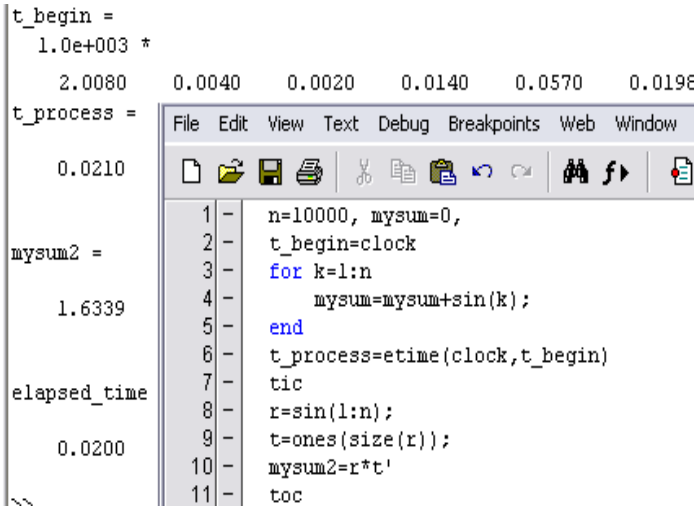
Функция от процедуры отличается тем, что она может быть частью

алгебраического выражения и это накладывает некоторые ограничения на ее синтаксис, а именно, функция должна иметь

- один выходной параметр
- идентификатор выходного параметра, совпадающий с последним исполняемым оператором
- подавляемые выводы всех исполняемых инструкций кода



Скорость выполнения процедур и функций существенно выше скрипт–файлов, в этом легко убедиться применяя процедуры контроля времени. На следующем рисунке приведен код программы, реализующий такой контроль, здесь $[clock, etime]$ первая процедура фиксирует текущее время, вто-



рая – определяет разность времен (аргументов функции).

Простейшей конструкцией процедуры является процедура *inline*, аргументом которой является строка, представляющая собой некоторое алгебраическое выражение, в качестве аргументов могут быть явно указаны аргументы выражения-строки. В противном случае их следует определять

с помощью функции $argnames$: $args = argnames(inline('expression'))$ здесь $args$ – массив ячеек, содержащий все аргументы $expression$ и все элементы $args\{:\}$ – строки.

Создание независимых приложений и библиотек

При инсталляции пакета необходимо предусмотреть предварительную установку пакетов C, C++.

Для получения независимо исполняемого приложения, включающего графику, следует использовать версии C, начиная с 6-й.

На компьютере могут находиться разные версии C, для выбора конкретной, участвующей в компиляции и получении объектных модулей, следует воспользоваться процедурой mcc:

mbuild -setup в режиме диалога следует выбрать требуемый компилятор, на его основе создать приложение, объединяющее несколько файлов.

mcc -x mainfile – генерирует исполняемый mex–файл, а также файл–dll, аналог библиотеки Windows.

mcc -B sglcpp mainfile filefunc1 filefunc2 filefunc3 – здесь опция -B указывает на установку создания приложения mainfile.exe; опция sglcpp – использование графических C++ библиотек и создания C++ кодов процедур.

mcc -B sgl mainfile filefunc1 filefunc2 filefunc3 – аналогично предыдущему, но с использованием C-компилятора.

5 Лекция 11. Элементы управления. Создание интерфейса пользователя

Предком любого элемента управления является графический объект типа *figure*. Объекты пользовательского интерфейса являются структурой, поля которой описывают его фактические свойства.Параметры, которые соответствуют размерам этих объектов, задаются в тех единицах измерения, которые заданы в поле *unit* объекта *figure* с текущим дескриптором *gcf*. Размеры монитора определяются командой $>> get(0,'screensize')$

Кнопка

Создание элемента управления произвольного типа осуществляется с помощью конструктора

$$uicontrol(hparent,'style','pushbutton','position',\ldots$$

$$[x, y, h_x, h_y],'string','buttontext')$$

Природа элемента управления определяется свойством поля *Style*. В рассматриваемом случае создается кнопка, которая отжимается, но действий при этом не происходит. Кнопка подписывается текстом *buttontext*, являющегося значением свойства $'string'$. Она расположена левой нижней вершиной в позиции x, y относительно левого нижнего угла экрана, $h_x,\ h_y$ – длины сторон.

Текст

Текст в меню пользователя создается с помощью того же конструктора созданием элемента иного типа. Текст может задаваться с редактированием (edit) и без (text). В первом случае поле структуры *string* не задается и

текст может вводиться с клавиатуры, во втором – текст является поясняющей надписью и не модифицируется.

$$uicontrol(h,'Style','edit',)$$

$$uicontrol(h,'Style','text','string','mytext')$$

$$uicontrol(h,'Style','edit','Position',[x,y,h_xh_y],...$$

$$'BackgroundColor','white','HorizontalAlignment','left');$$

Существенно, что значение поля – строка, если с его помощью передается числовая информация, то ее требуется преобразовать к типу *numeric* с помощью конвертера:

$$str2num(get(he,'string')).$$

↺ . *Заметим, что BackgroundColor не задается для кнопок, выравнивание текста по умолчанию происходит по центру.*

Индикация признака

"Чекбокс" – элемент управления, означающий наличие или отсутствие предусмотренной логикой задачи опции,

$$cb = uicontrol(hf,'Style','checkbox','Position',[x_l\ y_l\ h_x\ h_y],\cdots$$

$$'String','someoption','HorizontalAlignment','top');$$

он состоит из окошка индикации и текста, который объясняет изменение свойств, например, это может быть признак накопления графиков в окне *axes* при наличии "галочки"либо создание каждый раз нового графика в случае ее отсутствия. Обязательные параметры: *position*; желательный – *string*; отсутствие выравнивания – по центру. Получить значение признака можно из поля *value*

$$> ind = get(cb,'value')$$

сконструированной структуры.

Список

Список – элемент управления, который позволяет из предлагаемого на поле интерфейса набора поименованных элементов (любой природы) с помощью курсора или мыши интерактивно выбрать необходимый. В отличие от всех, ранее введенных элементов, параметром поля *string* является массив ячеек из строковых переменных.

$$hl = uicontrol(hf,'Style','listbox','Position',[x_l\ y_l\ h_x\ h_y],'BackGroundColor',\cdots$$

$$'magenta',[a,\ b,\ c],'String',str1,str2,\cdots,strn,'HorizontalAlignment','right');$$

С помощью конструктора создается список с дескриптором hl. В случае, если не задавать *position*, будет доступно только одно имя, остальные – с помощью строки прокрутки. Заметим, что параметры цвета могут также быть заданы в виде вектора из трех констант ($\in (0,1)$), соответствующих концентрации цвета из палитры RGB;

Если параметры цвета поля списка потребовать неотличимыми от поля $figure$, следует воспользоваться двумя командами:

$$> colorlist = get(gcf,'color')$$

$$> set(hl,'color',colorlist)$$

В результате выбора курсором требуемой функции параметр поля *value* содержит информацию об индексе элемента массива ячеек всех возможных

строк; получить его можно с помощью команды

$$> x = 0 : 0.1 : pi > ind = get(hl,' value')$$

$$> name\_cell\_string = get(hl,' string')$$

$$> our func = name\_cell\_stringind$$

$$> feval(our\_func)$$

Ползунок-слайдер

Если *checkbox* следует рассматривать как "интерактивный" признак целого типа, то ползунок – некоторый масштабирующий вещественный коэффициент, без которого не обойтись. К уже известным поименованным полям следует добавить $'SliderStep'$, который позволяет соотносить реальные изменения положения бегунка с величиной масштабирующего множителя.

$$> hs = uicontrol(hf,' Style',' slider',' SliderStep', 0.1);$$

Получить значение признака можно из поля *value* нашего "движка":

$$> coeff = get(cb,' value'),$$

при этом тип переменной $coeff$ является вещественным.

