KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021
SYLLABUS

Semester – V

		L	T	P	C
15MMU504	**MAT LAB PROGRAMMING**	**5**	**0**	**0**	**5**

Course Objective:

This course will introduce a comprehensive introduction to fundamental programming concepts using a block-structured language (MATLAB).

Course Outcome:

To enable the students to learn about the Mathematical software MATLAB for high-performance numerical computations and visualizations.

UNIT I

Introduction - Basics of MATLAB, Input – Output, File types – Platform dependence – General commands.

UNIT II

Interactive Computation: Matrices and Vectors – Matrix and Array operations – Creating and Using Inline functions – Using Built-in Functions and On-line Help – Saving and loading data – Plotting simple graphs.

UNIT III

Programming in MATLAB: Scripts and Functions – Script files – Functions files-Language specific features – Advanced Data objects.

UNIT IV

Applications – Linear Algebra - Solving a linear system – Finding Eigen values and Eigen vectors – Matrix Factorizations.

UNIT V

Applications – Data Analysis and Statistics – Numerical Integration – ordinary differential equations – Nonlinear Algebraic Equations.

TEXT BOOK

1. RudraPratap, 2003. Getting Started with MATLAB-A Quick Introduction for Scientists and Engineers, Oxford University Press.

REFERENCES

1. William John Palm, 2005. Introduction to Matlab 7 for Engineers, McGraw-Hill Professional.New Delhi.

2. Dolores M. Etter, David C. Kuncicky, 2004.Introduction to MATLAB 7, Prentice Hall, New Delhi.

3. Kiranisingh.Y,Chaudhuri.B.B, 2007.Matlab Programming, Prentice-Hall Of India Pvt.Ltd, New Delhi.

4. Brian Hahn, 2016. Essential MATLAB for Engineers and Scientists.6th edition , Elsevier publication.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021.

Department of Mathematics

LECTURE PLAN

Subject: MATLAB programming **Subject Code: 15MMU504**

S.No	Lecture Duration	Topic to be covered	Support Material
		Unit – I	
1.	1	Introduction to MATLAB	T1: Ch 1: Pg: 3-5
2.	1	MATLAB environment	T1: Ch 1: Pg: 7-9
3.	1	MATLAB desktop	R2: Ch 2: Pg: 18-20
4.	1	MATLAB Windows	R2: Ch 2: Pg: 20-22
5.	1	Graphic window and Edit window	R4: Ch 2: Pg: 23-24
6.	1	Input – Output	T1: Ch 1: Pg: 10-11
7.	1	File types in MATLAB	T1: Ch 1: Pg: 11-12
8.	1	Platform dependence	T1: Ch 1: Pg: 12-13
9.	1	General commands in MATLAB	T1: Ch 1: Pg: 13-14
10.	1	Matrices and Arrays	T1: Ch 2: Pg: 17-20
11.	1	Example programs	T1: Ch 2: Pg: 25-28
12.	1	Example programs	T1: Ch 2: Pg: 33-38
13.	1	Recapitulation and discussion of possible question	
Total No. of Lecture hours planned – 13 hours			
T1. RudraPratap, 2003.Getting Started with MATLAB- A Quick Introduction for Scientists and Engineers, Oxford University Press. **R2.** Dolores M. Etter, David C. Kuncicky, 2004. Introductoin to MATLAB 7, Prentice Hall, New Delhi. **R4** Brian Hahn, 2016. Essential MATLAB for Engineers and Scientists.6th edition , Elsevier publication.			
		Unit – II	
1.	1	Examples of Interactive Computation	T1: Ch 3: Pg: 63-64
2.	1	Matrices and vectors	T1: Ch 3: Pg: 65-66
3.	1	Matrix manipulation and Creating vectors	T1: Ch 3: Pg: 66-69
4.	1	Continuation of Matrix manipulation	T1: Ch 3: Pg: 70-72
5.	1	Matrix and array operations	T1: Ch 3: Pg: 73-74
6.	1	Continuation of Matrix and array operations	T1: Ch 3: Pg: 74-75
7.	1	Elementary math functions	T1: Ch 3: Pg: 75-77
8.	1	Character strings	T1: Ch 3: Pg: 77-79
9.	1	Manipulation of Character strings	T1: Ch 3: Pg: 79-81
10.	1	Command line functions	T1: Ch 3: Pg: 83-84
11.	1	Using built in functions	T1: Ch 3: Pg: 85-87

12.	1	on line help	T1: Ch 3: Pg: 88-90
13.	1	Saving ,loading data	T1: Ch 3: Pg: 90-93
14.	1	Plotting simple graphs	T1: Ch 3: Pg: 94-96
15.	1	Recapitulation and discussion of possible question	
Total No. of Lecture hours planned – 15 hours			
T1. RudraPratap, 2003.Getting Started with MATLAB- A Quick Introduction for Scientists and Engineers, Oxford University Press.			
Unit – III			
1.	1	Script files	T1: Ch 4: Pg: 99-101
2.	1	Function files	T1: Ch 4: Pg: 102-104
3.	1	Nested Function	T1: Ch 4: Pg: 105-107
4.	1	Sub functions	T1: Ch 4: Pg: 109-110
5.	1	Inside of another function	T1: Ch 4: Pg: 110-111
6.	1	Language- specific features	T1: Ch 4: Pg: 111-112
7.	1	Types of variable in functions	R3: Ch 8: Pg: 247-248
8.	1	Continuation of Types of variable in functions	R3: Ch 8: Pg: 249-250
9.	1	Loops	T1: Ch 4: Pg: 114
10.	1	Branches	T1: Ch 4: Pg: 115-116
11.	1	control flow	T1: Ch 4: Pg: 116-117
12.	1	Interactive input	T1: Ch 4: Pg: 117-119
13.	1	Input and Output	T1: Ch 4: Pg: 119-120
14.	1	Advanced data objects	T1: Ch 4: Pg: 121
15.	1	Structures	T1: Ch 4: Pg: 122-124
16.	1	Creating structures	T1: Ch 4: Pg: 124
17.	1	manipulating structures	T1: Ch 4: Pg: 124-125
18.	1	Creating cells	T1: Ch 4: Pg: 125-126
19.	1	Manipulating cells	T1: Ch 4: Pg: 127-128
20.	1	Recapitulation and discussion of possible question	
Total No. of Lecture hours planned – 20 hours			
T1. RudraPratap, 2003.Getting Started with MATLAB- A Quick Introduction for Scientists and Engineers, Oxford University Press. **R3.** Kiranisingh. Y, Chaudhuri. B. B, 2007. Matlab Programming, Prentice- Hall of India Pvt. Ltd, New Delhi.			
Unit – IV			
1.	1	Solving a linear system of equations	T1: Ch 5: Pg: 135-136
2.	1	Continuous of solving systems of equations	R1: Ch 8: Pg: 341-344
3.	1	Continuous of solving systems of equations	R1: Ch 8: Pg: 345-347
4.	1	Continuous of solving systems of equations	R1: Ch 8: Pg: 348-350
5.	1	A general solution program	R1: Ch 8: Pg: 354-356
6.	1	Gaussian elimination method in MATLAB	T1: Ch 5: Pg: 136-137
7.	1	Finding determinant of a matrices	T1: Ch 3: Pg: 86-87
8.	1	Finding eigen values and eigen vectors	T1: Ch 5: Pg: 137-138
9.	1	Continuation of Finding eigen values and eigen	T1: Ch 3: Pg: 87-88

		vectors	
10.	1	Matrix factorizations	T1: Ch 5: Pg: 138-139
11.	1	Recapitulation and discussion of possible question	
Total No. of Lecture hours planned – 11 hours			
T1. RudraPratap, 2003.Getting Started with MATLAB- A Quick Introduction for Scientists and Engineers, Oxford University Press. **R1.** William John Palm, 2005. Introduction to Matlab 7 for Engineers.McGraw- Hill Professional. New Delhi.			
Unit – V			
1.	1	Data Analysis and Statistics	T1: Ch 5: Pg: 150-152
2.	1	Statistics and Histograms	R4: Ch 5: Pg: 106-112
3.	1	Continuation of Statistics and Histograms	R1: Ch 7: Pg: 298-300
4.	1	Numerical integration	T1: Ch 5: Pg: 152-154
5.	1	Double integration	T1: Ch 5: Pg: 154-156
6.	1	Ordinary differential equations	T1: Ch 5: Pg: 156-157
7.	1	Solving First order ODE	T1: Ch 5: Pg: 158-160
8.	1	Solving Second order ODE	T1: Ch 5: Pg: 160-162
9.	1	The ODE suite	T1: Ch 5: Pg: 163-164
10.	1	Event location	T1: Ch 5: Pg: 165-167
11.	1	Nonlinear algebraic Equations	T1: Ch 5: Pg: 168-169
12.	1	Advanced Topics	T1: Ch 5: Pg: 169-170
13.	1	Recapitulation and discussion of possible questions	
14.	1	Discussion of previous ESE question papers	
15.	1	Discussion of previous ESE question papers	
16.	1	Discussion of previous ESE question papers	
Total No. of Lecture hours planned -16 hours			
T1. RudraPratap, 2003.Getting Started with MATLAB- A Quick Introduction for Scientists and Engineers, Oxford University Press. **R1.** William John Palm, 2005. Introduction to Matlab 7 for Engineers.McGraw- Hill Professional. New Delhi. **R4** Brian Hahn, 2016. Essential MATLAB for Engineers and Scientists.6thedition , Elsevier publication.			

TEXTBOOK:

T1. RudraPratap, 2003.Getting Started with MATLAB- A Quick Introduction for Scientists and Engineers, Oxford University Press.

REFERENCES:

R1. William John Palm, 2005. Introduction to Matlab 7 for Engineers.McGraw- Hill Professional. New Delhi.

R2. Dolores M. Etter, David C. Kuncicky, 2004. Introductoin to MATLAB 7, Prentice Hall, New Delhi.

R3. Kiranisingh. Y, Chaudhuri. B. B, 2007. Matlab Programming, Prentice- Hall of India Pvt. Ltd, New Delhi.

R4 Brian Hahn, 2016. Essential MATLAB for Engineers and Scientists.6th edition , Elsevier publication.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021.
Department of Mathematics

Subject : MATLAB programming	**Subject Code : 15MMU504**	**L T P C**
Class : III – B.Sc. Mathematics	**Semester : V**	**5 0 0 5**

UNIT I

Introduction - Basics of MATLAB, Input – Output, File types – Platform dependence – General commands.

TEXT BOOK

1. RudraPratap, 2003. Getting Started with MATLAB-A Quick Introduction for Scientists and Engineers, Oxford University Press.

REFERENCES

1. William John Palm, 2005. Introduction to Matlab 7 for Engineers, McGraw-Hill Professional.New Delhi.

2. Dolores M. Etter, David C. Kuncicky, 2004.Introduction to MATLAB 7, Prentice Hall, New Delhi.

3. Kiranisingh.Y,Chaudhuri.B.B, 2007.Matlab Programming, Prentice-Hall Of India Pvt.Ltd, New Delhi.

4. Brian Hahn, 2016. Essential MATLAB for Engineers and Scientists.6th edition , Elsevier publication.

UNIT – I

Introduction to MATLAB

What Is MATLAB?

MATLAB is a software package for high-performance numerical computation andvisualization. It provides an interactive environment with hundreds of built-in functionsfor technical computation, graphics, and animation. Best of all, it also provideseasy extensibility with its own high-level programming language. The nameMATLAB stands for MATrixLABoratory.

The diagram in Fig. 1 .1 shows the main features and capabilities of MATLAB.MATLAB's built-in functions provide excellent tools for linear algebra computations, data analysis, signal processing, optimization, numerical solution of ordinarydifferential equations (ODEs), quadrature, and many other types of scientific computations. Most of these functions use state-of-the-art algorithms. There are numcrom;functions for 2-D and 3-D graphics , as well as for animation. Also, for Lhoscw ho cannot do without their Fortran or C codes, MATLAB even provides an externalinterface to run those programs from within MATLAB. The user, however,is not limited to the built-in functions; he can write his own functions in the MATLABlanguage. Once written, these functions behave j ust like the built-in functions.

MATLAB 's language is very easy to learn and to use.There are also several optional "toolboxes" available from the developers ofMATLAB. These toolboxes are collections of functions written for special applicationssuch as symbolic computation, image processing, statistics, control systemdesign, and neural networks. The list of toolboxes keeps growiug with time. Thereare now more than 50 such toolboxes. We do not attempt introduction to anytoolbox here, with the exception of the Symbolic Math Toolbox.The basic building block o f MATLAB i s the matrix. The fundamental data typeis the array. Vectors, scalars, real matrices, and complex matrices are all automaticallyhandled as special cases of the basic data type. What is more, you almostnever have to declare the dimensions of a matrix. MATLAI3 simply loves matricesand matrix operations. The LuilL-in functions are optimized for vector operations.Consequently, vcctorized1 commands or codes run much faster in MATLAB.

Does MATLAB Do Symbolic Calculations?

(MATLAB vs. Mathematica or Maple)

If you are new to MATLAB, you are likely to ask this question. The first thing torealize is that MATLAB is primarily a numerical cumputatiou package, althoughwith the Symbolic Math Toolbox (standard with the Student Edition of MATLAB,for an introduction) it can do symbolic algebra. 2 MathematicaandMaple are primarily symbolic algebra packages. Of course , they do numerical computationstoo. In fact , if you know any of these packages really well, you can doalmost every calculation that MATLAB does using that software. So why learnMATLAB? Well, MATLAB 's ease of use is its best feature. Also, it has a shallowlearning curve (more learning with less effort) whereas the computer algebra

systemshave a steep learning curve. Because MATLAB was primarily designed to donumerical calculations and computer algebra systems were not, MATLAB is oftenmuch faster at these calculations--often as fast as C or Fortran. There are otherpackages, such as Xmath, that are also closer in aim and scope but seem to bepopular with people in some specialized application areas. The bottom line is, innumerical computations, especially those that us vectors and matries, MATLABbeats everything hand clown in terms of ease of use, availability of built-in functions,ease of programming, and speed. The proof is in the phenomenal growth ofMATLAB users around the world in the la.'lt two decades . There arc more than 2000universities and thousands of companies listed as registered users. MATLAB's popularitytoday has forced such powerful packages as Mathematica and many othersto provide extensions for files in MATLAB's format!