Графическое окно

Графическое окно служит для визуализации графиков и различных видимых эффектов. Создается тем же самым конструктором, с учетом требуемого типа объекта. Создаваемый объект является элементом дескрипторной графики *axes* с уже известными свойствами. Родительским графическим элементом по-прежнему является элемент типа $figure$.

$$ha = axes('Parent', hF1,' Color', [111],' Units',' normalized',$$

$$'Position', [0.050.0511],' FontSize', 6,' Fontname',' TimesNewRoman');$$

Здесь используется нормализованный способ задания размеров, когда все поле $figure$ имеет размеры [0 0 1 1]

Поиск объектов с заданными свойствами

Поиск объектов с заданными свойствами – весьма полезная задача. Положим, жизнь изменилась, и все серое срочно нужно изменить на розовое. понятно, что следует определить все дескрипторы, свойства параметров цвета которых – серый.

$$list\_descr = findobj(gcf,'color','grey')$$

В результате в $list\_descr$ содержатся дескрипторы нужных объектов.

Для объединения разнородных объектов или свойств в общий класс, полезно используется поле Tag–ярлык, которому присваивается любое имя $NameTag$, в результате такие объекты легко модифицировать одномоментно.

Сопряжение элементов управления

Создание всех перечисленных элементов управления целесообразно поместить в *.m–файл. Однако обработку результатов интерактивного воздействия на них следует обрабатывать в файлах инициирующих выполнение тех или иных алгоритмов.

Для этого, например, при создании кнопки (или впоследствии) следует добавить поле $'Callback'$, параметром которого будет имя функции, которую следует выполнить.

Предположим, по нажатии одной из кнопок следует стереть все графики в окне $axes$ с дескриптором ha. Для этого следует после создания (или при создании) добавить параметр обратной связи, а именно

$$set(ha,'callback','clear\_axes')$$

и в файле, инициирующем очистку окна осей описать как глобальную переменную

$$> global\ ha$$

$$> axes(ha)$$

$$> cla(ha)$$

в результате возможно реализовать многократное использование поля объекта $AXES$.

6 *Вопросы для самопроверки:*

Скрипт–файлы

- обрабатываются интерпретатором и выполняются в интерактивном режиме?
- выполняются в интерактивном режиме, отображающем в командном окне как инструкцию, так и результат ее выполнения
- имеют расширение $*.m$, $*.mat$?
- имеют имена, идентификаторы которых могут начинаться как с цифры, так и с буквы?

Процедуры

- должны иметь имена, совпадающие с именем m–файла?
- могут иметь число входных формальных параметров переменной длины?
- должны всегда содержать блок инструкций, формирующий масив ячеек $varagout$?

Uicontrol

- конструктор всех элементов управления GUI?
- регулирует размер свойствами поля *position*?
- осуществляет обратную связь (выполнение действия) с помощью m–файла с именем, указанным в поле $CALLBACK$ или GO?
- задает тип элемента управления с помощью свойства поля $Style$ или $Type$?

Такие графические объекты как

- IMAGE описывают свойство поля $Color$ массивом *double* $[\alpha, \beta, \gamma]$, $(\alpha, \beta, \gamma \in [0, 1])$. Почему?
- *Animation* требуют изменения предварительного свойства поля $'nextplot'$ осей gca?
- *Line*–объект дескрипторной графики может быть получен с помощью изображения линий командой *ezplot*?

Проектное задание

Заключается в построении многомодульной программы с программированием процедур, имеющих входные и выходные параметры переменной длины. Программирование головных файлов следует выполнить в виде процедур без параметров. Создать независимое приложение.

Глава 4

Схемы эффективного программирования

Идеология MatLab связана с пополнением модулей, охватывающих все новые предметные области. Чем более подготовленный пользователь пользуется пакетом, тем более эффективна его работа. Очевидно, что студентам младших курсов, не имеющие базовых знаний по теории вероятности, статистике, решении оптимизационных задач, будет весьма затруднительно работах в рамках модулей статистика, нейронные сети, решения дифференциальных уравнений в частных производных. в разделе приведены также приемы эффективного программирования, рассмотренные на базе программирования итерационных методов решения СЛАУ и метода Ньютона – систем нелинейных уравнений.

1 Лекция 12. Приемы программирования решения нелинейных систем высоких порядков. Сочетание методов SYM и CHAR

Системы нелинейных уравнений малых порядков решаются в пакете с использованием ядра *Maple*. Аргументы функции решения систем *solve* – уравнения, которые следует выбирать как массив строк или символьные

выражения:

$$solve('eqn1','eqn2',...,'eqnN'),$$

$$solve('eqn1','eqn2',...,'eqnN','var1,var2,...,varN')$$

здесь $'eqn1','eqn2',...,'eqnN'$ – уравнения; $'var1,var2,...,varN'$ – искомые неизвестные. Результат выполнения – структуры, элементами которых являются символьные значения. В рассматриваемом случае решение $ans.var1$, $ans.var2$, ..., $ans.varN$ преобразуется к типу *double* с помощью цепочки преобразований $str2num(char(ans.varN(i)))$, здесь i – номер решения.

Метод Ньютона решения нелинейных систем

Пусть требуется решить систему $\begin{cases} f_1(x_1,x_2,\ldots,x_n)=0 \\ \ldots \\ f_n(x_1,x_2,\ldots,x_n)=0 \end{cases}$ n – го порядка, заданную нелинейными уравнениями $f_i(P)=0$, где $P=(x_1,x_2,\ldots,x_n)\in \Re^n$, $i=1,\ldots,n$.

Вводя обозначение $F(P)=\begin{bmatrix} f_1(x_1,x_2,\ldots,x_n) \\ \ldots \\ f_n(x_1,x_2,\ldots,x_n) \end{bmatrix}$, алгоритм решения системы сводится к следующим шагам:

1. Задать нулевое приближение (текущее)

2. Вычислить значения вектор-функции в текущей точке:

$$F(P^k)=\begin{bmatrix} f_1(x_1^k,x_2^k,\ldots,x_n^k) \\ \ldots \\ f_n(x_1^k,x_2^k,\ldots,x_n^k) \end{bmatrix}$$

3. Вычислить значение матрицы Якоби в текущей точке:

$$J(P^k) = \begin{pmatrix} \frac{\partial f_1(P^k)}{\partial x_1} & \cdots & \frac{\partial f_1(P^k)}{\partial x_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_n(P^k)}{\partial x_1} & \cdots & \frac{\partial f_n(P^k)}{\partial x_n} \end{pmatrix}$$

4. Решить линейную систему относительно приращения аргумента:

$$J(P^k)\Delta P = -F(P^k)$$

5. Вычислить текущеее приближения решения системы:

$$P^k = P^k + \Delta P$$

6. Оценить точность решения или количество итераций

7. Достигнуто решение или повторить шаги 1–5

Приемы эффективного программирования. Метод Ньютона

Приведем пример реализации такой программы, для системы третьего порядка, алгоритм которой рассчитан на произвольное число n уравнений. Формальный параметр F, передающий уравнения процедуре – массив яче-

```
1  clear; clc;
2
3  n=2; eps=1e-5; kmax=10; X0=[1 1 1]';
4  F=cell(n,1); X=cell(n,1);
5  F{1}='x1+y+z^3-6';
6  F{2}='x1-y^2+z-2';
7  F{3}='x1-z+y';
8
9
10 [X,k,eps2,N]=Newton(F,X0,eps,kmax)
11
12
```

Рис. 4.1: Пример скрипта для метода Ньютона

ек. Распознавание неизвестных здесь реализовано с помощью $findsym$. В ML выше шестой версии результат такой операции – массив ячеек, каждый элемент которого содержит неизвестные уравнений. В нашем случае код реализован для ML6, и результат - строка, неизвестные в которой отделены запятой и пробелом. Программисту требуется “накопить” эти неизвестные, обработав строку результата. Предлагаемый код описывает эту процедуру.

```
function [Xnew,k,k_real,Nev]=Newton(F,X0,eps,kmax)
%Метод Ньютона решени_ нелинейнЫх систем уравнений

n=size(F); n=n(1);
Xs=findsym(sym(F(1)));

%Преобразование строки в массив из переменнЫх
k=0; start_sym=1;
for i=1:length(Xs)
    if (Xs(i)==',')
        end_sym=i-1;
        k=k+1;
        X(k)=Xs(start_sym:end_sym);
        start_sym=end_sym+3;
    end
    if (i==length(Xs))
        k=k+1;
        X(k)=Xs(end_sym+3:i);
    end
end
```

В результате массив ячеек X хранит неизвестные. Такого же результата можно достигнуть, воспользовавшись функцией $Inline$. Предложенный алгоритм позволяет вычислить Якобиан. Здесь (коды рисунка 4.2) F, J –

```

%Построение матрицу _коби
for i=1:n
    for j=1:n
        J(i,j)=diff(F(i),X(j));
    end
end
```

символьные объекты; F_chisl, J_chisl – объекты $numeric$. Команда $subs$ позволяет найти в уравнениях неизвестные, в их символьные аналоги подставить числовые значения и получить Якобиан, а также и вектор–столбец как объект $double$. Решение линейной системы сводится к обычному левому

```
29 %Подготовка массива _чеек дл_ подстановки конкретного значени_
30 for i=1:n F_chisl(i)=sym(F(i)); end
31 F_chisl=F_chisl';
32 F_chisl2=F_chisl;
33
34 %Метод Ньютона
35 R=100+eps; k=0; Xnew=zeros(n,1);
36 while (R>eps)&(k<kmax)
37     %подстановка в матрицу конкретного значен.
38     J_chisl=J;
39     for i=1:n J_chisl=subs(J_chisl,X(i),X0(i)); end
40     Ji=inv(J_chisl);
41
42     F_chisl=F_chisl2;
43     for i=1:n F_chisl=subs(F_chisl,X(i),X0(i)); end
44
```

Рис. 4.2: Код, реализующий вычисление якобиана в символьном поле и *numeric*

```
45     %нахождение следущего приближен.
46     Xnew=-Ji*F_chisl+X0;
47
48     %нахождение погрешности
49     R=abs(max(abs(Xnew))-max(abs(X0)));
50
51     X0=Xnew;
52     k=k+1;
53 end
54
55 k_real=R;
56 Nev=F_chisl;
```

Рис. 4.3: Решение СЛАУ. Условие сходимости

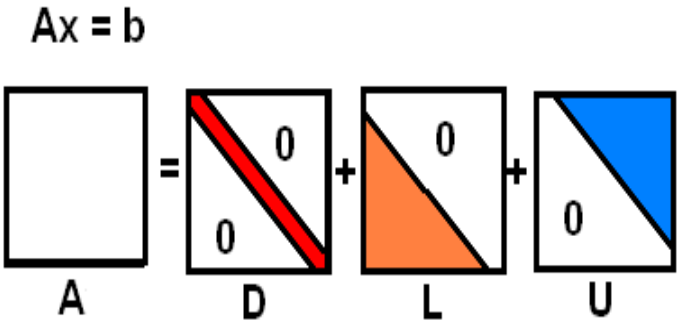
делению, которое реализовано по методу Гаусса. В случае неудовлетворительной точности решения (проверяются условия сходимости рисунок (4.3) шаги 1–5 повторяются (). Следует отметить, что в качестве входных параметров выбирается система уравнений, заданная массивом ячеек строк или символьных объектов, нулевое приближение, точность и максимально число допустимых итераций. Выходные параметры – решение системы, достигнутая точность, количество выполненных итераций. Следует отметить, что выход из алгоритма может осуществиться по одному из перечисленных условий. Либо достигается точность и выводится номер текущей итерации,

либо по недостигнутой точности алгоритм заканчивается максимальной итерацией.

2 Лекция 13. Итерационные методы. Эффективное программирование

Как уже выше говорилось, в пакете реализованы как прямые, так и итерационные методы линейной алгебры решения алгебраических уравнений. Кроме того, имеются библиотеки процедур, с помощью которых осуществляется факторизация матриц, в том числе симметричных. В настоящем курсе решаются задачи не только освоения библиотечных функций, но и написание пользовательских процедур, основанных на матричном мышлении.

Для этого рассмотрим итерационные методы решениия СЛАУ Зейделя и Якоби. Матрицу системы можно представить в виде суммы трех матриц: нижне-, верхнетреугольной и диагональной. Сущность различия методов следует из предлагаемой схемы:



Выбор матриц M, N приводит к той или иной модификации метода Якоби, а также Зейделя:

(a,b) - Якоби: $N = L + D\ (U + D),\ M = -U\ (-L)$

(c) - Зейделя: $N = D\ \ M = -(U + L)$

Используя в MatLab функции выбора диагональной матрицы, верхнетреугольной и нижентреугольной матриц, соответственно,

$$D = diag(diag(A)), U = triu(A, 1), \qquad L = tril(A, -1))$$

легко представить алгоритм, получения решения, а именно:

$$N x = (M x + b)$$

$$x = N^{-1}(M x + b)$$

$$x_{k+1} = N^{-1}(M x_k + b)$$

$$x_{k+1} = N^{-1}(M N^{-1}(M \dots N^{-1}(M x_0 + b) \dots))$$

$$\text{eig}(N^{-1} M)$$

Очевидно, что условие сходимости предлагаемого метода определяется собственными значениями матрицы R:

$R = N^{-1}M, \quad \rho(R) < 1;$

λ_i – **с.з.** R, V **- скорость сходимости**

$V \downarrow 0$ если $\exists\, j: \ \lambda_j \sim 1$

$V \uparrow \infty$ если $max_j(\lambda_j) \downarrow 0$

которые должны быть все меньшими единицы. Проверку этого условия необходимо реализовать при программировании, и обеспечить следующий перечень входных-выходных параметров:

Входные параметры обязательные: матрица системы: A, b - вектор правой части;

Выходные параметры обязательные: x – вектор-решение;

Входные параметры необязательные: x_0 – начальное приближение; eps – точность сходимости; n_{max} – максимальное число итераций;

Выходные параметры необязательные: eps_{real} – достигнутая точность; n_{real} – достигнутое число итераций.

3 Лекция 14. Пакет решения дифференциальных уравнений PDETool.

Оболочка *PDEtool* предназначена для решения ДУ второго порядка в частных производных:

$$Au_{xx} + Bu_{xy} + Cu_{yy} = f(x, y, u, u_x, u_y)$$

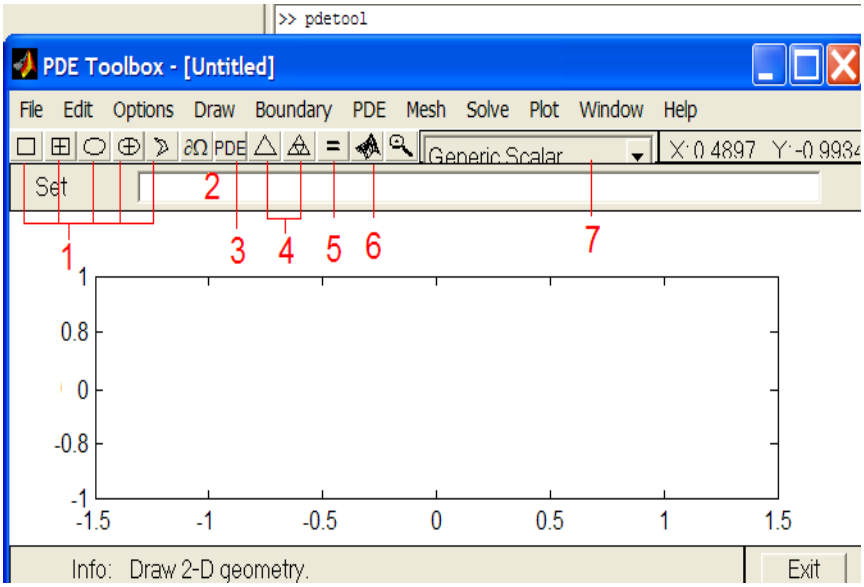
здесь A, B, C – квазилинейные множители; справедлива следующая классификация:

1. $B^2 - 4AC < 0$ – уравнения эллиптического типа
2. $B^2 - 4AC = 0$ – уравнения параболического типа
3. $B^2 - 4AC > 0$ – уравнения гиперболического типа