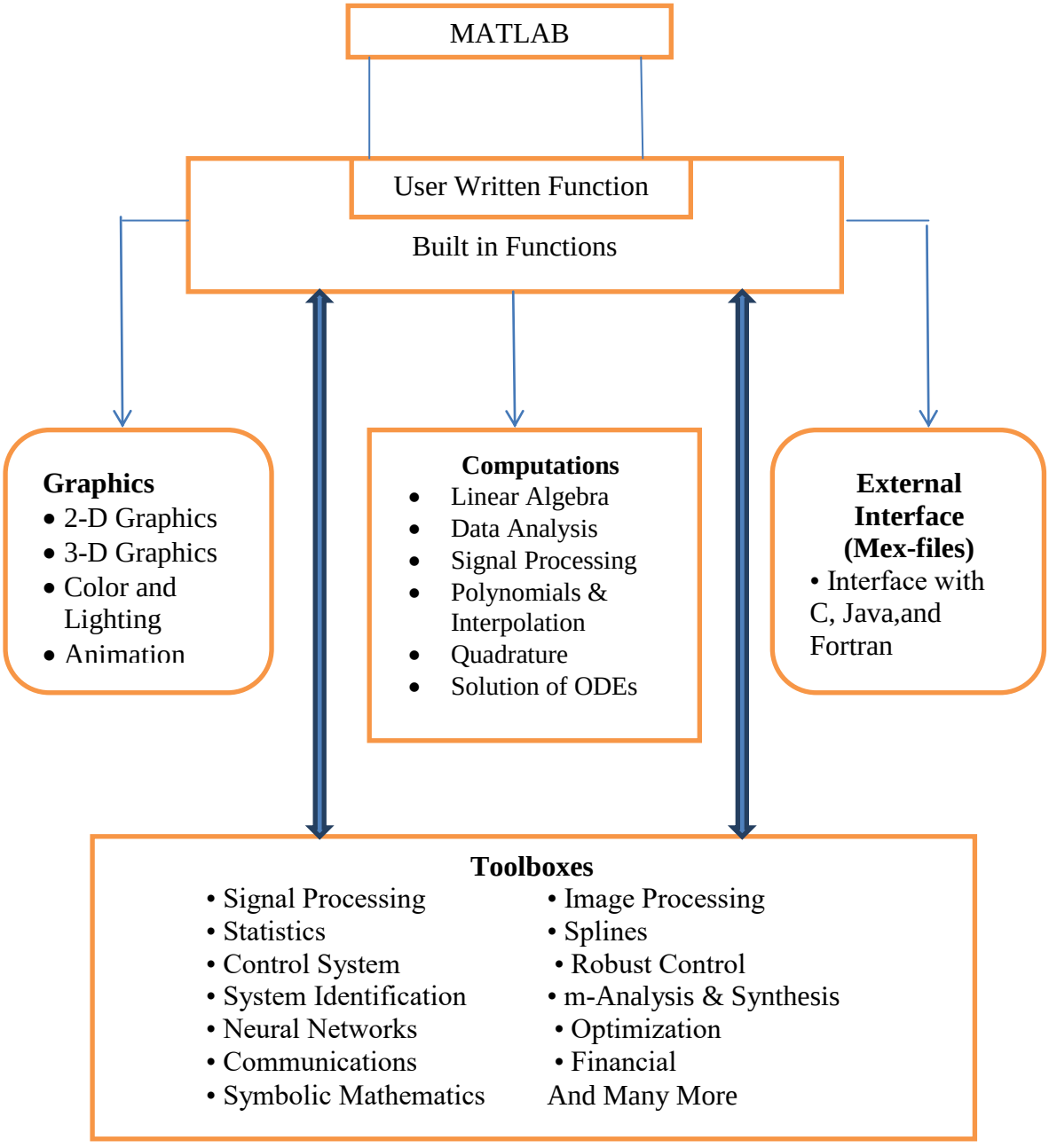


Figure 1.1: A schematic diagram of MATLAB 's main feat ures

MATLAB is a very powerful and sophisticatedpackage. It takes a while to understand its real power. Unfortunately, mostpowerful packages tend to be somewhat intimidating to a beginner. That is whythis book exists-to help you overcome the fear, get started quickly, and becomeproductive in very little time. The most useful and easily accessible features ofMATLAB are discussed first to make you productive and build your confidence.Several features are discussed in sufficient depth, with an invitation to explore themore advanced features on your own. All features are discussed through examplesusing the following conventions:

• **Typographical styles**:

- All actual MATLAB commands or instructions are shown in typedface. Menu commands, files names, etc. , are shown in sa ns serif font .

- Place holders for variables or names in a command are shown in italics.

So, a command shown as he lp topic implies that you have to type theactual name of a topic in place of topic in the command.

- Italic text has also been used to emphasize a point and, sometimes, tointroduce a new term.

• **Actual examples**: Actual examples carried out in MATLAB are shown ingray, shaded boxes. Explanatory notes have been added within small whiterectangles in the gray boxes, as shown in Fig. 1.2. These gray, boxed figuresare intended to provide a parallel track for the impatient reader. If you wouldrather try out MATLAB right away, you are encouraged to go through theseboxed examples. Most of the examples are designed :oo that you can (more orless) follow them without reading the entire text . All examples are systemindependent. After trying out the examples, you should read the appropriatesections.

• **On-line help**: We encourage the use of on-line help. For almost all majortopics, we indicate the on-line help information in a :::mall box in the margin,as shown here on the left .

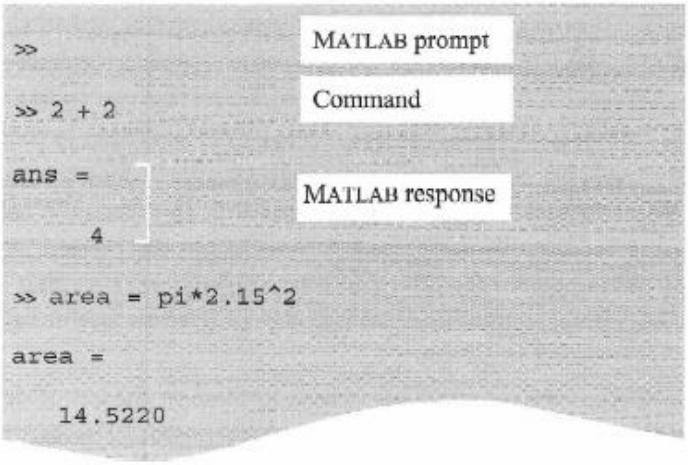


Figure 1.2: Actual examples carried out in MATLAB are shown in gray boxesthroughout this book. The text in the white boxes inside these gray boxes areexplanatory notes.

Typing help topic in MATLAB with the appropriate topic name providesa list of functions and commands for that topic. Detailed help can then beohtained for any of those commands and functions.

Basics of MATLAB

Here we discuss some basic features and commands . To begin, let us look at thegeneral structure of the MATLAB environment .

MATLAB windows

On almost all systems, MATLAB works through three basic windows , which areshown in Fig. 1.3 and discussed here.

1. MATLAB desktop: This is where MATLAB puts you when you launch it(see Fig. 1.3). The MATLAR desktop , by default, consists of the followingsubwindows .

Command window: This is the main window. It is characterized by the MATLAB command prompt (>>). When you launch the applicationprogram, MATLAB puts you window. All commands, includingthose for running user-written programs, are typed m this window at theMATLAB prompt . In MATLAB , this window is a part of the MATLABwindow (see Fig. 1 . 3) that contains other smaller windows or panes. Ifyou can get to the command window, we advise you to ignore the otherfoursubwindows at this point . As software packages become more andmore powerful, their creators add more and more features to address the needs of experienced user Unfortunately, it makes life harder for thebeginners-there is more room for confusion, distraction, and intimidation.Although we describe the other subwindows here that appear withthe command window,

Current Directory pane: This pane is located on the left of the Command Window i n the default MATLAB desktop layout . This is where all yourfiles from the current directory are listed. You can do file navigationhere. Make sure that this is the directory where you want to work sothat MATLAB has access to your files and where it can save your newfiles. If you change the current directory (by navigating through your filesystem), make sure that the selected directory is also reflected in the littlewindow above the Command Window marked Current Directory. Thislittle window and the current directory pane are interlinked; changingthe directory in one is automatically reflected in the other.You also have several options of what you can do with a file onceyou select it (with a mouse click). To see the options, click the rightbutton of the mouse after selecting a file. You can run M-files, renamethem, delete them, etc.

(File) Details pane : Just below the Current Directory pane is the Detailspane that shows the details of a file you select in the current directorypane. These details are normally limited to listing of variables from aMAT-file (a binary data file discussed later), showing titles of M-files,and listing heading of cells if present in M-files. You do not need tounderstand these details yet .

Workspace pane: This subwindow lists all variables that you have generatedso far and shows their type and size. You can do various thingswith these variables, such as plotting, by clicking on a variable and thenusing the right button on the mouse to select your options.

Command History pane: All commands typed on the MATLAB promptin the command window get recorded, even across multiple sessions (youworked on Monday, then on Thur::;day, and then on next Wednesday, andso on), in this window. You can select a command from this window withthe mouse and execute it in the command window by double-clicking onit. You can also seled a set. of commands from this window and createan M-file with the right click of the mouse (and selecting the appropriateoption from the menu).

2. Figure window: The output of all graphics commands typed in the commandwindow are flushed to the graphics or figure window, a separate graywindow with (default) white background color. The user can create as manyfigure windows as the system memory will allow.

3. Editor window: This is where you write, edit , create , and save your ownprograms in files called M-jiles. You can use any text editor to carry out thesetasks . On most systems , MATLAB provides its own built-in editor. However,you can use your own editor by typing the standard file-editing command thatyou normally use on your system. From within MATLAB , the command istyped at the MATLAB prompt following the exclamation character (!) . Theexclamation character prompts MATLAD to return the control temporarilyto the local operating system, which executes the command following thecharacter. After the editing is completed, the control is returned to MATLAB.

For example, on UNIX systems , typing !myprogram . m at the MATLABprompt (and hitting the return key at the end) invokes the vi editor on thefilemyprogra m . m . Typing ! emac s myprogram . m invokes Lheemacs editor.

On-line help

• On- line documentation: MATLAB provides on-line help for all its builtinfunctions and programming language constructs . The commands lo ok for,help, helpwin, and helpdesk provide on-line help.for a description of the help facility.

• Demo: MATLAB has a demonstration program that shows many of features.The program includes a tutorial introduction that is worth trying.Type demo at the MATLAB prompt to invoke the demonstration program,and follow the instructions on the screen.

Input-output

MATLAB supports interactive computation, taking the input fromthe screen, and flushing the output to the screen. In addition, it can read input files and write output files . The following features hold for allforms of input-output :

• Data type: The fundamental data type in MATLAB is an array. It encompassesseveraldisLlnct data objects-integers , doubles (real numbers), matrices,character

strings, structures, and cells. 3 In most cases, however, younever have to worry about the data type or the data object declarations . Forexample, there ia no need to declare variables as nml or complex . When a real number is entered as the value of a variable, MATLAB automatically sets thevariable to be real (double).

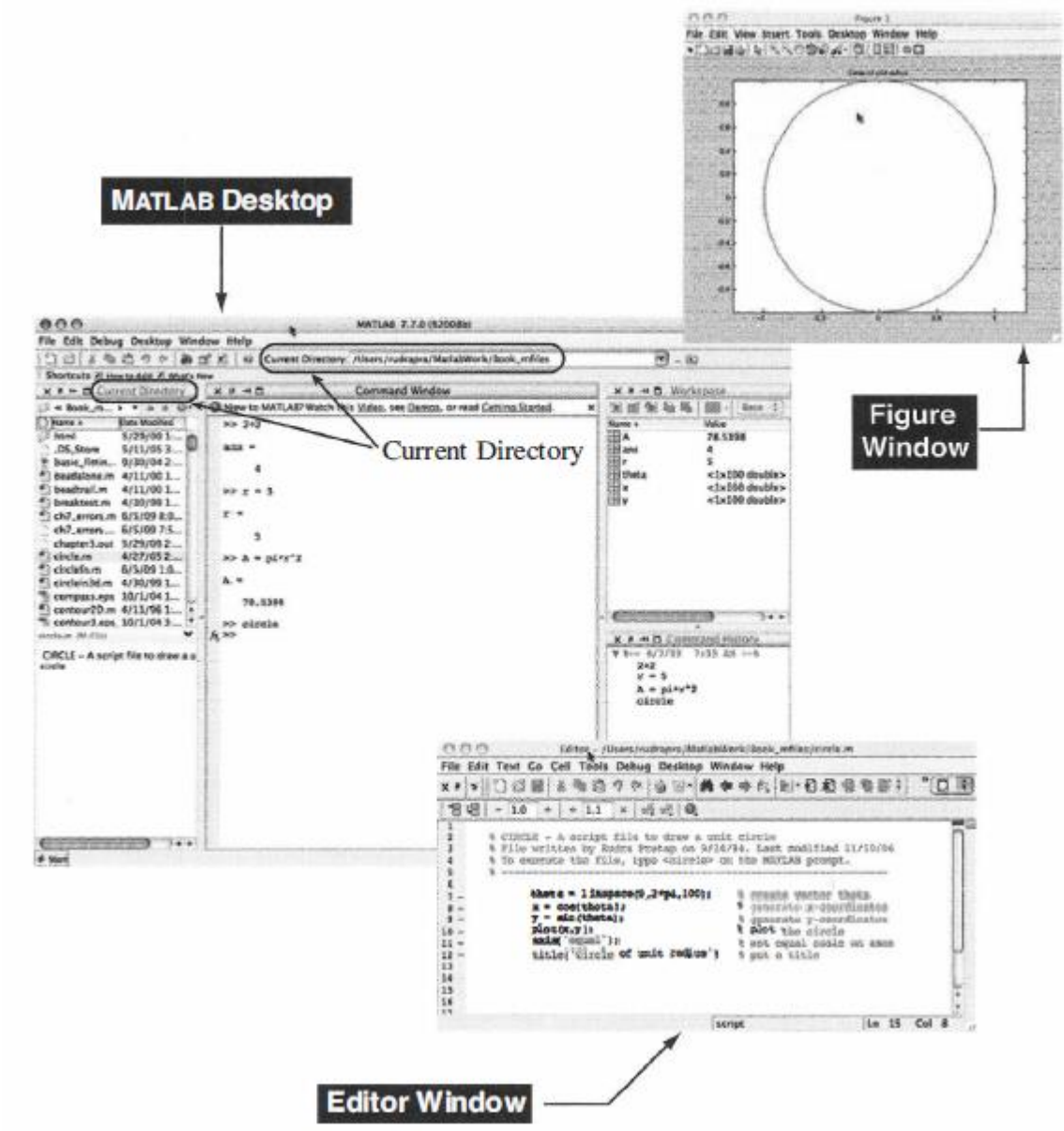


Figure 1.3: The MATLAB environment consists of the MATLAB desktop, a figurewindow, and an editor window. The figure and the editor windows appear onlywhen invoked with the appropriate commands. For example, you can open theeditor window by selecting File →New → Blank →M - F i le, and open a blank figurewindow by selecting F i l e→New →Figure.