- *уравнения параболического типа*: теплопроводности: $u_y = c^2 u_{xx}$, ($y \equiv t$)
- *уравнения гиперболического типа*: волновое уравнение: $u_{yy} = c^2 u_{xx}$
- *уравнения эллиптического типа*: $\nabla^2 u = 0$ – уравнение Лапласа; $\nabla^2 u = g(x, y)$ – уравнение Пуассона; $\nabla^2 u + f(x, y)u = g(x, y)$ – уравнение Гельмгольца)

Запуск оболочки осуществляется из командного окна, строки команд, вызовом программы *pdetool*. Оболочка имеет дружественный интерфейс, с указанным назначением пиктограмм:

1. Геометрические примитивы
2. Выбор дифференциального уравнения в частных производных (эллиптического, параболического, гиперболического типа и на собственные значения)



3. Топология тела (отверстия); сопряжение тел;

4. Триангуляция области (уточнение сетки)

5. Запуск решателя

6. Выбор опций и визуализация полученных результатов

7. Выбор предметной области

Решение уравнений следует начинать с выбора типа уравнения и определения его коэффициентов. Это осуществляется с помощью индикатора параметров панели, с помощью геометрических примитивов, а также возможности редактирования строится геометрия и задаются граничные условия.

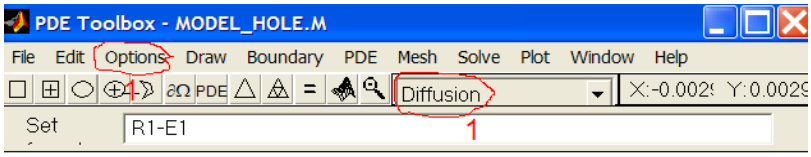
$$u_{xx} + u_{yy} = 1,$$

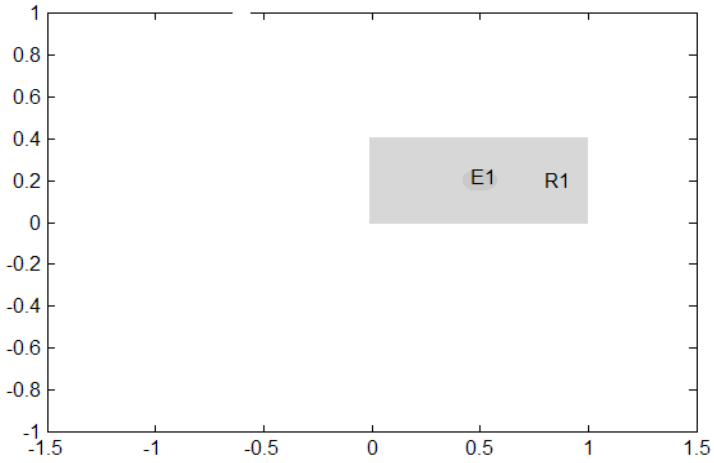
$$(u(x,y), \quad (x,y) \in \Omega = [0,1] \times [0,0.4];$$

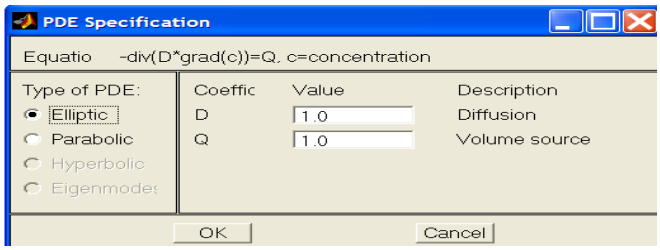
$$u|_{x=0,x=1} = (y-1)^2;$$

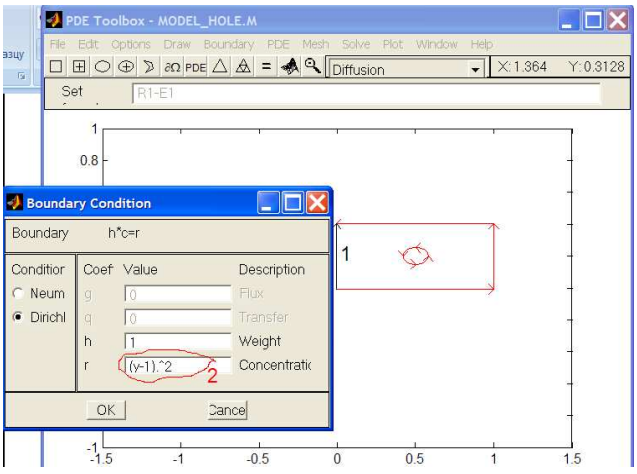
покажем, как удовлетворить указанным граничным уловиям для $y = 0, y = 0.4$:

$$u|_{y=0,y=0.4} = x^2;$$


PDE Toolbox - MODEL_HOLE.M
File Edit Options Draw Boundary PDE Mesh Solve Plot Window Help
Diffusion
Set
R1-E1
1

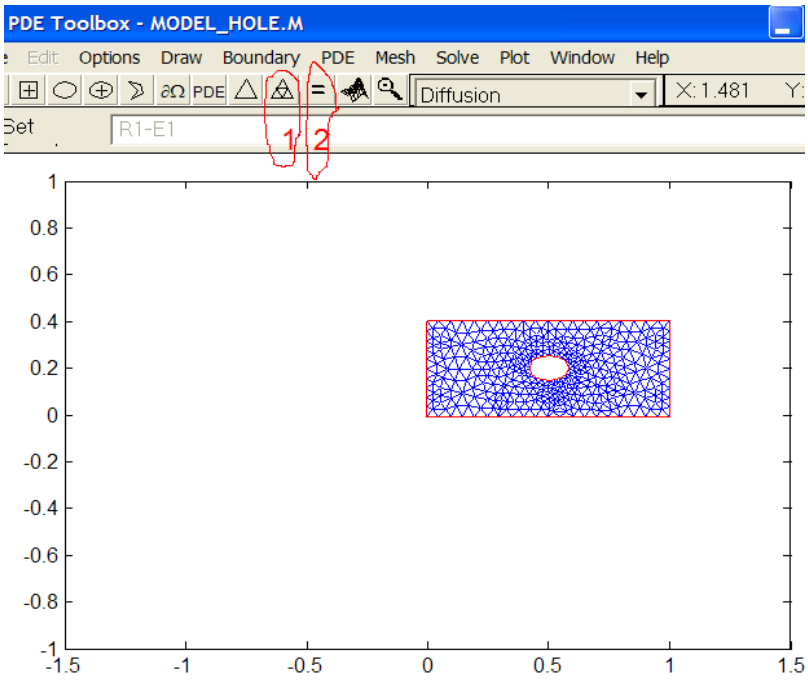

E1
R1


PDE Specification
Equatio -div(D*grad(c))=Q, c=concentration
Type of PDE:
Elliptic
Parabolic
Hyperbolic
Eigenmodes
Coeffic Value Description
D 1.0 Diffusion
Q 1.0 Volume source
OK
Cancel


PDE Toolbox - MODEL_HOLE.M
Boundary Condition
Boundary h*c=r
Condition Coef Value Description
Neum
Dirichl
g Flux
q Transfer
h 1 Weight
r (y-1).^2 Concentratic
OK
Cancel
1
2

заметим, что все операции являются векторными (.^)

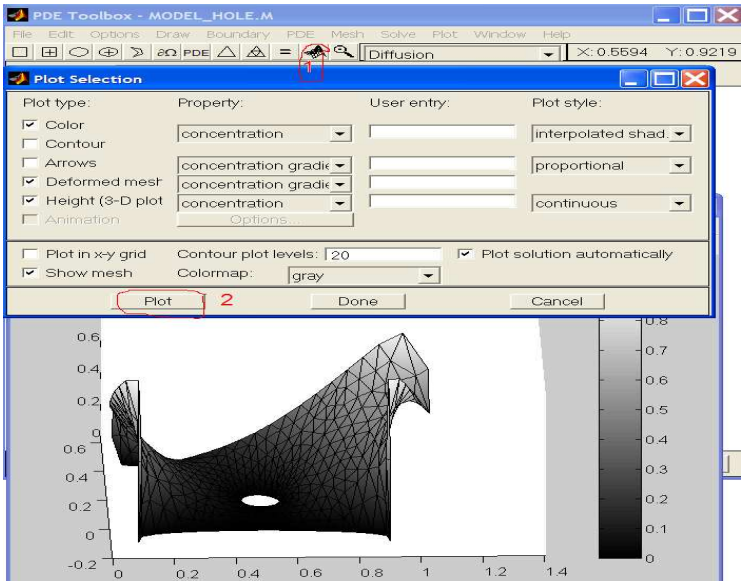
аналогично граничное условие задается и для $y = 0.4$; Далее выполняется триангуляция, в случае надобности в варианте сгущения. На панели указано, как запустить разбиение на элементы *1 2*–инициировать решение. Распределение поля скаляров указано с помощью цветовой шкалы. иные параметры визуализации можно выбрать указанным на панели постпроцессорной обработки способом.