• **Dimensioning:** Dimensioning is automatic in MATLAB . No dimension statementsare required for vectors or arrays. You can find the dimensions of anexisting matrix or a vector with the size and length (for vectors only) commands.

• **Case sensitivity**: MATLAB is case-sensitive; that is, it differentiates betweenthe lowercase and uppercase letters. Thus a and A are different variables.Most MATLAB commands and built-in function calls are typed inlowercase letters . You can turn case sensitivity on and off with the casesencommand. However, we do not recommend it .

• **Output display**: The output of every command is displayed on the screenunless MATLAB is directed otherwise . A semicolon at the end of a commandsuppresses the screen output , except for graphics and on-line help commands.

The following facilities are provided for controlling the screen output :

- **Paged output** : To direct MATLAB to show one screen of output ata time, type more on at the MATLAB prompt . Without it , MATLAB flushes the entire output at once, without regard to the speed at whichyou read.

- Output format :

Though computations inside MATLAB are performed using double precision,the appearance of floating point numbers on the screen is controlledby the output format in use. There are several different screen outputformats . The following table shows the printed value of l01r in sevendifferent formats .

format short	31.4159
format short e	3.141 6e+00 1
format long	31.41592653589793
format long e	3.141592653589793e+001
format short g	31.416
format long g	31.4159265358979
format hex	403f6a7a2955385e
format rat	3550/ 113
format bank	31.42

The additional formats, format compact and format loose , control thespacing above and below the displayed lines, and format + displays a + ,-, and blank for positive, negative, and zero numbers, respectively. Thedefaultis format short . The display format is set by typing f ormo.ttype on the command line

• Command history: MATLAB saves previously typed commands in a buffer.These commands can be recalled with the up-arrow key (↑). This helps inediting previous commands. You can also recall a previous command by typing the first few characters and then pressing the ↑ key. Alternatively, youcan double-click on a command in the Command

History pane (where all yourcommands from even previous sessions of MATLAB are recorded and listed)to execute it in the command window. On most UNIX systems , MATLAB 'scommand-line editor also understands the standard emacskeybindings.

File types

MATLAB can read and write several types of files. However, there are mainly fivedifferent types of files for storing data or programs that you are likely to use often:

M-files are standard ASCII texL files, with a . m extension to the filename. Thereare two types of these files: script files and junction files. Most programs you write in MATLAB are saved as M-files. All built-infunctionR in MATLAB are M-files, most of which reside on your computer inprecompiled format . Some built-in functions are provided with source code inreadable M-files so that they can be copied and modified.

Mat-files are binary datafiles, with a . m a t extension to the filename. Mat-files arecreated by MATLAB when you save data with the s ave command. The datais written in a special format that only MATLAB can read. Mat-files can beloaded into MATLAB with the l o ad command.

Fig-files are binary figure files with a .fig extension that can be opened again inMATLAB as figures. Such files are created by saving a figure in this formatusing the Save or Save As options from the File menu or using thesaveas command in the command window. A fig-file contains all the informationrequired to recreate the figure. Such files can be opened with the openfilename . fig command.

P-files are compiled M-files with a . p extension that can be executed in MATLABdirectly (without being parsed and compiled). These files are created with thepcode command. If you develop an application that other people can use butyou do not want to give them the source code (M-file), then you give themthe corresponding p-code or the p-file.

Mex-files are MATLAB-callable Fortran, C , and Java programs, with a . mex extension

to the filename. Use of these files requires some experience with MATLABand a lot of patience. We do not discuss Mex-files in this introductorybook.

Platform dependence

One of the best features of MATLAB is its platform independence. Once you are inMATLAB , for the most part , it does not matter which computer you are on. Almostall commands work the same way. The only commands that differ are the ones thatnecessarily depend on the local operating system, such as editing (if you do not usethe built-in editor) and saving M-files. Programs written in the MATLAB languagework exactly the same way on all computers. The user interface (how you interactwith your computer), however, may vary a little from platform to platform.

• **Launching MATLAB :** If MATLAB is installed on your machine correctly thenyou can launch it by following these directions:

On PCs : Navigate and find the MATLAB folder, locate the MATLAB program,and double-click on the program icon to launch M ATLAB. If youhave worked in MATLAB before and have an M-file or Mat-file thatwas written by MATLAB , you can also double-click on the file lulaunchMATLAB.

On UNIX machines: Type mat lab on the UNIX prompt and hit returnorenter. If MATLAB is somewhere in your path, it will be launched. If itis not , ask your system administrator.

• **Creating a directory and saving files:** Where should you save your files sothat MATLAB can easily access them? MATLAB create a default foldercalledMatlab in□ide Documents (on Macs) , or My Docu m e n ts (on PCs)where it saves your files if you do not specify any other location. If you arethe only user of MATLAB on the computer you are working on, this is fine.You can save all your work in this folder and access all your files easily (defaultsetup). If not, you have to create a separate folder for saving your work.

Theoretically, you can create a directory /folder anywhere, save your files,and direct MATLAB to find those files. The most convenient place, however,to save all user-written files is in the default directory M ATLAB created bythe application in your Documents or My Documents folder. This way alluser-written files are automatically accessible to MATLAB . If you need tostore the files somewhere else , you might have to specify the path to the filesusing the path command, or change the working directory of MATLAD tothe desired directory with a few navigational clicks in the Current Directorypane. We recommend the latter.

• **Printing:**On PCs : To print the contents of the current active window (command,figure, or edit window) , select Print . . . from the File menu and clickPrint in the dialog box. You can also print the contents of the figurewindow by typing print at the MATLAB prompt.

On UNIX machines : To print a file from inside MATLAB , type the appropriateUNIX command preceded by the exclamation character (!) . Forexample, to print the file startup.m, type ! lpr startup .m on the MATLABprompt . To print a graph that is currently in the figure windowsimply type print on the MATLAB prompt .

Basic Commands

At your command line type each of the following lines (but not '>>'). Separate each with the 'enter' key.

```
>> 1+1;
```

```
>>1+1
>>3*2
>>2^4
>>16.6-3/6
```

The preceding lines used basic matlab operators, +,-,^,*, and / for calculations. Next, try out some basic matlab functions by typing the lines below, each followed by the 'enter' key. The argument is in the parentheses.

```
>> sin (90)
>> sin(pi/2)
>>sind(90)
>> round(sind(45))
```

Notice that the *reserved* word 'pi' is employed and that the basic trig function require radians though there are trig functions available that work in degrees. Also, in general you may *pass* anything that would *return* a numerical value to any basic function, *e.g* the use of round.

Basic Assignment of Variables

Try out the following lines at your command line.

```
>> a = 1
>> b = 2
>> c = a + b
>> C = A + B
>> d = 5 + c
```

Note that we can only add a and b because they have been previously defined. Also, since A and B are not defined, we can not add them. Yes, matlab is case sensitive.

Variable Types

You have up to this point been working with *scalar* variables, arrays in 1D having a single value. Matlab also supports *n-dimensional arrays.* Type the following:

```
>> d
>> f = [1/ 5 ,23,6,3,-4,34.6,5, 1.3^5]
>> g = [a,b,c;0,0,7;3,2,1]
```

What is the function of the semicolon in this use?

What is the function of the comma? Why did we skip *e* as a variable name? Why do we have *integers* in 'g' and *floatingpoint* numbers in 'f'?

Accessing Variables in Arrays

Use the (row,col) location to retrieve values.

```
>> c
>> g(2,3)
>> f(1,5)
>> g(2,3) + 4*f(1,5)
```

If you have not guessed yet, the comma is used to separate columns and the semicolon is used at the end of a row in matlab matrices. The final semicolon is normally omitted.

You can also assign variables to string values

```
>> h = 'me'
>> k = 'oh'
>> l = 'my'
>> m = ' '
```

```
>> [h, k, l]

>> [h,,m,k,m,l]

>>h+m+k+l
```

Why were *i* and *j* skipped? Type *i* and *j* on your command line.

Useful Matrix Functions

The command 'sort' sorts within columns in ascending order by default. If this is not possible, it defaults to a row-wise sort in ascending order.

```
>> sort(f)

>> sort(g)
```

For a quick description of how sort operates, type 'help sort'. This will work with any command for which matlab has a definition.

Now try these commands, *max, min, mean, std,* operating on *f* and *g*.. What are they doing on *f* and *g*? Also try typing *f'* (note the apostrophe).

Matrix Operations

You can perform basic operations on matrices as long as the indices agree.

```
>> f+2*f

>> g*g

>> g^2

>>f+g

>> f*f'
```

Matlab will try to operate on all elements of a matrix with an operator or a function.

```
>> g-1
>> sin(f)
>> 2*f
```

You may also operate on all elements explicitly.

```
>>f.^2
```

Now

```
>> f^2
>>f.f
>>f.f'
```

Note f' is the transpose of f,f.fis the dot product. Operationg only work when indices agree.

Matrix Assignment

Use the (row,col) location to place values in a matrix.

```
>> mat1 = zeros(4,4)
>> mat1(2,2)=16
>> mat1(1,:)=6
>> mat1(3,:)=[3,2,1,0]
```

What is the purpose of the colon, *:* ?

Basic Plotting

Type the following commands to generate a simple line plot from data.

```
>> x=[1,2,3,4,5,6,7,8,9,10]

>> y=11-x

>> plot(x,y)
```

Where did all the values for *y* come from?

Basic Programming and Using the Editor

Click on the blank sheet icon in the matlab main toolbar. This will open a matlab editor window. Begin typing the remainder of the tutorial in that window. The syntax for basic *for* loops is like other programming languages:

```
for i=1:1:4
        i
        mat1(4,i)=i
pause
end
```

Note that the for loop is closed by an *end* statement. It also runs from 1 by 1 to 4 using the colons to delimit the index definitions (i=1:4 will produce the same result since the default increment is 1). The pause is inserted so that you can see the results.

To run the program, look for the 'evaluate entire file' icon on the toolbar in the editor. Click this. You must hit the space bar to break the pause for each cycle.

By this point you should be familiar with basic matlab commands and assignments for scalar variables and for matrix variables. You should also be comfortable with programming a basic for loop and plotting data.

Basics:

If you type in a valid expression and press Enter, MATLAB will immediately execute it and return the result, just like a calculator.

```
>> 2+2
ans =
4

>> 4^2
ans =
16

>> sin(pi/2)
ans =
1

>> 1/0
Warning: Divide by zero.
ans =
Inf

>>exp(i*pi)
ans =
-1.0000 + 0.0000i
```

Notice some of the special expressions here: pi for □, Inf for ∞, and i for $\sqrt{-1}$. Another special value is NaN, which stands for **not a number**. NaN is used to express an undefined value. For example,

```
>> Inf/Inf      ans = NaN
```

* Challenge: Calculate the following statements: $\rho = \sqrt{1+\sqrt{5}}/2$ and $\left|\sin^{-1}(-1/2)\right|$ and tan(e) (Hint: >>help exp).

Here are a few other demonstration statements.

```
%                       % Anything after a % sign is a comment.
x = rand(2,2);          % ; means "don't print out result"
s = 'Hello world';      % single quotes enclose a string
t = 1 + 2 + 3 + ...     % ... means continue a line
4 + 5 + 6 % ...
```

Here are a few useful commands:

```
who                 % gives you your variables
cd                  % Change current working directory.
pwd                 % Show (print) current working directory.
dir                 % List directory.
ls                  % List directory.
```

General commands you should remember

On-line help

help	lists topics on which help is available
helpwin	opens the interactive help window
helpdesk	opens the web browser-based help facility
helptopic	provides help on topic
lookfor	stringlists help topics containing string
demo	runs the demo program

Workspace information

who	lists variables currently in the workspace
whos	lists variables currently in the workspace with their size

what	lists M- , Mat-, and Mex-files on the disk
clear	clears the workspace, all variables are removed
clear x y z	clears only variables x, y, and z
clear all	clears all variables and functions from workspace
mlock fun	locks function fun so that c lear cannot remove it
munlock fun	unlocks function fun so that c l e ar can remove it
clc	clears command window, cursor moves to the top
home	scrolls the command window to put the curser on top
clf	clears figure window

Directory information

pwd	shows the current working directory
cd	changes the current working directory
dir	lists contents of the current directory
ls	lists contents of the current directory, same as dir
path	gets or sets MATLAB search path
editpath	modifies MATLAB search path
copyfile	copies a file
mkdir	creates a directory

General information

computer	tells you the computer type you are using
clock	gives you wall clock time and date as a vector
date	tells you the date as a string
more	controls the paged output according to the screen size
ver	gives the license and the MATLAB version information
bench	benchmarks your computer on running MATLABcompared to other computers

Termination

Control-c	local abort , kills the current command execution
quit	quits MATLAB
exit	same as quit

Part B (5x8=40 Marks)

Possible Questions

1. Explain about Matlab environment with diagrammatic representation.
2. Describe about Matlab main features with schematic diagram
3. Discuss about the Matlab Windows
4. Write a short note on (i) Matlab Desktop (ii) Fiqure window (iii)Editor window
5. List out advantages and disadvantages in Matlab.
6. Explain in detail about MAT LAB ‘ s features of input and output.
7. Write a short note on (i) Data type (ii) Output display (iii) Command History.
8. Write a brief note on Matlab platform dependence.
9. Explain in detail about Matlabfile types.
10. List out the mat lab general commands with explanation.
11. Describe about entering command and expression in Matlab with an example.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021

Subject: MATLAB programming **Subject Code: 15MMU504**

Class : III - B.Sc. Mathematics **Semester : V**

Unit I

Introduction to MATLAB

Part A (20x1=20 Marks)
(Question Nos. 1 to 20 Online Examinations)

Possible Questions

Question	Choice 1	Choice 2	Choice 3	Choice 4	Answer
what does MATLAB stands for ?	Math Laboratory	Matrix Laboratory	Math Work	Math Language	Matrix Laboratory
what symbol precedes all comments in Matlab ?	"	{	>>	[[	>>
which of the following is not pre-defined variable in Matlab?	pi	inf	i	gravity	gravity
which of the following command is used to clear all data and variables in memory ?	clc	clear	delete	deallocate	clear
_____ character in Matlab are represented in their value in memory.	decimal	hex	ASCII	string	ASCII
In Matlab, this keyword immediately moves to the next iterartion of the loop	update	goto	into	continue	continue
which of the following will correctly define x,y and z as symbols?	syms x y z	sym x,y,z	syms (x,y,z)	sym (x,y,z)	syms x y z

which of the following is used to see if two elements are equal in Matlab ?	:	.=	.==	is equal	.==
To add a comment to the mfile, the Matlab command is _____	%	@	&	(' ')	%
The clc command is used to ____	erase everything in mfile	clear the command window	clean the destop	save the existing mfile	clear the command window
The basic building block o f Matlab is the _____	array	string	matrix	list	matrix
The fundamental data type of Matlab is _____	matrix	array	list	string	array
To find the dimension of an existing matirx in Matlab with _____ command	size	length	dim	eye	size
To find the dimension of an existing a vector in Matlab with _____ command	size	length	dim	eye	length
what command is used to turn case sensitivity on or off in Matlab ?	alp	cases	casesen	sen	casesen
To suppresses the screen output we use _____ at the end of the command	dot	camma	colon	semicolon	semicolon
Mat-files can be loaded into MATLAB with the_____command	load	lod	list	update	load
Typing _____ at the Matlab prompt to print the content of the figure window	eye	print	copy	prt	print
_____ command is used to clear the figure window	clc	clear all	clf	clearf	clf

_____ command is used to change the current working directry	dir	cd	pwd	ls	cd
_____command is used to list a contents of the current directory	ls	pwd	dir	cd	dir
Mat-files are binary ____ files	data	strimg	figure	matrix	data
mkdir command is used to create a _____	script file	directory	function file	m file	directory
clf command is used to clear the _____ window	figure	command	editor	workspace	figure
which of the following is pre-defined variable in Matlab?	weight	pi	gravity	mass	pi
_____ is called a Matlab prompt	"	{	>>	[[	>>
_____ command is listed the M , Mat, Mex-files on the disk	what	who	whos	when	what

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021.
Department of Mathematics

Subject : MATLAB programming	**Subject Code : 15MMU504**	**L T P C**
Class : III – B.Sc. Mathematics	**Semester : V**	**5 0 0 5**

UNIT II

Interactive Computation: Matrices and Vectors – Matrix and Array operations – Creating and Using Inline functions – Using Built-in Functions and On-line Help – Saving and loading data – Plotting simple graphs.

TEXT BOOK

1. RudraPratap, 2003. Getting Started with MATLAB-A Quick Introduction for Scientists and Engineers, Oxford University Press.

REFERENCES

1. William John Palm, 2005. Introduction to Matlab 7 for Engineers, McGraw-Hill Professional.New Delhi.

2. Dolores M. Etter, David C. Kuncicky, 2004.Introduction to MATLAB 7, Prentice Hall, New Delhi.

3. Kiranisingh.Y,Chaudhuri.B.B, 2007.Matlab Programming, Prentice-Hall Of India Pvt.Ltd, New Delhi.

4. Brian Hahn, 2016. Essential MATLAB for Engineers and Scientists.6th edition , Elsevier publication.

UNIT – II
INTERACTIVE COMPUTATION

In principle, one can do all calculations in MATLAB interactively by entering commands sequentially in the command window, although a script file is perhaps a better choice for computations that involve more than a few steps. The interactive mode of computation, however, makes MATLAB a powerful scientific calculator that puts hundreds of built-in mathematical functions for numerical calculations and sophisticated graphics at the fingertips of the user.

In this chapter , we introduce you to some of MATLAB 's built-in functions and capabilities, through examples of interactive computation. The basic things to keep in mind are:

Where to type commands: All MATLAB commands or expressions are entered in the command window at the MATLAB prompt (»).

How to execute commands : To execute a command or statement , you must press return or enter at the end.

What to do if the command is very long: If your command does not fit on one line, you can continue the command on the next line if you type three consecutive periods at the end of the first line. You can keep continuing this way until the length of your command hits the limit , which is 4,096 characters.

How to name variables: Names of variables must begin with a letter. After the first letter, any number of digits or underscores may be used, but MATLAB remembers only the first 31 characters .

What is the precision of computation: All computations are carried out internally in double precision unless specified otherwise. The appearance of numbers on the screen, however, depends on the format in use.

How to control the display format of the output : The output appearance of floating-point numbers (number of digits after the decimal, etc.) is controlled with the format command. The default is format short , which displays four digits after the decimal .

How to suppress the screen output : A semicolon (;) at the end of a command suppresses the screen output , although the command is carried out and the result is saved in the variable assigned to the command or in the default variable ans .

How t o recall previously typed commands: Use the up-arrow key to recall previously typed commands. MATLAB uses smart recall, so you can also type one or two letters of your command and use the up-arrow key for recalling the command starting with those letters. Also, all your commands are recorded in the command history subwindow. You can double click on any command in the command history subwindow to execute it in the command window.

How to set paged-screen display: For paged-screen display (one screenful ofoutput display at a time) use the command more on.
Where and how to save results: If you need to save some of the computed resultsfor later processing, you can save the variables in a file in binary orASCII format with the s ave command.

How to save figures : You can save a figure in a .fig file by selecting F i l e ---> S a veAs from the figure window. Once you save it , you can open it later in MATLABusing open command or by selecting F i l e ---> O pen . from the MATLABmain menu. You can also save figures in one of numerous formats for printingor exporting to other applications.

How to print your work: You can print your entire session in MATLAB, partof it , or selected segments of it , in one of several ways. The simplest way,perhaps, is to create a diary with the di ary command and save your entire session in it . Then you can print the diary just the way you would print any other file on your computer. On PC sand Macs , however, you can print the session by selecting Print from the F i l emenu. (Before you print , make sure that the command window i s the activewindow. If it isn't , just click on the command window to make it active).

What about comments: MATLAB takes anything following a % as a commentand ignores it . 1 You are not likely to use a lot of comments while computinginteractively, but you will use them when you write programs in MATLAB.

Because MATLAB derives most of its power from matrix computations andassumes every variable to be, at least potentially, a matrix, we start with descriptionsand examples of how to enter, index, manipulate, and perform some usefulcalculations with matrices.

MATRICES AND VECTORS

Introduction to Vectors in Matlab

This is the basic introduction to Matlab. Creation of vectors is included with a few basic operations. Topics include the following:

1. **Defining a vector**
2. **Accessing elements within a vector**
3. **Basic operations on vectors**

Defining a Vector

Matlab is a software package that makes it easier for you to enter matrices and vectors, and manipulate them. The interface follows a language that is designed to look a lot like the notation use in linear algebra. In the following tutorial, we will discuss some of the basics of working with vectors.

If you are running windows or Mac OSX, you can start matlab by choosing it from the menu. To start matlab on a unix system, open up a unix shell and type the command to start the software: *matlab*. This will start up the software, and it will wait for you to enter your commands. In the text that follows, any line that starts with two greater than signs (>>) is used to denote the matlab command line. This is where you enter your commands.

Almost all of Matlab's basic commands revolve around the use of vectors. A vector is defined by placing a sequence of numbers within square braces:

```
>> v = [3 1]

v =

     3     1
```

This creates a row vector which has the label "v". The first entry in the vector is a 3 and the second entry is a 1. Note that matlab printed out a copy of the vector after you hit the enter key. If you do not want to print out the result put a semi-colon at the end of the line:

```
>> v = [3 1];
>>
```

If you want to view the vector just type its label:

```
>> v

v =

     3     1
```

You can define a vector of any size in this manner:

```
>> v = [3 1 7 -21 5 6]

v =

     3     1     7    -21     5     6
```

Notice, though, that this always creates a row vector. If you want to create a column vector you need to take the transpose of a row vector. The transpose is defined using an apostrophe ("'"):

```
>> v = [3 1 7 -21 5 6]'

v =

     3
     1
     7
```

```
   -21
     5
     6
```

A common task is to create a large vector with numbers that fit a repetitive pattern. Matlab can define a set of numbers with a common increment using colons. For example, to define a vector whose first entry is 1, the second entry is 2, the third is three, up to 8 you enter the following:

```
>> v = = [1:8]

v =

     1     2     3     4     5     6     7     8
```

If you wish to use an increment other than one that you have to define the start number, the value of the increment, and the last number. For example, to define a vector that starts with 2 and ends in 4 with steps of .25 you enter the following:

```
>> v = [2:.25:4]

v =

  Columns 1 through 7

    2.0000    2.2500    2.5000    2.7500    3.0000    3.2500    3.5000

  Columns 8 through 9

    3.7500    4.0000
```

Accessing elements within a vector

You can view individual entries in this vector. For example to view the first entry just type in the following:

```
>> v(1)

ans =

     2
```

This command prints out entry 1 in the vector. Also notice that a new variable called *ans* has been created. Any time you perform an action that does not include an assignment matlab will put the label *ans* on the result.

To simplify the creation of large vectors, you can define a vector by specifying the first entry, an increment, and the last entry. Matlab will automatically figure out how many entries you need

and their values. For example, to create a vector whose entries are 0, 2, 4, 6, and 8, you can type in the following line:

```
>> 0:2:8

ans =

     0     2     4     6     8
```

Matlab also keeps track of the last result. In the previous example, a variable "ans" is created. To look at the transpose of the previous result, enter the following:

```
>> ans'

ans =

     0
     2
     4
     6
     8
```

To be able to keep track of the vectors you create, you can give them names. For example, a row vector v can be created:

```
>> v = [0:2:8]

v =

     0     2     4     6     8

>> v

v =

     0     2     4     6     8

>> v;
>> v'

ans =

     0
     2
     4
     6
     8
```

Note that in the previous example, if you end the line with a semi-colon, the result is not displayed. This will come in handy later when you want to use Matlab to work with very large systems of equations.

Matlab will allow you to look at specific parts of the vector. If you want to only look at the first three entries in a vector you can use the same notation you used to create the vector:

```
>> v(1:3)

ans =

     0     2     4

>> v(1:2:4)

ans =

     0     4

>> v(1:2:4)'

ans =

     0
     4
```

Basic operations on vectors

Once you master the notation you are free to perform other operations:

```
>> v(1:3)-v(2:4)

ans =

    -2    -2    -2
```

For the most part Matlab follows the standard notation used in linear algebra. We will see later that there are some extensions to make some operations easier. For now, though, both addition subtraction are defined in the standard way. For example, to define a new vector with the numbers from 0 to -4 in steps of -1 we do the following:

```
>> u = [0:-1:4]
u = [0:-1:-4]

u =

     0    -1    -2    -3    -4
```

We can now add u and v together in the standard way:

```
>> u+v

ans =
```

```
     0     1     2     3     4
```

Additionally, scalar multiplication is defined in the standard way. Also note that scalar division is defined in a way that is consistent with scalar multiplication:

```
>> -2*u

ans =

     0     2     4     6     8

>> v/3

ans =

         0    0.6667    1.3333    2.0000    2.6667
```

With these definitions linear combinations of vectors can be easily defined and the basic operations combined:

```
>> -2*u+v/3

ans =

         0    2.6667    5.3333    8.0000   10.6667
```

You will need to be careful. These operations can only be carried out when the dimensions of the vectors allow it. You will likely get used to seeing the following error message which follows from adding two vectors whose dimensions are different:

```
>> u+v'
??? Error using ==> plus
Matrix dimensions must agree.
```

Introduction to Matrices in Matlab

A basic introduction to defining and manipulating matrices is given here. It is assumed that you know the basics on how to define and manipulate **vectors** using matlab.

1. **Defining Matrices**
2. **Matrix Functions**
3. **Matrix Operations**

Defining Matrices

Defining a matrix is similar to defining a **vector**. To define a matrix, you can treat it like a column of row vectors (note that the spaces are required!):

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

A =

     1     2     3
     3     4     5
     6     7     8
```

You can also treat it like a row of column vectors:

```
>> B = [ [1 2 3]' [2 4 7]' [3 5 8]']

B =

     1     2     3
     2     4     5
     3     7     8
```

(Again, it is important to include the spaces.)

If you have been putting in variables through this and the tutorial on **vectors**, then you probably have a lot of variables defined. If you lose track of what variables you have defined, the *whos* command will let you know all of the variables you have in your work space.

```
>> whos
  Name      Size                    Bytes  Class

  A         3x3                        72  double array
  B         3x3                        72  double array
  v         1x5                        40  double array

Grand total is 23 elements using 184 bytes
```

We assume that you are doing this tutorial after completing the previous tutorial. The vector v was defined in the previous tutorial.

As mentioned before, the notation used by Matlab is the standard linear algebra notation you should have seen before. Matrix-vector multiplication can be easily done. You have to be careful, though, your matrices and vectors have to have the right size!

```
>> v = [0:2:8]

v =

     0     2     4     6     8

>> A*v(1:3)
??? Error using ==> *
Inner matrix dimensions must agree.

>> A*v(1:3)'

ans =

    16
    28
    46
```

Get used to seeing that particular error message! Once you start throwing matrices and vectors around, it is easy to forget the sizes of the things you have created.

You can work with different parts of a matrix, just as you can with vectors. Again, you have to be careful to make sure that the operation is legal.

```
>> A(1:2,3:4)
???  Index exceeds matrix dimensions.

>> A(1:2,2:3)

ans =

     2     3
     4     5

>> A(1:2,2:3)'

ans =

     2     4
     3     5
```

Matrix Functions

Once you are able to create and manipulate a matrix, you can perform many standard operations on it. For example, you can find the inverse of a matrix. You must be careful, however, since the operations are numerical manipulations done on digital computers. In the example, the matrix A is not a full matrix, but matlab's inverse routine will still return a matrix.

```
>> inv(A)

Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 4.565062e-18


ans =

   1.0e+15 *

   -2.7022    4.5036   -1.8014
    5.4043   -9.0072    3.6029
   -2.7022    4.5036   -1.8014
```

By the way, Matlab is case sensitive. This is another potential source of problems when you start building complicated algorithms.

```
>> inv(a)
??? Undefined function or variable a.
```

Other operations include finding an approximation to the eigen values of a matrix. There are two versions of this routine, one just finds the eigen values, the other finds both the eigen values and the eigen vectors. If you forget which one is which, you can get more information by typing *help eig* at the matlab prompt.

```
>> eig(A)

ans =

   14.0664
   -1.0664
    0.0000

>> [v,e] = eig(A)

v =

   -0.2656    0.7444   -0.4082
   -0.4912    0.1907    0.8165
   -0.8295   -0.6399   -0.4082


e =
```

```
   14.0664         0         0
         0   -1.0664         0
         0         0    0.0000

>> diag(e)

ans =

   14.0664
   -1.0664
    0.0000
```

Matrix Operations

There are also routines that let you find solutions to equations. For example, if Ax=b and you want to find x, a slow way to find x is to simply invert A and perform a left multiply on both sides (more on that later). It turns out that there are more efficient and more stable methods to do this (L/U decomposition with pivoting, for example). Matlab has special commands that will do this for you.

Before finding the approximations to linear systems, it is important to remember that if A and B are both matrices, then AB is not necessarily equal to BA. To distinguish the difference between solving systems that have a right or left multiply, Matlab uses two different operators, "/" and "\". Examples of their use are given below. It is left as an exercise for you to figure out which one is doing what.