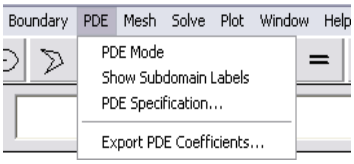


Сохранить модель, а также все его параметры (коэффициенты, триангуляцию или решение)

- всю модель — *save as model*, — создать *m*-файл модели, отражающий все этапы процесса *save as model*,
- Переменные среды, описывающие уравнения, граничные условия, разбиения, решение, экспортируются с помощью переменных с указанными идентификаторами типа *double*.

Здесь приведены некоторые "прозрачные" коды модели, задания области изменения переменных, построения геометрии.





C:\MATLAB6p1\work\model_hole.m

File Edit View Text Debug Breakpoints Web Window Help

Stack: Base

```
1   function pdemodel
2 -  [pde_fig,ax]=pdeinit;
3 -  pdetool('appl_cb',10);
4 -  set(ax,'DataAspectRatio',[1 1 1]);
5 -  set(ax,'PlotBoxAspectRatio',[1.5 1 1]);
6 -  set(ax,'XLim',[-1.5 1.5]);
7 -  set(ax,'YLim',[-1 1]);
8 -  set(ax,'XTickMode','auto');
9 -  set(ax,'YTickMode','auto');
10
11  % Geometry description:
12 -  pderect([0.9972105997210596 3 -1.0055788005578801 0.39051603905160381 -0.40167364016736418],'R1');
13 -  pdeellip(0.0041841004184099972,0.0055788005578798483,0.13668061366806139,0.097629009762900898,...
14  0,'E1');
15 -  set(findobj(get(pde_fig,'Children'),'Tag','PDEEval'),'String','R1-E1')
16
```

Глава 5

Модули Statistics toolbox и NeuralNetworks

1 Лекция 15. Статистическое моделирование в Statistics toolbox

Статистическая обработка данных с помощью пакета MatLab начинается с формализации и сохранения информации о статистически обследованном объекте. В случае, если анализируются (n) свойств объекта $\{O_1, O_2, \cdots, O_n\}$, наблюдаемых в нескольких экспериментах, то результат такого обследования представляется в виде таблицы (матрицы) *объект-свойство*. В такой модели элементы i-й строки матрицы – зарегистрированные свойства в i-м эксперименте. В случае, когда элементы матрицы выражают меру сходства или различия между анализируемыми свойствами (объектами) природа элементов матрицы становится иной, и здесь речь идет о модели *парных сравнений*.

$$, \text{“}''MatLab, Statistics, , .$$

пакет содердит тематические блоки функци и справок об их назначении: (1) – сводная информация о распределении случайных величин (подробно пункты 2-6); функция распределения вероятности (F(x)) в случае непре-

The Statistics Toolbox's Main Categories of Functions	
Probability Distributions	Parameter Estimation 1
	Cumulative Distribution Functions (cdf) 2
	Probability Density Functions (pdf) 3
	Inverse Cumulative Distribution Functions 4
	Random Number Generators 5
	Moments of Distribution Functions 6
Descriptive Statistics	Descriptive statistics for data samples 7
Statistical Plotting	Statistical plots 8
Statistical Process Control	Statistical Process Control
Cluster Analysis	Grouping items with similar characteristics into clusters
Linear Models	Fitting linear models to data 9
Nonlinear Regression	Fitting nonlinear regression models
Design of Experiments	Design of Experiments
Principal Components Analysis	Principal Components Analysis
Hypothesis Tests	Statistical tests of hypotheses 10
File I/O	Reading data from and writing data to operating-system files
Demonstrations	Demonstrations
Data	Data for examples

рывного распределения или накопленная частота для дискретных случайных величин (СВ); (2) – функция плотности вероятности (f(x)) в случае непрерывного распределения; (3) – Обратная функция распределения вероятности $V(p)$: СВ $X_0 = F^{-1}(p) = V(p)$, для заданного закона распределения F(x); – генератор случайных величин, можно использовать для случайного формирования выборок с помощью функции $V(x)$.

Началом любого статистического исследования является вычисление описательных статистик, это: (5) – моменты как мера оценки распределения СВ, дисперсия – разброса; ассиметрия – симметрии распределения относительно среднего; скошенность – оценка степени случайности процесса (тенденция к островершинности – процесс становится более детерминированным); (6) – меры среднего (среднее, медиана, размах, мода, статистики. В пункте (7) – представлены функции визуализация распределения СВ: графики распределений, полигон частот, гистограмма, диаграмма

рассеяния и т.д.; (8) – модуль (подбора) фиттинга распределений, позволяющий по имеющимся данным, определить параметры непрерывных законов распределения, которые по МНК являются максимально подходящими. Модуль (9) содержит список функций проверки статистических гипотез,

Parameter Estimation	
betafit	Parameter estimation for the beta distribution
betalike	Beta log-likelihood function
binofit	Parameter estimation for the binomial distribution
expfit	Parameter estimation for the exponential distribution
gamfit	Parameter estimation for the gamma distribution
normfit	Parameter estimation for the normal distribution
poissfit	Parameter estimation for the Poisson distribution
unifit	Parameter estimation for the uniform distribution

Рис. 5.1: Функции распределения вероятностей

например, о среднем, медиане, законе распределения, дисперсии и т.п.

Cumulative Distribution Functions (cdf)	
betacdf	Beta cdf
binocdf	Binomial cdf
chi2cdf	Chi-square cdf
expcdf	Exponential cdf
fcdf	F cdf
gamcdf	Gamma cdf
geocdf	Geometric cdf
normcdf	Normal (Gaussian) cdf
poisscdf	Poisson cdf
raylcdf	Rayleigh cdf
tcdf	Student's t cdf
unidcdf	Discrete uniform cdf
unifcdf	Continuous uniform cdf
weibcdf	Weibull cdf

Probability Density Functions (pdf)	
betapdf	Beta pdf
binopdf	Binomial pdf
chi2pdf	Chi-square pdf
exppdf	Exponential pdf
fpdf	F pdf
gampdf	Gamma pdf
geopdf	Geometric pdf
hygepdf	Hypergeometric pdf
normpdf	Normal (Gaussian) pdf
poisspdf	Poisson pdf
raylpdf	Rayleigh pdf
tpdf	Student's t pdf
unidpdf	Discrete uniform pdf
unifpdf	Continuous uniform pdf
weibpdf	Weibull pdf

Так функции фиттинга (подбора) распределения вероятности следующей таблицы (рисунок 5.1) имеют смысловые имена, соответствующие нормаль-

ному распределению, хи-квадрат, Стьюдента и т.д, равно как и функции плотности вероятности