```
>> v = [1 3 5]'

v =

     1
     3
     5

>> x = A\v

Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 4.565062e-18


x =

   1.0e+15 *

    1.8014
   -3.6029
    1.8014

>> x = B\v

x =
```

```
     2
     1
    -1

>> B*x

ans =

     1
     3
     5

>> x1 = v'/B

x1 =

    4.0000   -3.0000    1.0000


>> x1*B

ans =

    1.0000    3.0000    5.0000
```

Finally, sometimes you would like to clear all of your data and start over. You do this with the "clear" command. Be careful though, it does not ask you for a second opinion and its results are **final**.

```
>> clear

>> whos
```

Vector Functions

Matlab makes it easy to create vectors and matrices. The real power of Matlab is the ease in which you can manipulate your vectors and matrices. Here we assume that you know the basics of defining and manipulating vectors and matrices. In particular we assume that you know how to create vectors and matrices and know how to index into them. For more information on those topics see our tutorial on either **vectors** or **matrices**.

In this tutorial we will first demonstrate simple manipulations such as addition, subtraction, and multiplication. Following this basic "element-wise" operations are discussed. Once these operations are shown, they are put together to demonstrate how relatively complex operations can be defined with little effort.

First, we will look at simple addition and subtraction of vectors. The notation is the same as found in most linear algebra texts. We will define two vectors and add and subtract them:

```
>> v = [1 2 3]'

v =

     1
     2
     3

>> b = [2 4 6]'

b =

     2
     4
     6

>> v+b

ans =

     3
     6
     9

>> v-b

ans =

    -1
    -2
    -3
```

Multiplication of vectors and matrices must follow strict rules. Actually, so must addition. In the example above, the vectors are both column vectors with three entries. You cannot add a row vector to a column vector. Multiplication, though, can be a bit trickier. The number of columns of the thing on the left must be equal to the number of rows of the thing on the right of the multiplication symbol:

```
>> v*b
??? Error using ==> *
Inner matrix dimensions must agree.

>> v*b'

ans =

     2     4     6
     4     8    12
     6    12    18

>> v'*b

ans =
```

```
    28
```

There are many times where we want to do an operation to every entry in a vector or matrix. Matlab will allow you to do this with "element-wise" operations. For example, suppose you want to multiply each entry in vector v with its cooresponding entry in vector b. In other words, suppose you want to find v(1)*b(1), v(2)*b(2), and v(3)*b(3). It would be nice to use the "*" symbol since you are doing some sort of multiplication, but since it already has a definition, we have to come up with something else. The programmers who came up with Matlab decided to use the symbols ".*" to do this. In fact, you can put a period in front of any math symbol to tell Matlab that you want the operation to take place on each entry of the vector.

```
>> v.*b

ans =

     2
     8
    18

>> v./b

ans =

    0.5000
    0.5000
    0.5000
```

Since we have opened the door to non-linear operations, why not go all the way? If you pass a vector to a predefined math function, it will return a vector of the same size, and each entry is found by performing the specified operation on the cooresponding entry of the original vector:

```
>> sin(v)

ans =

    0.8415
    0.9093
    0.1411

>> log(v)

ans =

         0
    0.6931
    1.0986
```

The ability to work with these vector functions is one of the advantages of Matlab. Now complex operations can be defined that can be done quickly and easily. In the following example a very

large vector is defined and can be easily manipulated. (Notice that the second command has a ";" at the end of the line. This tells Matlab that it should not print out the result.)

```
>> x = [0:0.1:100]

x =

  Columns 1 through 7

         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000

    [stuff deleted]

  Columns 995 through 1001

   99.4000   99.5000   99.6000   99.7000   99.8000   99.9000  100.0000


>> y = sin(x).*x./(1+cos(x));
```

Through this simple manipulation of vectors, Matlab will also let you graph the results. The following example also demonstrates one of the most useful commands in Matlab, the "help" command.

```
>> plot(x,y)
>> plot(x,y,'rx')
>> help plot

 PLOT   Linear plot.
    PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up.  If X is a scalar and Y is a vector, length(Y)
    disconnected points are plotted.

    PLOT(Y) plots the columns of Y versus their index.
    If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
    In all other uses of PLOT, the imaginary part is ignored.

    Various line types, plot symbols and colors may be obtained with
    PLOT(X,Y,S) where S is a character string made from one element
    from any or all the following 3 columns:

           b     blue          .     point              -     solid
           g     green         o     circle             :     dotted
r     red           x     x-mark             -.    dashdot
           c     cyan          +     plus               --    dashed
           m     magenta       *     star
           y     yellow        s     square
           k     black         d     diamond
                               v     triangle (down)
                               ^     triangle (up)
<     triangle (left)
>     triangle (right)
                               p     pentagram
```

```
                                h     hexagram

    For example, PLOT(X,Y,'c+:') plots a cyan dotted line with a plus
    at each data point; PLOT(X,Y,'bd') plots blue diamond at each data
    point but does not draw any line.

    PLOT(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) combines the plots defined by
    the (X,Y,S) triples, where the X's and Y's are vectors or matrices
    and the S's are strings.

    For example, PLOT(X,Y,'y-',X,Y,'go') plots the data twice, with a
    solid yellow line interpolating green circles at the data points.

    The PLOT command, if no color is specified, makes automatic use of
    the colors specified by the axes ColorOrder property.  The default
    ColorOrder is listed in the table above for color systems where the
    default is blue for one line, and for multiple lines, to cycle
    through the first six colors in the table.  For monochrome systems,
    PLOT cycles over the axes LineStyleOrder property.

    PLOT returns a column vector of handles to LINE objects, one
    handle per line.

    The X,Y pairs, or X,Y,S triples, can be followed by
    parameter/value pairs to specify additional properties
    of the lines.

    See also SEMILOGX, SEMILOGY, LOGLOG, PLOTYY, GRID, CLF, CLC, TITLE,
    XLABEL, YLABEL, AXIS, AXES, HOLD, COLORDEF, LEGEND, SUBPLOT, STEM.

 Overloaded methods
    help idmodel/plot.m
    help iddata/plot.m


>> plot(x,y,'y',x,y,'go')
>> plot(x,y,'y',x,y,'go',x,exp(x+1),'m--')
>> whos
  Name      Size                    Bytes  Class

  ans       3x1                        24  double array
  b         3x1                        24  double array
  v         3x1                        24  double array
  x         1x1001                   8008  double array
  y         1x1001                   8008  double array

Grand total is 2011 elements using 16088 bytes
```

The compact notation will let you tell the computer to do lots of calculations using few commands. For example, suppose you want to calculate the divided differences for a given equation. Once you have the grid points and the values of the function at those grid points, building a divided difference table is simple:

```
>> coef = zeros(1,1001);
>> coef(1) = y(1);
>> y = (y(2:1001)-y(1:1000))./(x(2:1001)-x(1:1000));
>> whos
  Name       Size                    Bytes  Class

  ans        3x1                        24  double array
  b          3x1                        24  double array
  coef       1x1001                   8008  double array
  v          3x1                        24  double array
  x          1x1001                   8008  double array
  y          1x1000                   8000  double array

Grand total is 3008 elements using 24064 bytes

>> coef(2) = y(1);
>> y(1)

ans =

    0.0500

>> y = (y(2:1000)-y(1:999))./(x(3:1001)-x(1:999));
>> coef(3) = y(1);
>>
>>
```

From this algorithm you can find the Lagrange polynomial that interpolates the points you defined above (vector x). Of course, with so many points, this might get a bit tedious.

Character Strings:

In MATLAB, these are arrays of ASCII values that are displayed as their character string representation. For example:

```
>> t = 'Hello'
t =
Hello
>> size(t)
ans =
     1     5
```

A character string is simply text surrounded by single quotes. Each character in a string is one element in the array. To see the underlying ASCII representation of a character string, you can type,

```
>> double(t)
ans =
```

```
   104   101   108   108   111
```

The function char provides the reverse transformation:

```
>> char(t)
ans =
hello
```

Since strings are numerical arrays with special attributes, they can be manipulated just like vectors or matrices. For example,

```
>> u=t(2:4)
u =
ell
>> u=t(5:-1:1)
u =
olleh
>> u=t(2:4)'
u =
e
l
l
```

One can also concatenate strings directly. For instance,

```
>> u='My name is ';
>> v='Mr. MATLAB';
>> w=[u v]
w =
My name is Mr. MATLAB
```

The function `disp` allows you to display a string without printing its variable name. For example:

```
>> disp(w)
My name is Mr. MATLAB
```

In many situations, it is desirable to embed a numerical result within a string. The following string conversion performs this task.

```
>> radius=10; volume=(4/3)*pi*radius^3;
```

```
>> t=['A sphere of radius ' num2str(radius) ' has volume… of '
num2str(volume) '.'];
>> disp(t)
```

It may sometimes be required to find a certain part of a longer string. For example,

```
>>a='TexasTechUniversity';
>>findstr(a,' ')   %% Finds spaces
ans =
  6  6  11
>>findstr(a,'Tech')  %% Finds the string Tech
ans
  7
>>findstr(a,'Te')  %% Finds the string starting with Te
ans =
     1     7
>>findstr(a,'tech')  %% This command is case-sensitive
ans =
   [  ]
```

If it is desired to replace all the case on Tech to TECH then one can do this by using,

```
>>strrep(a,'Tech','TECH')
ans =
TexasTECHUniversity
```

Relational and Logical Operators/Functions

Every time you create an M-file, you are writing a computer program using the MATLAB programming language. A **logical** is a variable which is assigned to a relational or logical expression .

```
>> a = true
a =
1
>> b = false
b =
0
```

Relational operators in MATLAB

Relational operators are used to compare two arrays of the same size or to compare an array to a scalar. In the second case, the scalar is compared with all elements of the array and the result has

the same size as the array. A statement that includes a relational operator is called a logical expression either true or false. If the statement is true, it is assigned a value of 1, and a value of 0 when it is wrong.

MATLAB's relational operators are

```
==  equal
~=  not equal
<   less than
>    greater than
<=  less than or equal
>=  greater than or equal
```

Note that a single = is different than double == and denotes assignment and never a test for equality in MATLAB.

```
>> A=1:5, B=2.*A-1
A =
1 2 3 4 5
B =
1 3 5 7 9
>> compAB = A <B
compAB =
0 1 1 1 1
>> compAB2= A == B
compAB2 =
1 0 0 0 0
```

Logical operators and find command

& (logical and) operator takes two logical expressions and returns true if both expressions are true, and false otherwise.

| (logical or) operator takes two logical expressions and returns true if either of the expressions are true, and false only if both expressions are false.

~ (logical not) operator takes only one logical expression and returns the opposite (negation) of that expression.

Relational and Logical functions in MATLAB: There are many useful logical functions whose names begin with is. The results of MATLAB's logical operators and logical functions are logical arrays of 0s and 1s.

isequal (A,B): To test whether arrays A and B are equal, that is, of the same size with identical elements, the expression can be used:

```
>> isequal (A,B)
ans=
0
```

isempty: test for empty array
isequal: test if arrays are equal
isfinite: detect finite array elements
isinf: detect infinite array elements
isinteger: test for integer array
issorted: test for sorted vector

Other important logical functions are: ischar, isequalwithequalnans, isfloat, islogical, isnan, isnumeric, isreal, isscalar, isvector.

Other important logical functions are ***all, any,*** and, ***find*** for specifying nonzero elements of arrays:

all returns true if all elements of vector is nonzero

any returns true if any element of vector is nonzero

find command also can be used to extract the nonzero elements of an array:

```
>> x = [ -3 1 0 -inf OJ ;
>> f = find(x)
f =
1 2 4
>> x(f)
ans=
-3 1 –Inf
>> x(find(isfinite(x)))
ans=
-3
1
0
0
```

Remark (Operator Precedence)

Arithmetic, relational, and logical operators can all be combined in mathematical expressions. When an expression has such a combination, the result depends on the order in which the operations are carried out. The following is the order used by MATLAB:

```
(highest) Parenthese
Exponentiation
Logical NOT (~)
Multiplication, division
Addition, subtraction
Relational operators (>,<,>=,<=,==,~=)
Logical AND (&)
(lowest) Logical OR (|)
```

MATLAB Functions

In MATLAB you will use built-in functions as well as functions that you create yourself. MATLAB has many built-in functions, typing ***help elfun*** and/or ***help specfun*** calls up full lists of elementary and special functions. These include ***sqrt, cos, sin, tan, log,*** and, ***exp***.

For the user-defined functions, you can use ***inline ('function','independent variable')*** command:

```
>> f = inline('xˆ2 + 2*x + 1', 'x')
f =
Inline function:
f(x) = x^2 + 2*x + 1
>> f(4)      % Once the function is defined, you can evaluate it
ans =
```

```
25
```

Symbolic Computation in MATLAB

You can carry out algebraic or symbolic calculations in MATLAB, such as simplifying polynomials, differentiation with **diff** function, integration with **int** function or solving algebraic equations. To find out about the int function, for example, from the Command Window:

>> help sym/int

To perform symbolic computations, use ***syms*** to declare the variables you plan to use as symbolic variables. Some of the important functions are: ***simplify, subs, solve, diff*** and ***int***.

```
>> syms x y
>> (x-y)*(x+y)*(x^2+2*x+1)
ans =
(x + y)*(x - y)*(x^2 + 2*x + 1)
>> f=simplify((x-y)*(x+y)*(x^2+2*x+1))      % to simplify the expression
f =
(x^2 - y^2)*(x + 1)^2
>> subs(f, x, 2)       % to substite x=2 in f
ans =
36 - 9*y^2
>> solve(f) % to solve f=0 with respect to x
ans =
 y
-1
-1
-y
>> f1=diff(f,x)  % to differentiate f with respect to x
f1 =
 (2*x + 2)*(x^2 - y^2) + 2*x*(x + 1)^2
>> f2=int(f,x)  % to integrate f with respect to x
f2 =
x^4/2 - x*y^2 - x^3*(y^2/3 - 1/3) - x^2*y^2 + x^5/5
```

Input/Output

It is possible to write programmes that accept input from the user and produce informative output. Statements that are called input/output statements are used for these tasks with MATLAB functions ***input*** and ***fprintf.***

```
>> rad = input('Enter the radius: ')
Enter the radius: 5
rad =
      5
>>name=input('Enter your name: ')   % to enter characters
Enter your name: 'basak'
Name=
       basak
>> fprintf('The value of six square is %d\n',36)
The value of six square is 36
```

```
>> fprintf('Six square is %3d and the square root of 2 is %6.2f\n',36,1.47)
Six square is  36 and the square root of 2 is     1.47
```

The character '\n' at the end of the string is a special character called the newline character; when it is printed the output moves down to the next line. The %d in fprintf is sometimes called a placeholder; it specifies where the value of the expression that is after the string is to be printed. The character in the placeholder is called the conversion character, and it specifies the type of value that is being printed. A list of the simple placeholders:

```
%d integers (it actually stands for decimal integer)
%f floats
%c single characters
%s strings
```

USING BUILT- IN FUNCTIONS AND ON-LINE HELP

MATLAB provides hundreds of built-in functions for numerical linear algebra, data analysis, Fourier transforms, data interpolation and r,urve fitting, root-finding , numericalsolution of ordinary differential equations, numerical quadrature, sparsematrix calculations, and general-purpose graphics. There is on-line help for allbuilt-in functions. With so many built-in functions , it is import ant to know howto look for functions and how to learn to use them. There are several ways to gethelp :

help the most direct on-line help: If you know the exact name of a function, you can get help on it by typing help functionname on the command line.For example, typing help help provides help on the function help itself.

lookfor the keyword search function: If you are looking for a function, uselookfor keyword to get a list of functions with the string key'word in their description.For example, typing lookfor ' identity matrix ' lists functions(there are two of them) that create identity matrices.

helpwin the click and navigate help: If you want to look around and get afeel for the on-line help by clicking and navigating through what catches your attention, use the window- help, helpwin, To activate the help window,type helpwin at the command prompt or select Help Window from the Helpmenu on the command window menu bar.

helpdesk the web browser-based help: MATLAB provides extensive on-linedocumentation in both HTML and PDF formats , If you like to read on-linedocumentation and get detailed help by clicking on hyperlinked text , use theweb browser-based help facility, helpdesk, To activate the help window, clickon the help icon. on the menu bar. Alternatively, type helpdesk at the commandprompt or select Product Help from the Help menu on the commandwindow menu bar

As you work more in MATLAB , you will realize that the on-line help with thecommand help and keyword search with lookfor are the easiest and fastest waysto get help.

Plotting

we will introduce the basic operations for creating plots. To show how the *plot* command is used, an approximation using Euler's Method is found and the results plotted. We will approximate the solution to the D.E. y'= 1/y, y(0)=1. A step size of h=1/16 is specified and Euler's Method is used. Once done, the true solution is specified so that we can compare the approximation with the true value. (This example comes from the tutorial on **loops**.)

```
>> h = 1/16;
>> x = 0:h:1;
>> y = 0*x;
>> size(y)

ans =

     1    17

>> max(size(y))

ans =

    17

>> y(1) = 1;
>> for i=2:max(size(y)),
     y(i) = y(i-1) + h/y(i-1);
   end
>> true = sqrt(2*x+1);
```

Now, we have an approximation and the true solution. To compare the two, the true solution is plotted with the approximation plotted at the grid points as a green 'o'. The *plot* command is used to generate plots in matlab. There is a wide variety of arguments that it will accept. Here we just want one plot, so we give it the range, the domain, and the format.

```
>> plot(x,y,'go',x,true)
```

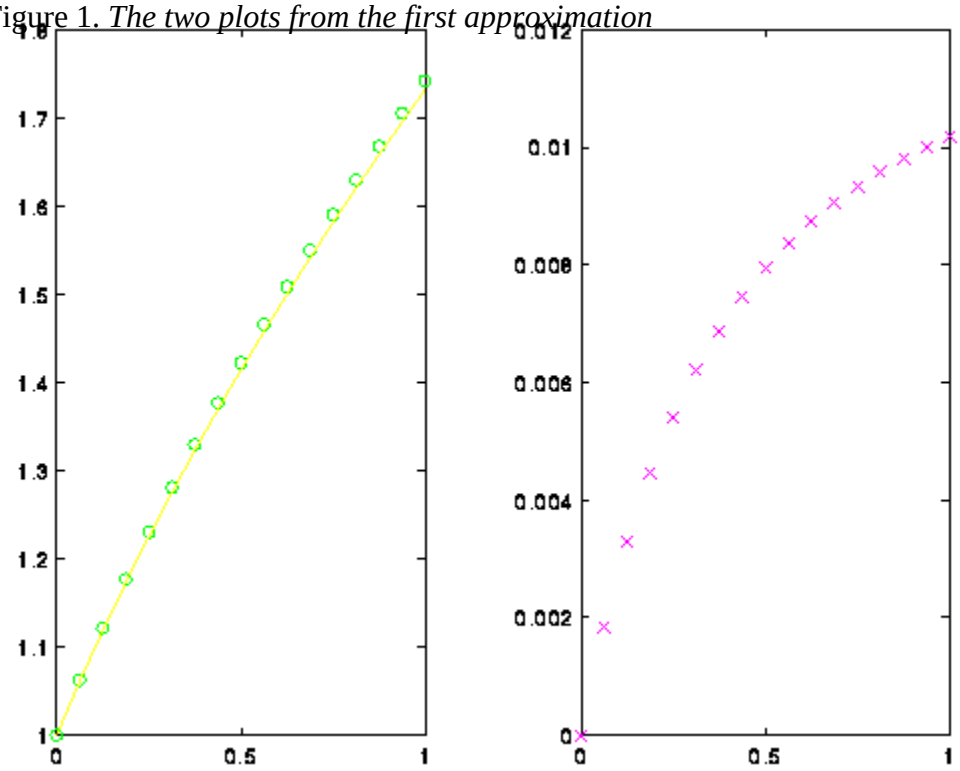
That's nice, but it would also be nice to plot the error:

```
>> plot(x,abs(true-y),'mx')
```

Okay, let's print everything on one plot. To do this, you have to tell matlab that you want two plots in the picture. This is done with the *subplot* command. Matlab can treat the window as an array of plots. Here we will have one row and two columns giving us two plots. In plot #1 the function is plotted, while in plot #2 the error is plotted.

```
>> subplot(1,2,1);
>> plot(x,y,'go',x,true)
>> subplot(1,2,2);
>> plot(x,abs(true-y),'mx')
```

Figure 1. *The two plots from the first approximation*



Let's start over. A new approximation is found by cutting the step size in half. But first, the picture is completely cleared and reset using the *clf* comand. (Note that I am using new vectors *x1* and *y1*.)

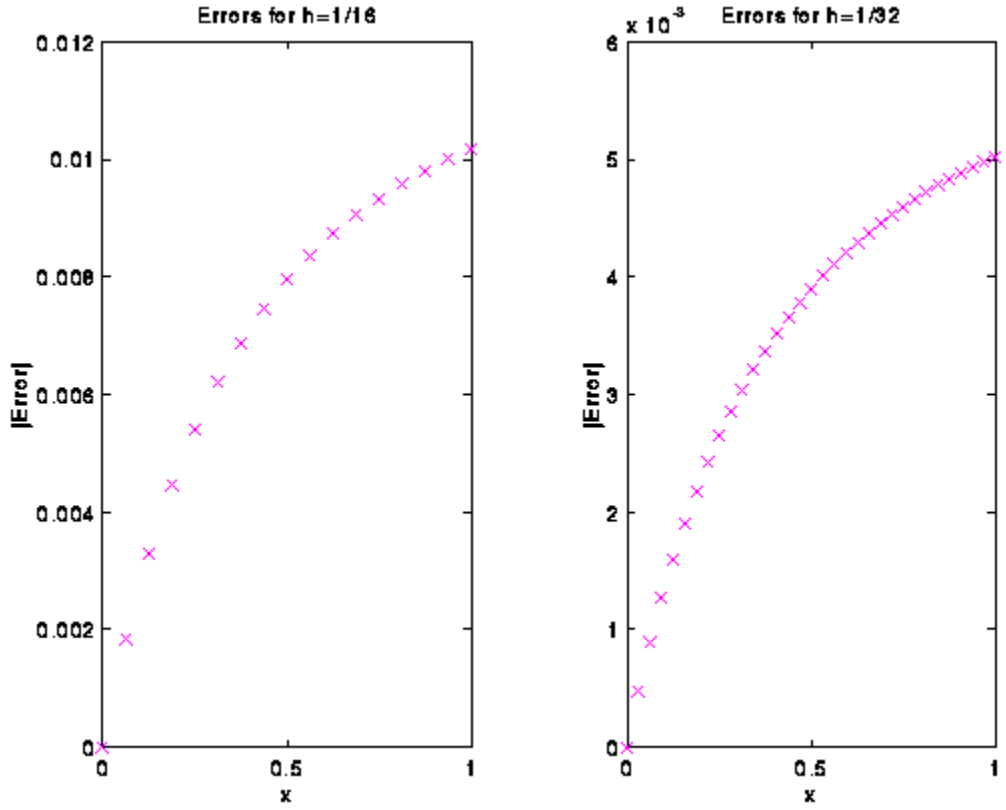
```
>> clf
>> h = h/2;
>> x1 = 0:h:1;
>> y1 = 0*x1;
>> y1(1) = 1;
>> for i=2:max(size(y1)),
     y1(i) = y1(i-1) + h/y1(i-1);
   end
>> true1 = sqrt(2*x1+1);
```

The new approximation is plotted, but be careful! The vectors passed to *plot* have to match. The labels are given for the axis and a title is given to each plot in the following example. The following example was chosen to show how you can use the *subplot* command to cycle through the plots at any time.

```
>> plot(x,y1,'go',x,true1)
??? Error using ==> plot
Vectors must be the same lengths.

>> plot(x1,y1,'go',x1,true1)
>> plot(x1,abs(true1-y1),'mx')
>> subplot(1,2,1);
```

```
>> plot(x,abs(true-y),'mx')
>> subplot(1,2,2);
>> plot(x1,abs(true1-y1),'mx')
>> title('Errors for h=1/32')
>> xlabel('x');
>> ylabel('|Error|');
>> subplot(1,2,1);
>> xlabel('x');
>> ylabel('|Error|');
>> title('Errors for h=1/16')
```

Figure 2. *The errors for the two approximations*



Finally, if you want to print the plot, you must first print the plot to a file. To print a postscript file of the current plot you can use the *print* command. The following example creates a postscript file called error.ps which resides in the current directory. This new file (error.ps) can be printed from the UNIX prompt using the lpr command.

```
>> print -dps error.ps
```

Creating a Plot

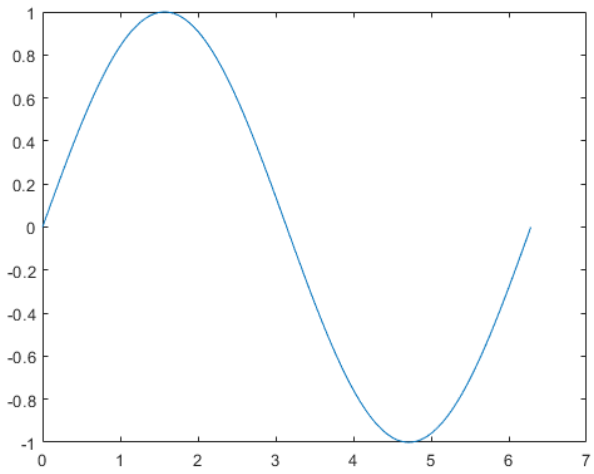
Try this Example

The `plot` function has different forms, depending on the input arguments.

- If `y` is a vector, `plot(y)` produces a piecewise linear graph of the elements of `y` versus the index of the elements of `y`.
- If you specify two vectors as arguments, `plot(x,y)` produces a graph of `y` versus `x`.

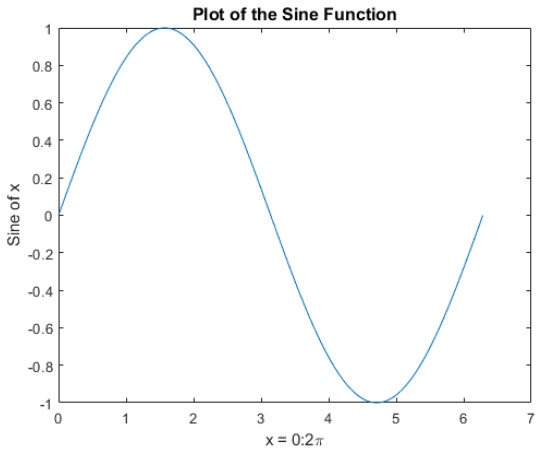
Use the colon operator to create a vector of `x` values ranging from `0` to 2π, compute the sine of these values, and plot the result.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```



Add axis labels and a title. The characters `\pi` in the `xlabel` function create the symbol π. The `FontSize` property in the `title` function increases the size the text used for the title.

```
xlabel('x = 0:2\pi')
ylabel('Sine of x')
title('Plot of the Sine Function','FontSize',12)
```



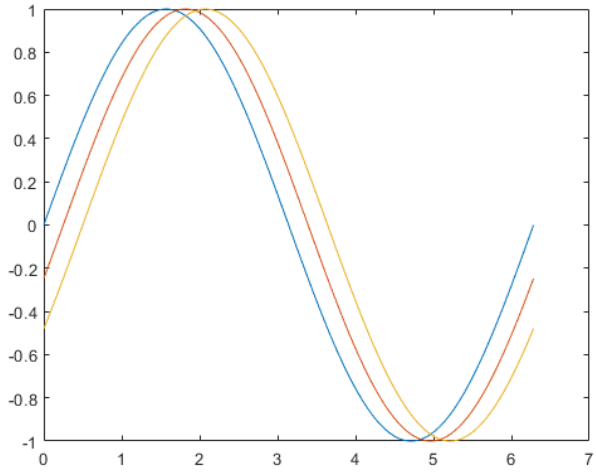
Plotting Multiple Data Sets in One Graph

Try this Example

Multiple `x-y` pair arguments create multiple graphs with a single call to `plot`. MATLAB® uses a different color for each line.

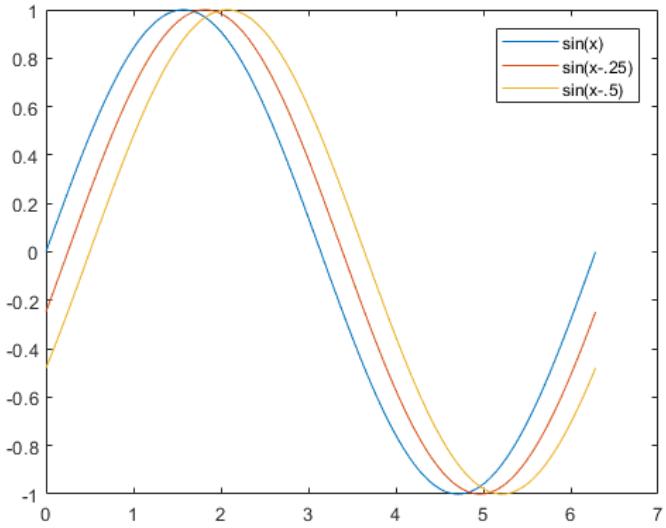
For example, these statements plot three related functions of `x` :