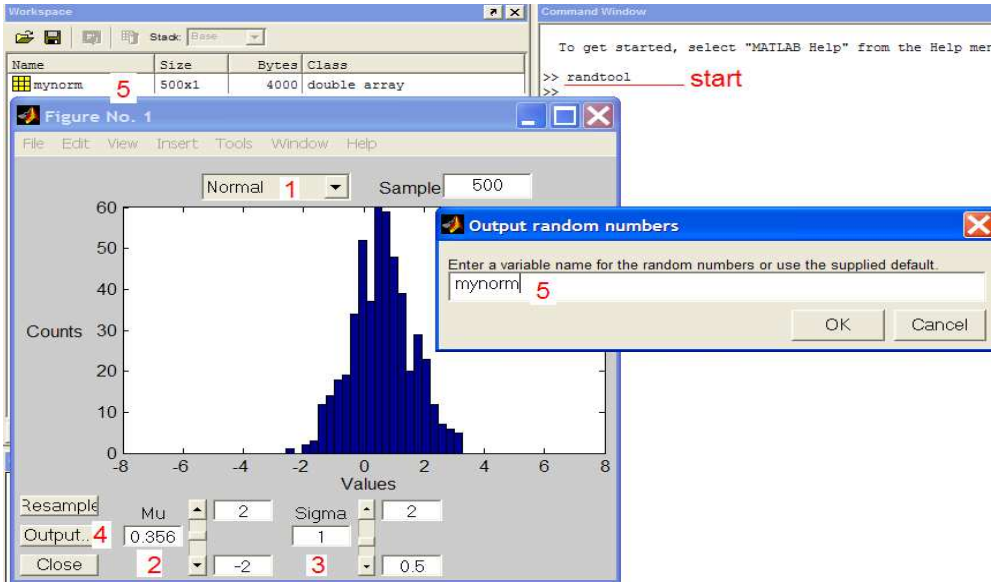
На рисунке 5.2 приведена таблица, в которой

Descriptive Statistics	
corrcoef	Correlation coefficients (in MATLAB) 1
cov	Covariance matrix (in MATLAB) 2
kurtosis	Sample kurtosis 3
mean	Arithmetic average (in MATLAB) 4
median	50th percentile (in MATLAB) 5
moment	Central moments of all orders 6
nanmax	Maximum ignoring missing data 7
nanmean	Average ignoring missing data 8
range	Sample range 9
skewness	Sample skewness 10
std	Standard deviation (in MATLAB) 11
var	Variance 12

Рис. 5.2: Описательные статистики

- (1) – коэффициенты корреляции; (2) – матрица ковариации;
- (3) – Эксцесс; (4) – среднее; (5) – медиана (50% - точка или 0.5-квантиль);
- (5) – центральные моменты всех порядков;
- (6) – максимальное число пропущенных данных; (7) – среднее (при игнорировании пропущенных данных);
- (8) – размах (min-max);
- (9) – выборочная скошенность;
- (10) – стандартное отклонение; (11) – дисперсия.

в пакете также реализован интерфейс мобильного элементарного статистического анализа (рисунок), Запуск оболочки осуществляется по кноп-



captionСтатистический интерфейс

ке *Start* для сгенерированной выборки длиной $Sample = 500$ с именем именем "*mynorm*" и заданным теоретическим нормальным распределением (средним и среднеквадратичным отклонением), а также с последующей визуализацией полученного выборочного распределения. Далее приводится пример скрипта статистического исследования и фиттинга закона распределения.

```
1  clear;
2  clc;
3  xx=[128, 121, 124, 127, 116, 118, 126, 123, 121, 122, 124, 125, 128, 126, 129, 130,...,
4         129, 127, 119, 121, 124, 126, 123, 125, 128, 123, 126, 126,126, 132, 131, 134, 129 ];
5  xx=xx';
6  k=5
7  disp('Стандартизированные данные')
8  x=(xx-mean(xx))/std(xx)
9  subplot(2,2,1)
10 hist(x,k)
11 disp('n- частоты попаданий в i категорию')
12 disp('v-середины категорий')
13 [n,v]=hist(x,k)
14 p=n/sum(n)
15 disp('p- частоты попаданий в i категорию')
16 normfit(xx)
17 disp('musample sigmasample -- параметры наиболее близкого по мнк з-на распределения')
18 disp('mi si -- доверительные интервалы')
19 [musample sigmasample mi si]=normfit(xx,0.05)
```

здесь в строке (3) кода задана выборка, которой требуется фиттинг, в

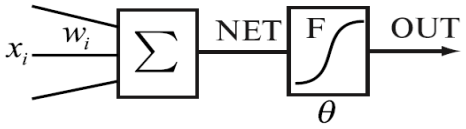
строке (8) данные выборки стандартизированы, в (8) создан графический объект из четырех окон, в строке (10) код означает помещение гистограммы в первое графическое окно; код (13) – создание массивов n, v, назначение которых следует из комментария; (16) – выполнение подбора параметров; (19) – получение оптимальных параметров распределения; полученные па-

```
K>> [musample sigmasample mi si]=normfit(xx,0.05)

musample =

  125.3636


sigmasample =

    4.0064


mi =

  123.9430
  126.7842


si =

    3.2219
    5.2992

K>>
```

раметрами представлены строкою вывода.

2 Лекция 16. Принципы нейросетевого моделирования

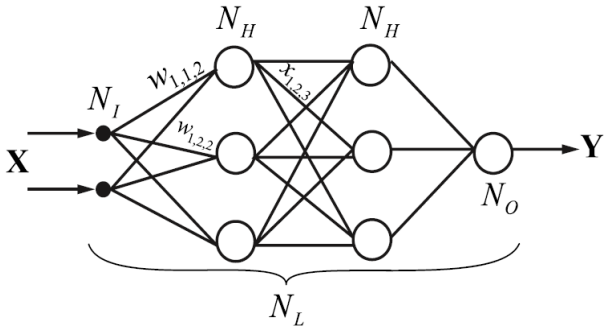
Мотивация использования модели живого нейрона в качестве математической модели основана на примерах эффективного обучения и получения адекватного результата. Схематически модель формального нейрона может быть представлена на схеме, которая интерпретируется следующим образом:



- Входная информация (внешний раздражитель, требующий реакции)
- Накопитель “знаний” (сумматор);
- Веса (весовые коэффициенты); характеризуют “вклад” элемента информации;
- Функция активации - характеризует природу работы нейрона: накопление информации, по достижении порогового значения осуществляется реакция;
- Выход - некоторый образ, созданный нейроном о входном “объект”.

Модель персептрона объединяет серию формальных нейронов и содержит скрытые слои, которые влияют на скорость сходимости оптимизационного процесса.

- Сеть – некоторое количество объединенных формальных нейронов;
- Многослойный персептрон – самый распространенный вид сети; приведен на рисунке;



- Количество слоев (их увеличение) позволяет работать с нелинейными моделями.

Успех функционирования сети, многослойного персептрона, не является безусловным, он определяется многими параметрами N_l – числом нейро-

$$NET_{jl} = \sum_i w_{ijl} x_{ijl}$$
$$OUT_{jl} = F\left(NET_{jl} - \theta_{jl}\right)$$
$$x_{ij(l+1)} = OUT_{il}$$

нов во входном слое; N_H – числом нейронов в скрытом слое; N_O – числом нейронов в выходном слое; $\boldsymbol{X}$ – представительностью входных сигналов; адекватным описанием $\boldsymbol{y}$ – выходных сигналов; успешным выбором начальных приближений w_{ijl} – весовых коэфициентов; ϑ_{jl} – порогового уровня для нейрона в слое; lздесь i – индекс входа; j – номер нейрона в слое; l – номер слоя.