```
x = 0:pi/100:2*pi;
y = sin(x);
y2 = sin(x-.25);
y3 = sin(x-.5);
plot(x,y,x,y2,x,y3)
```



The `legend` function provides an easy way to identify the individual lines:

```
legend('sin(x)','sin(x-.25)','sin(x-.5)')
```



Specifying Line Styles and Colors

It is possible to specify color, line styles, and markers (such as plus signs or circles) when you plot your data using the `plot` command:

```
plot(x,y,'color_style_marker')
```

color_style_marker contains one to four characters (enclosed in single quotes) constructed from a color, a line style, and a marker type. For example,

```
plot(x,y,'r:+')
```

plots the data using a red-dotted line and places a + marker at each data point.

color_style_marker is composed of combinations of the following elements.

Type	Values	Meanings
Color	'c' 'm' 'y' 'r' 'g' 'b' 'w' 'k'	cyan magenta yellow red green blue white black
Line style	'-' '--' ':' '-.' no character	solid dashed dotted dash-dot no line

Type	Values	Meanings
Marker type	'+' 'o' '*' 'x' 's' 'd' '^' 'v' '>' '<' 'p' 'h' no character	plus mark unfilled circle asterisk letter x filled square filled diamond filled upward triangle filled downward triangle filled right-pointing triangle filled left-pointing triangle filled pentagram filled hexagram no marker

Plotting Lines and Markers

If you specify a marker type, but not a line style, MATLAB® creates the graph using only markers, but no line. For example,

```
plot(x,y,'ks')
```

plots black squares at each data point, but does not connect the markers with a line.

The statement

```
plot(x,y,'r:+')
```

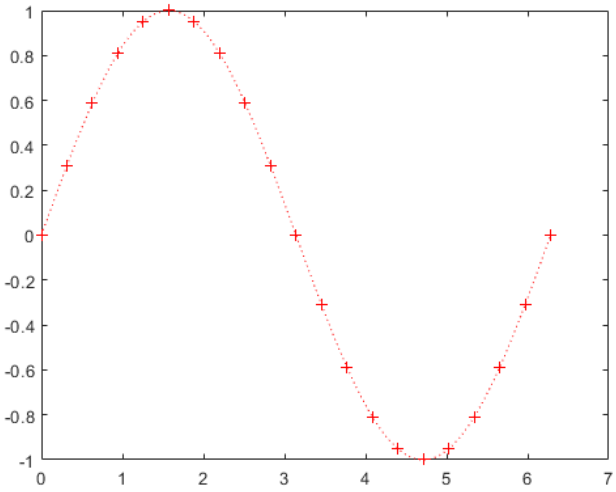
plots a red-dotted line and places plus sign markers at each data point.

Placing Markers at Every Tenth Data Point

Try this Example

This example shows how to use fewer data points to plot the markers than you use to plot the lines. This example plots the data twice using a different number of points for the dotted line and marker plots.

```
x1 = 0:pi/100:2*pi;
x2 = 0:pi/10:2*pi;
plot(x1,sin(x1),'r:',x2,sin(x2),'r+')
```



Graphing Imaginary and Complex Data

When you pass complex values as arguments to `plot`, MATLAB ignores the imaginary part, *except* when you pass a single complex argument. For this special case, the command is a shortcut for a graph of the real part versus the imaginary part. Therefore,

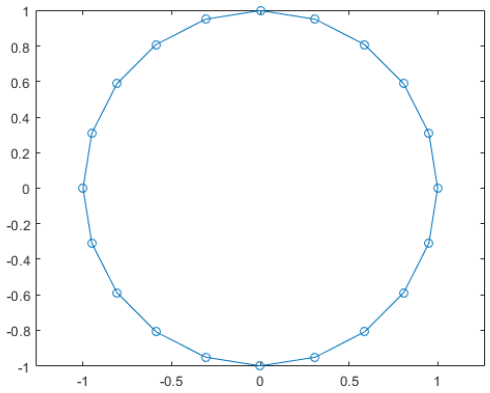
```
plot(Z)
```

where `Z` is a complex vector or matrix, is equivalent to

```
plot(real(Z),imag(Z))
```

The following statements draw a 20-sided polygon with little circles at the vertices.

```
t = 0:pi/10:2*pi;
plot(exp(i*t),'-o')
axis equal
```



The `axis equal` command makes the individual tick-mark increments on the *x*- and *y*-axes the same length, which makes this plot more circular in appearance.

Adding Plots to an Existing Graph

The `hold` command enables you to add plots to an existing graph. When you type,

```
hold on
```

MATLAB does not replace the existing graph when you issue another plotting command. Instead, MATLAB combines the new graph with the current graph.
For example, these statements first create a surface plot of the `peaks` function, then superimpose a contour plot of the same function.

```
[x,y,z] = peaks;
% Create surface plot
surf(x,y,z)
% Remove edge lines a smooth colors
shading interp
% Hold the current graph
hold on
% Add the contour graph to the pcolor graph
contour3(x,y,z,20,'k')
% Return to default
hold off
```

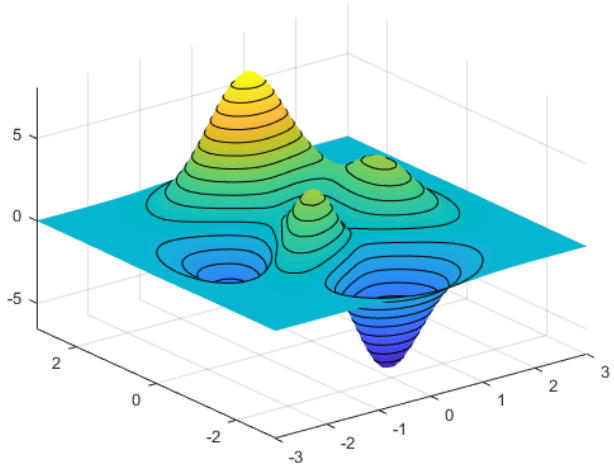


Figure Windows

Plotting functions automatically open a new figure window if there are no figure windows already created. If there are multiple figure windows open, MATLAB uses the one that is designated as the “current figure” (usually, the last figure used).

To make an existing figure window the current figure, you can click the mouse while the pointer is in that window or you can type,

```
figure(n)
```

where `n` is the number in the figure title bar.

To open a new figure window and make it the current figure, type

```
figure
```

Clearing the Figure for a New Plot

When a figure already exists, most plotting commands clear the axes and use this figure to create the new plot. However, these commands do not reset figure properties, such as the background color or the colormap. If you have set any figure properties in the previous plot, you can use the `clf` command with the `reset` option,

```
clf reset
```

before creating your new plot to restore the figure's properties to their defaults.

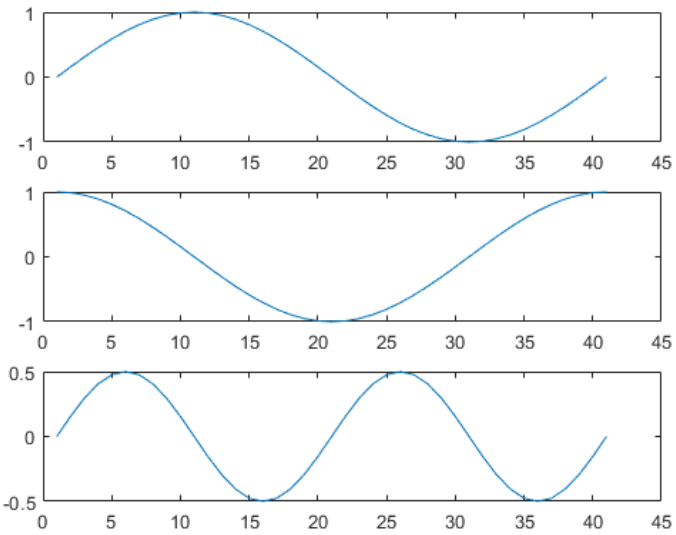
Displaying Multiple Plots in One Figure

The `subplot` command enables you to display multiple plots in the same window or print them on the same piece of paper. Typing

```
subplot(m,n,p)
```

partitions the figure window into an `m`-by-`n` matrix of small subplots and selects the p^{th} subplot for the current plot. The plots are numbered along the first row of the figure window, then the second row, and so on. For example, these statements plot data in three subregions of the figure window.

```
x = 0:pi/20:2*pi;
subplot(3,1,1); plot(sin(x))
subplot(3,1,2); plot(cos(x))
subplot(3,1,3); plot(sin(x).*cos(x))
```



Functions

surf	Surface plot
surfc	Contour plot under a 3-D shaded surface plot
surface	Create surface object

surfl	Surface plot with colormap-based lighting
surfnorm	Compute and display 3-D surface normals
mesh	Mesh plot
meshc	Plot a contour graph under mesh graph
meshz	Plot a curtain around mesh plot
hidden	Remove hidden lines from mesh plot
fsurf	Plot 3-D surface
fmesh	Plot 3-D mesh
fimplicit3	Plot 3-D implicit function
waterfall	Waterfall plot
ribbon	Ribbon plot
contour3	3-D contour plot
peaks	Example function of two variables
cylinder	Generate cylinder
ellipsoid	Generate ellipsoid
sphere	Generate sphere
pcolor	Pseudocolor (checkerboard) plot
surf2patch	Convert surface data to patch data

Properties

Surface Properties	Control chart surface appearance and behavior
Surface Properties	Control primitive surface appearance and behavior
FunctionSurface Properties	Control surface chart appearance and behavior
ImplicitFunctionSurface Properties	Control implicit surface chart appearance and behavior
ParameterizedFunctionSurface Properties	Control parameterized surface chart appearance and behavior

Part B (5x8=40 Marks)

Possible Questions

1. Describe in detail how you will create matrices with example.
2. Explain about Matrix Manipulation with example.
3. Write a short note on
   (i) Appending row or column
   (ii) Deleting a row or column
   (iii) Utility matrices
4. Describe about matrix and array operation with example.
5. List out logical operations with examples.
6. Describe about elementary math functions.
7. Explain in detail about character strings with example.
8. Explain the following with example
   (i) Manipulating Character String
   (ii) The eval function
9. Describe about command line functions
10. Describe about built in functions and online help.
11. Describe about the saving and loading from the binary Mat-Files.
12. Explain about plotting simple graph with example.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021

Subject: MATLAB programming **Subject Code: 15MMU504**

Class : III - B.Sc. Mathematics **Semester : V**

Unit II

Interactive Computation

Part A (20x1=20 Marks)
(Question Nos. 1 to 20 Online Examinations)

Possible Questions

Question	Choice 1	Choice 2	Choice 3	Choice 4	Answer
The output appearance of floating point numbers is controlled with the _____ command	format	long	deci	point	format
To suppresses the screen output by using _____ at the end of the command	dot	semicolon	camma	colon	semicolon
In Matlab command _____ does not need brackets	vector	string	scalar	matrix	scalar
If we give square brackets with no elements between them then it creates	identity matrix	square matrix	diagonal matrix	null matrix	null matrix
The three consecutive periods using in matrices are also called as _____	ellipse	hyperbola	parabola	circle	ellipse
_____ in MATLAB refers to the element a_{ij} of matrix A	Aij	A{ij}	A (i , j)	A (j, i)	A (i , j)
The dimensions of an existing matrix A may be obtained with the command _____	rand(A)	size(A)	eye(A)	length(A)	size(A)

which of the following command is used to delete the first and third rows of matrix A	A(: , [1 3]) = [ ]	A(1 3) = [ ]	A ([1 3] , :) = [ ]	A(1 ,3) = [ ]	A ([1 3] , :) = [ ]
The 0- 1 vector created by you is converted into a logical array with the command _____	array	logical	ones	zeros	logical
All the elements of matrix A can be strung into a single-column vector b by the command	b = A(:)	b = A(\)	b = A[ ]	b = A('')	b = A(:)
If matrix A is an m x n matrix, it can be reshaped into a p x q matrix with the command _____	reshape (A , :)	reshape (A , p , q)	reshape (p , q)	reshape (A ; p ; q)	reshape (A , p , q)
The transpose of matrix A is obtained by typing	A''	(A)	A'	A;	A'
A null matrix A is created by the command _____	A = { }	A = ( )	A = (0)	A = [ ]	A = [ ]
A (2 , :) = [ ] this command gives	deletes the second row of matrix, A	Create a 2x2 matrix, A	delete the second column of matix, A	create a two row matrix A	deletes the second row of matrix, A
To cretae a matrices with ones on the diagonal by using ____ command	rand(m,n)	eye(m,n)	diag(m,n)	zeros(m,n)	eye(m,n)
To extracts the first upper off-diagonal vector of matrix A with the ____ command	diag (A , 1)	eye(m,n)	zeros(m,n)	diag(m,n)	diag (A , 1)
The command eye(2) produce a 2x2 _____ matrix	zero	square	identity	null	identity
ones(3) , produce_____matrices of dimension 3.	square	null	zero	identity	square
To produce a = [0 0.5 1 1.5 2 2.5 3] by giving a command as ___	a = 0 … 3	a = 0 : 0.5 : 3	a = 0.5 : 3	a = 0 : 3	a = 0 : 0.5 : 3
_____ is the same as u=a : (b-a) / (n-1) : b	u = log(a,b,n)	u=line (a , b , n)	u=linspace (a , b , n)	u = logspace(a,b,n)	u=linspace (a , b , n)

_____ command is round the output towards 0	ceil	fix	floor	round	fix
To converts characters to their ASCII numeric values by using ____command	strcat	abs	char	ischar	abs
To converts any uppercase letters in the string to lowercase by using ___ command	lower	strcat	ischar	char	lower
_____ executes the string as a command	strcat	abs	eval	lower	eval
The syntax for creating an inline function is particularly simple	F = inline ('function')	F = inline ('function formula')	F = in('function formula')	F = inline ('formula')	F = inline ('function formula')
An _____is created by the command f = @(inputlist) mathematical expression	vector function	argument function	inline functions	anonymous function	anonymous function
To evaluates f (x) at x = 5 by giving	f(x)=5	fx(5)	f(5)	x(5)	fx(5)
____loop is conditionally execute statements	for	else	If	If else	If
_____ command is terminate scope of control statements.	If	end	for	else	end
To produce stunning surface plots in 3-D by using ____ command	ezsurf	ezpolar	ezplot	ezcontour	ezsurf

Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021.
Department of Mathematics

Subject : MATLAB programming		**Subject Code : 15MMU504**	**L**	**T**	**P**	**C**
Class : III – B.Sc. Mathematics		**Semester : V**	**5**	**0**	**0**	**5**

UNIT III

Programming in MATLAB: Scripts and Functions – Script files – Functions files-Language specific features – Advanced Data objects.

TEXT BOOK

1. RudraPratap, 2003. Getting Started with MATLAB-A Quick Introduction for Scientists and Engineers, Oxford University Press.

REFERENCES

1. William John Palm, 2005. Introduction to Matlab 7 for Engineers, McGraw-Hill Professional.New Delhi.

2. Dolores M. Etter, David C. Kuncicky, 2004.Introduction to MATLAB 7, Prentice Hall, New Delhi.

3. Kiranisingh.Y,Chaudhuri.B.B, 2007.Matlab Programming, Prentice-Hall Of India Pvt.Ltd, New Delhi.

4. Brian Hahn, 2016. Essential MATLAB for Engineers and Scientists.6thedition , Elsevier publication.

UNIT – III

Programming in MATLAB: Scripts and Functions

Script Files

Now suppose we want to modify the position vectors and find the distance between the new coordinates, it will not be worthwhile to type all the commands again. Instead one can create a *script* file, which will contain all the necessary information. To create a new script file, one can open their favorite text editor and type the following commands,