На рисунке 5.3 предлагается модель N–го слойного персептрона, предназначенная для аппроксимации нелинейной функции и пример 5.4 параметров успешной сети многослойного персептрона (три слоя, функции активации и т.д.), здесь очевидна невозможность применения линейной модели; входы – значения функции в дискретные моменты времени; результат разделения представлен на рисунке

$$f(x) = \underbrace{F\left(\underbrace{\sum_{i_N} w_{i_N j_N N} \ldots \underbrace{\sum_{i_2} w_{i_2 j_2 2}\, \underbrace{F\left(\sum_{i_1} w_{i_1 j_1 1} x_{i_1 j_1 1} - \theta_{j_1 1}\right)}_{\text{слой } 1} - \theta_{j_2 2} \ldots - \theta_{j_N N}}_{\text{слой } 2}\right)}_{\text{слой } N}$$

Рис. 5.3: Модель персептрона

$$f(x) = \sum_i c_i f_i(x)$$

Успех модели существенно зависит от нормировки входных данных, поскольку каждая компонента входного сигнала, вообще говоря, представляет свое признаковое пространство (наделенное своей размерностью). Для этого требуется подготовить входные данные, а именно: центрировать относительно среднего (если соразмерные признаки) или стандартизировать, основной целью такой процедуры является такая нормализация данных, чтобы большая часть результатов действия сумматора попадала в область изменяемых значений функции активации. В задачах прогноза если важно отследить тенденции поведения исследуемого признака, применяется преобразование данных вида: $y = sgn(f(t_{i+1} - f(t_i))$. Влияет на результат также очередность подаваемых входов. Следует иметь в виду, что сети следует предъявлять случайные прообразы (принадлежащие разным классам). При последовательном представлении входов каждого класса сеть практически не обучается. Пороговые значения функций активации целесообразно предъявлять как настраиваемые параметры, для этого порог включается как входной сигнал с весом, равным единице (см. рисунок 5.5). На каждом этапе выбора параметров сети существуют правила успеха, так немаловажным является назначение начальных весов для выбранной архитектуры сети.

Инициализация больших весовых коэффициентов – приводит к тому, что нейроны быстро насыщаются и выходы попадают в области малого из-

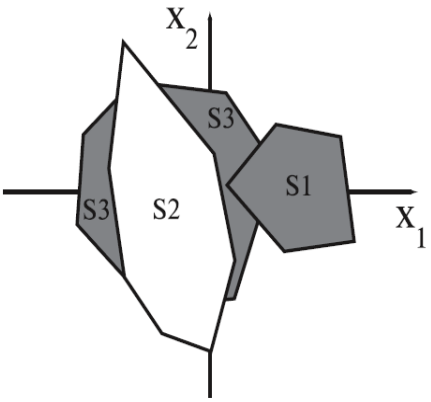


Рис. 5.4: Нелинейная модель. Аппроксимация

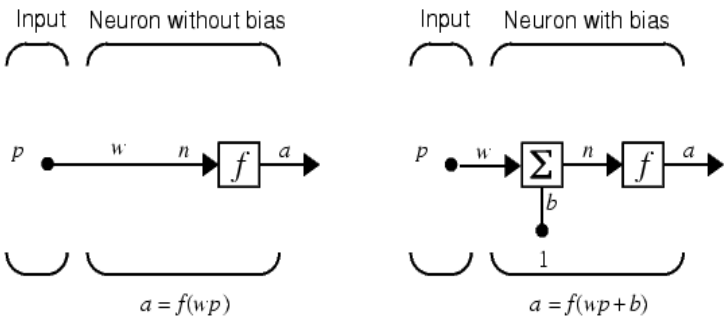


Рис. 5.5: Сравнение моделей.

менения градиентов, а значит, медленного обучения. Инициализация малых весовых коэффициентов приводит к медленному процессу обучения. Вес и параметры функции активации – следует выбирать согласованно, чтобы текущие выходы попадали в область активного разделения функции активации.

Существует возможность динамического влияния на параметры сети из рисунка 5.6) следует скачкообразное повышение эффективности; так на первом шаге задается заведомо малое число нейронов и происходит изменение числа нейронов скрытого слоя, при этом модификация происходит через фиксированный интервал времени (представляющий некоторое число итераций); динамически может меняться также число обучения $\varepsilon \in (0, 1)$; в начале обучения оно выбирается близким к единице, уменьшаясь от итерации к итерации. На рисунке 5.7 приведены графики функций активации,

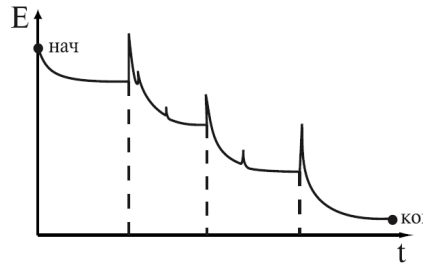


Рис. 5.6: Изменение ошибки при добавлении нейронов

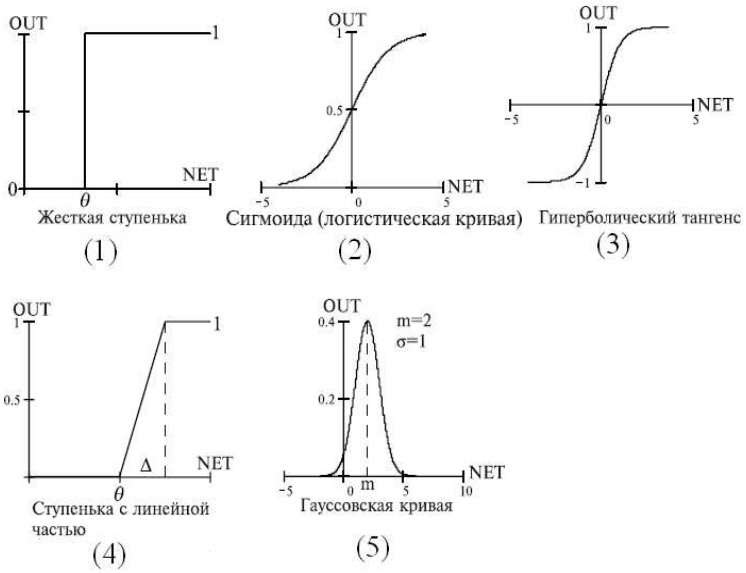


Рис. 5.7: Функции активации.

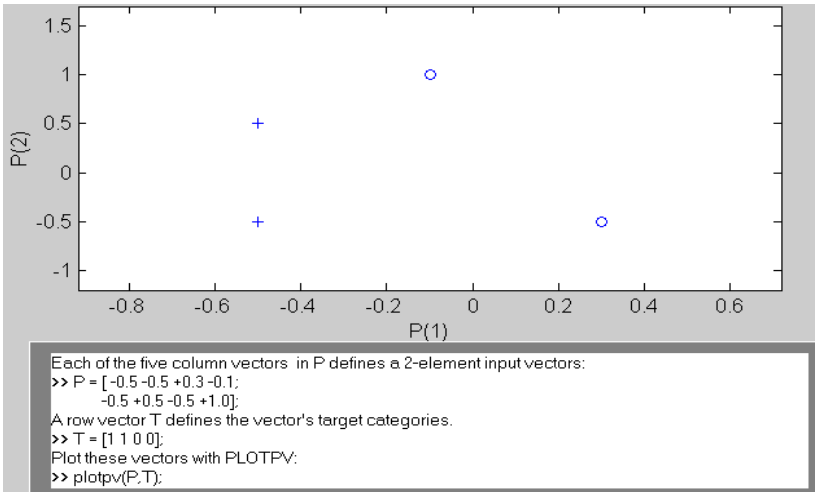
выбор каждой из которых определяется логикой задачи обучения (классификации).

3 Лекция 17. Некоторые функции Neural NetWork Toolbox для нейросетевой оптимизации

Основные этапы моделирования сети в MatLab заключаются в последовательном выполнении следующих действий:

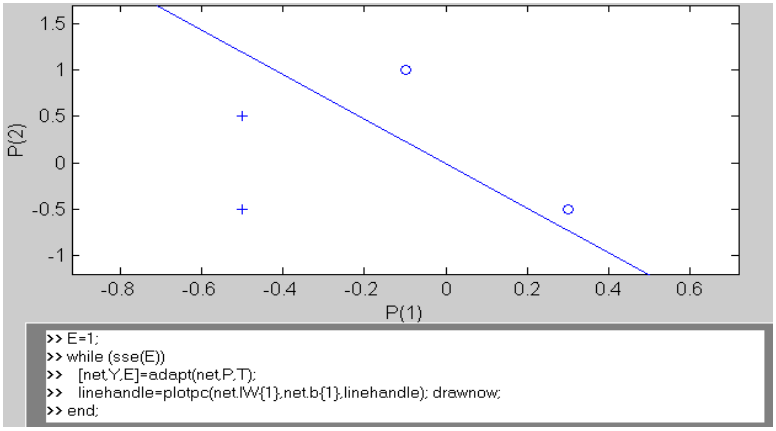
- **Выбор модели сети** выходное множество нормализуется
  - Линейная сеть
  - Персептрон
- **Выбор архитектуры сети**
  - количество нейронов в слое
  - количество скрытых слоев
- **Инициализация сети**
  - Выбор весов (случайный)
  - Стандартизация входов
  - Формализация выходов (согласно типу задачи)
  - Задание числа эпох или точности классификации – условия завершения алгоритма

В задачах классификации с помощью нейронных сетей входное множество состоит из объектов, многомерных векторов, которые каждой своей координатой определяют сущность значимых признаков, участвующих в классификации; выходное множество – количество классов, например, для которых предусмотрено разделение обучающим множеством. Как правило, выходные векторы – финитные векторы, в которых i–я координата соответствует принадлежности i–му классу.



Создадим сеть $net = newp([-11; -11], 1)$, – здесь первый параметр в квадратных скобках означает размах по каждой координате входного вектора, второй (1) – количество нейронов в слое, заметим, что для линейной классифиикации, достаточно одного слоя нейронов.

Рассмотрим пример, который представляет собой модель обучения, реализованный в пакете с помощью функций библиотеки модуля.



здесь E – задается большая ошибка и опция *sse* определяет меру отклонений (ошибку) как сумму квадратов ошибок сигнала выхода сети и выходов обучения. Функция *adapt* – подстраивает сеть, пока E не становится нулевой, при этом в полях структуры *net* хранятся параметры сети. С помощью графического оператора *plotpc*, с использование подстроенных параметров сети (весового коэффициента $net.IW\{1\}$ и порога $net.b\{1\}$)

строится классификация.

Рассмотрим случай одного скрытого слоя. Такая модель является оправданной в случае линейной классификации, а также при аппроксимации функции и определении ее параметров. На рисунке 5.8

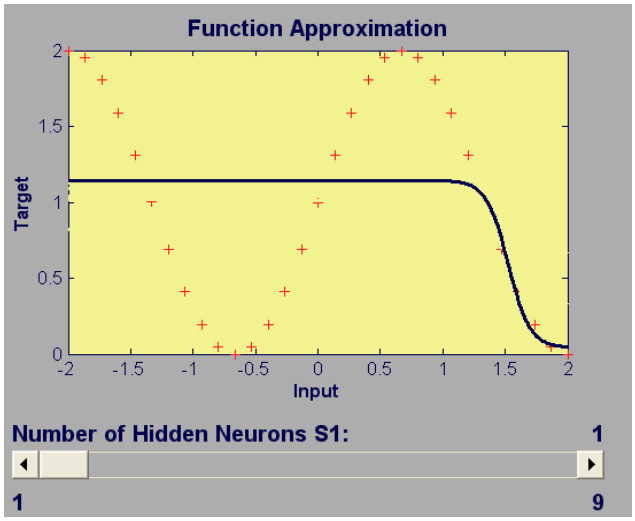


Рис. 5.8: пример неудовлетворительной аппроксимация функции, один слой

$\circlearrowleft$. *Заметим, для аппроксимации нелинейной функции нейронной сетью недостаточного одного слоя.* Величина нейронов в слое, количество слоев, выбор типа пороговых функций – задача оптимизационная, но очевидно, реализованная в пакете модель, демонстрирует факт влияния выбора серии скрытых слоев при аппроксимации функции в примере на рисунке 5.9 с числом слоев, большим трех ошибка получается меньше, чем 10^{-6}.

Модель персептрона с обратным распространением ошибки оказалась эффективной в оптимизационных нелинейных задачах. Так в в модели персептрона происходит “предъявление” образа всей сети (в направлении от входа к выходу), а затем подстройка параметров сети в обратном направлении задачах для каждого вектора-входа.

Создание модели персептрона с обратным распространением ошибки средствами пакета MatLab реализуется с использованием следующих функций:

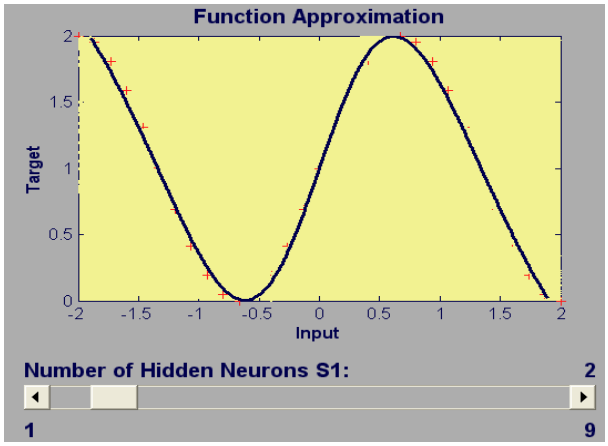


Рис. 5.9:

Сеть $net = newff$ формирует сеть персептрона с обратным распространением ошибки, которая последовательно задается параметрами этой сети: [размах 1-й компоненты входа; ... размах k-й компоненты входа]; [число нейронов в первом слое, ..., число нейронов в r-слое]; $\{'tansig'$ – имя тангенциальной функции активации,(... – и серии функций); $'purelin'$ – имя линейных r-функции активации; $'traingd'$ – градиентный метод обучения.

Инициализация параметров сети осуществляется : $net.trainParam.epochs = 500$; – заданием числа эпох; $net.performFcn =' sse'$; – метода вычисления ошибки; $net.trainParam.goal = 0.1$; – предела ошибки. Графически визуализируется каждая 20–я итерация $net.trainParam.show = 20$. В результате сеть построена, адаптирована своими параметрами к обучающему множеству и получение на ее основе образа входного сигнала осуществляется по следующей

$$Y = sim(net, p)$$

– команде, здесь net – сеть, p –подаваемый вектор входа.

4 *Вопросы для самопроверки:*

Персептрон

- – линейная модель классификации?
- с обратным распространением ошибки – безусловно сходящаяся модель?
-
- не имеет таких параметров как пороговые функции, число скрытых слоев, количество нейронов слоя?

Библиотека Statistics

- предоставляет системные функции распределения вероятностей и и плотности распределения вероятностей и объединяются в тематической справке с именами *Distrib DensDistrib*?
- предоставляет системные функции основных статистик центральной симметрии? Какие?
- предоставляет возможность визуализации статистических законов с помощью полигона частот и гистограмм, каким образом?

Проектное задание

Заключается в построении пользовательского интерфейса, объединяющего все элементы управления, в том числе $IMAGE$. Требуется реализовать в рамках интерфейса задачу элементарного статистического анализа или (и) классификации на основе обучающего множества (с помощью линейной сети или персептрона с одним скрытым слоем или серией слоев).

5 Литература

1. Говорухин В., Цибулин В. Компьютер в математическом исследовании. Maple, MATLAB, LatTeX СПб.: Питер 2001. 624 с.

2. Потемкин В. Г. Система инженерных и научных расчетов. MatLab 5.X М.: 1999. Т.1, Т.2

3. Герман-Галкин С. Г. Компьютерное моделирование полупроводниковых систем. MATLAB 6.0 СПб.: КОРОНА принт, 2001. 320 с. Электронный учебник

4. Гандер В., Гржебичек И. Решение задач в научных вычислениях с применением Maple и MATLAB. ISBN: 985-6642-06-X. И.: "Вассамедина"2005г. 520 стр.

5. Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. М.: Мир, 1985. 509 с.

6. В.Е. Гмурман. Теория вероятностей и статистика.М., Высшая школа, 1998 г., 479 с.

7. Конюшенко В.В. MatLab – язык технических вычислений. Вычисления. Визуализация. Программирование Электронный учебник

8. Дональд Кнут. Искусство программирования, том 1. Основные алгоритмы. The Art of Computer Programming, vol.1. Fundamental Algorithms. 3-е изд. М.: “Вильям”, 2006. С. 720. ISBN 0-201-89683-4

9. Курбатова Е.А. MATLAB 7. Самоучитель. Издательство: Вильямс. 2005г. 256 с.

10. С. Хайкин. Нейронные сети. М.-С.П.-К, И.:Вильямс 2006., 1103 с.