```
% dist_ab.m
% Calculates distance between any two position vectors a and b
d=b-a;
dd=d*d';
dist_ab=sqrt(dd)
```

Save these contents in a file named *dist_ab.m*and note that the script file has a '.m' extension which denotes that it is a MATLAB file. Now one can pick values for the vectors **a** and **b** by simply typing say,

```
>> a=[1 2 3];
>> b=[5 5 3];
```

Then find the distance by simply typing

```
>>dist_ab
dist_ab =
    5
```

This program can be called repeatedly to find the distance between any two position vectors that are entered by the user.

Function Files

It was tedious to have to assign the two vectors each time before using the above script file. One can combine the assignment of input values with the actual instruction, which invokes the script file by using a *function m-file*. Not only that, but also one can at the same time assign the answer to an output variable.

To create a new function file, one can open their favorite text editor and type the following commands,

```
% distfn.m
% Calculates the distance between two vectors a and b
% Input: a, b (position vectors)
% Output: dist_ab is the distance between a and b
function dist_ab = distfn(a , b)
  d= b – a;
dd = d*d';
  dist_ab = sqrt(dd);
```

Save these contents in a file named *distfn.m*and we should now be able to run,

```
>> dist_ab=distfn([1 2 3], [ 5 5 3])
```
or
```
>> a=[1 2 3];
>> b=[5 5 3];
>> dist_ab=distfn(a , b);
```
To save a certain part of the work in the MATLAB session, one can type,
```
>> diary work1
>> ….
>> ….
>> diary off
```
All the work between the diary commands will be saved into a file called *work1*.

Syntax

```
function [y1,...,yN] = myfun(x1,...,xM)
```

Description
example
function [y1,...,yN] = myfun(x1,...,xM) declares a function named myfun that accepts inputs x1,...,xM and returns outputs y1,...,yN. This declaration statement must be the first executable line of the function. Valid function names begin with an alphabetic character, and can contain letters, numbers, or underscores.

You can save your function:
In a function file which contains only function definitions. The name of the file should match the name of the first function in the file.
In a script file which contains commands and function definitions. Functions must be at the end of the file. Script files cannot have the same name as a function in the file. Functions are supported in scripts in R2016b or later.
Files can include multiple local functions or nested functions. For readability, use the end keyword to indicate the end of each function in a file. The end keyword is required when:
Any function in the file contains a nested function.
The function is a local function within a function file, and any local function in the file uses the end keyword.
The function is a local function within a script file.
Examples
collapse all
Function with One Output
Define a function in a file named average.m that accepts an input vector, calculates the average of the values, and returns a single result.

```
function y = average(x)
if ~isvector(x)
   error('Input must be a vector')
end
y = sum(x)/length(x);
```

```
end
```
Call the function from the command line.

```
z = 1:99;
average(z)
ans =
   50
```
Function in a Script File

Define a script in a file named integrationScript.m that computes the value of the integrand at and computes the area under the curve from 0 to . Include a local function that defines the integrand, .

```
% Compute the value of the integrand at 2*pi/3.
x = 2*pi/3;
y = myIntegrand(x)

% Compute the area under the curve from 0 to pi.
xmin = 0;
xmax = pi;
f = @myIntegrand;
a = integral(f,xmin,xmax)

function y = myIntegrand(x)
y = sin(x).^3;
end
y =

   0.6495


a =

   1.3333
```

Function with Multiple Outputs

Define a function in a file named stat.m that returns the mean and standard deviation of an input vector.

```
function [m,s] = stat(x)
n = length(x);
m = sum(x)/n;
s = sqrt(sum((x-m).^2/n));
end
```
Call the function from the command line.

```
values = [12.7, 45.4, 98.9, 26.6, 53.1];
[ave,stdev] = stat(values)
ave =
   47.3400
stdev =
   29.4124
```

Multiple Functions in a Function File

Define two functions in a file named stat2.m, where the first function calls the second.

```
function [m,s] = stat2(x)
n = length(x);
m = avg(x,n);
s = sqrt(sum((x-m).^2/n));
end

function m = avg(x,n)
m = sum(x)/n;
end
```

Function avg is a local function. Local functions are only available to other functions within the same file.

Call function stat2 from the command line.

```
values = [12.7, 45.4, 98.9, 26.6, 53.1];
[ave,stdev] = stat2(values)
ave =
   47.3400
stdev =
   29.4124
```

What Are Nested Functions?

A nested function is a function that is completely contained within a parent function. Any function in a program file can include a nested function.

For example, this function named `parent` contains a nested function named `nestedfx`:

```
function parent
disp('This is the parent function')
nestedfx

   function nestedfx
disp('This is the nested function')
   end

end
```

The primary difference between nested functions and other types of functions is that they can access and modify variables that are defined in their parent functions. As a result:

- Nested functions can use variables that are not explicitly passed as input arguments.
- In a parent function, you can create a handle to a nested function that contains the data necessary to run the nested function.

Requirements for Nested Functions

- Typically, functions do not require an `end` statement. However, to nest any function in a program file, *all* functions in that file must use an `end` statement.
- You cannot define a nested function inside any of the MATLAB® program control statements, such as `if/elseif/else`, `switch/case`, `for`, `while`, or `try/catch`.
- You must call a nested function either directly by name (without using `feval`), or using a function handle that you created using the `@` operator (and not `str2func`).
- All of the variables in nested functions or the functions that contain them must be explicitly defined. That is, you cannot call a function or script that assigns values to variables unless those variables already exist in the function workspace. (For more information, see Variables in Nested and Anonymous Functions.)

Sharing Variables Between Parent and Nested Functions

In general, variables in one function workspace are not available to other functions. However, nested functions can access and modify variables in the workspaces of the functions that contain them.

This means that both a nested function and a function that contains it can modify the same variable without passing that variable as an argument. For example, in each of these functions, `main1` and `main2`, both the main function and the nested function can access variable `x`:

```
function main1
x = 5;
nestfun1

   function nestfun1
     x = x + 1;
   end

end
```

```
function main2
nestfun2

   function nestfun2
      x = 5;
   end

x = x + 1;
end
```

When parent functions do not use a given variable, the variable remains local to the nested function. For example, in this function named `main`, the two nested functions have their own versions of `x` that cannot interact with each other:

```
function main
   nestedfun1
   nestedfun2

   function nestedfun1
      x = 1;
   end

   function nestedfun2
      x = 2;
   end
end
```

Functions that return output arguments have variables for the outputs in their workspace. However, parent functions only have variables for the output of nested functions if they explicitly request them. For example, this function `parentfun` does *not* have variable `y` in its workspace:

```
function parentfun
x = 5;
nestfun;

   function y = nestfun
     y = x + 1;
   end

end
```

If you modify the code as follows, variable `z` is in the workspace of `parentfun`:

```
function parentfun
x = 5;
z = nestfun;

   function y = nestfun
     y = x + 1;
   end

end
```

Using Handles to Store Function Parameters

Nested functions can use variables from three sources:

- Input arguments
- Variables defined within the nested function
- Variables defined in a parent function, also called *externally scoped* variables

When you create a function handle for a nested function, that handle stores not only the name of the function, but also the values of externally scoped variables.

For example, create a function in a file named `makeParabola.m`. This function accepts several polynomial coefficients, and returns a handle to a nested function that calculates the value of that polynomial.

```
function p = makeParabola(a,b,c)
p = @parabola;

   function y = parabola(x)
   y = a*x.^2 + b*x + c;
   end

end
```

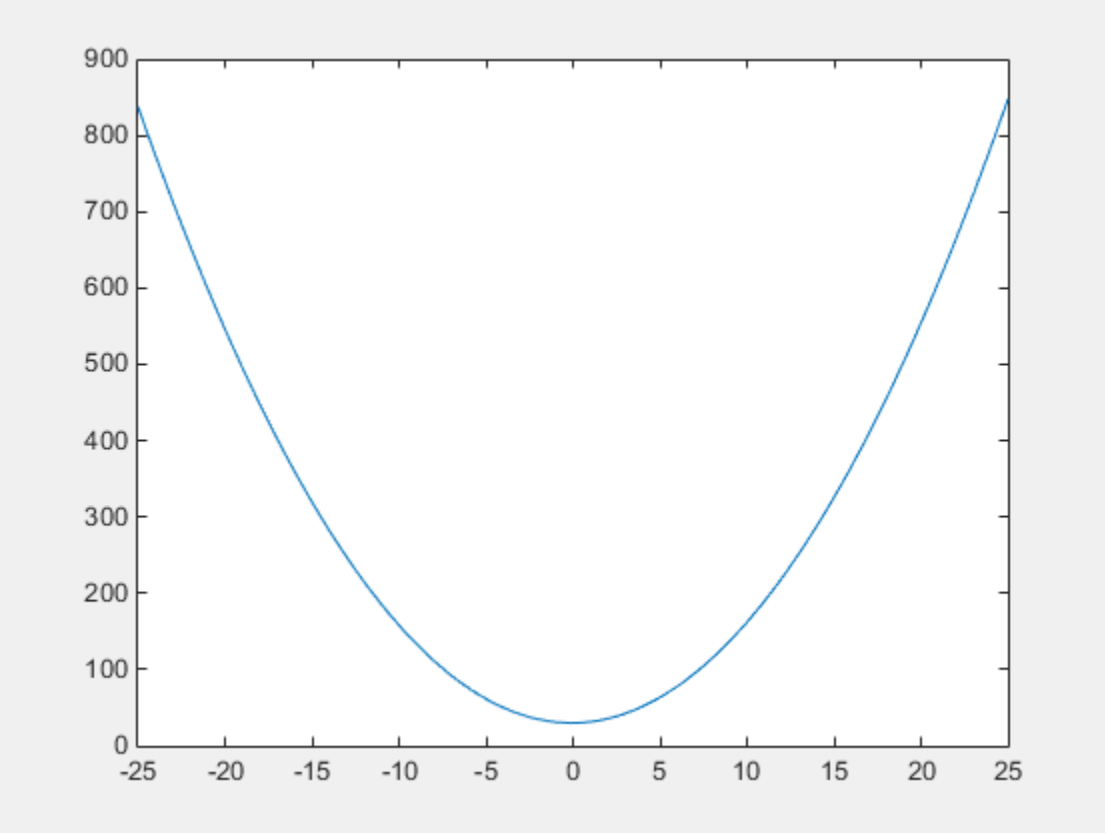
The `makeParabola` function returns a handle to the `parabola` function that includes values for coefficients `a`, `b`, and `c`.

At the command line, call the `makeParabola` function with coefficient values of `1.3`, `.2`, and `30`. Use the returned function handle `p` to evaluate the polynomial at a particular point:

```
p = makeParabola(1.3,.2,30);

X = 25;
Y = p(X)

Y =
  847.5000
```

Many MATLAB functions accept function handle inputs to evaluate functions over a range of values. For example, plot the parabolic equation from `-25` to `+25`:

```
fplot(p,[-25,25])
```



You can create multiple handles to the `parabola` function that each use different polynomial coefficients:

```
firstp = makeParabola(0.8,1.6,32);
secondp = makeParabola(3,4,50);
range = [-25,25];

figure
hold on
fplot(firstp,range)
fplot(secondp,range,'r:')
hold off
```

PROFILE

Profile execution time for functions

Syntax

profile action

profile action option1 ... optionN

profile option1 ... optionN

p = profile('info')

s = profile('status')

Description

example

profile action profiles the execution time for functions. Use action to start, stop, and restart the Profiler, and view or clear profile statistics. For example, profileon starts the Profiler.

example

profile action option1 ... optionN starts or restarts the Profiler with the specified options. For example, profile resume -history restarts the Profiler and records the sequence of function calls.

example

profile option1 ... optionN sets the specified Profiler options. If the Profiler is on and you specify one of the options, MATLAB throws an error. To change options, first specify profile off, and then specify the new options.

example

p = profile('info') stops the Profiler and displays a structure containing the results. To access the data generated by profile, use this syntax.

example

s = profile('status') returns a structure with the Profiler status information.

Examples

Create the file myFunction.m using this main function and local function.

```
function c = myFunction(a,b)

c = sqrt(square(a)+square(b));

end


function y = square(x)

y = x.^2;
```

```
end
```

Turn on the Profiler, and enable the function call history option. Profile a call to the myFunction function.

```
profile on -history
a = rand(5);
b = rand(5);
c = myFunction(a,b);
```

Save the profile results.

```
p = profile('info')
p =

FunctionTable: [2x1 struct]
FunctionHistory: [2x6 double]
ClockPrecision: 3.3475e-07
ClockSpeed: 3.0600e+09
          Name: 'MATLAB'
      Overhead: 0
```

Display the function call history.

```
p.FunctionHistory
ans =

    0    0    1    0    1    1
    1    2    2    2    2    1
```

Display function entry and exit information by iterating over the function call history.

```
numEvents = size(p.FunctionHistory,2);
for n = 1:numEvents
   name = p.FunctionTable(p.FunctionHistory(2,n)).FunctionName;

   if p.FunctionHistory(1,n) == 0
disp(['Entered ' name]);
```

```
   else
disp(['Exited ' name]);
   end
end
```

```
Entered myFunction
Entered myFunction>square
Exited myFunction>square
Entered myFunction>square
Exited myFunction>square
Exited myFunction
```

Set the function call history to the default value.

```
profile -nohistory
```

GLOBAL

Declare variables as global

Syntax

```
global var1 ... varN
```

Description

global var1 ... varN declares variables var1 ... varN as global in scope.

Ordinarily, each MATLAB® function has its own local variables, which are separate from those of other functions and from those of the base workspace. However, if several functions all declare a particular variable name as global, then they all share a single copy of that variable. Any change of value to that variable, in any function, is visible to all the functions that declare it as global.

If the global variable does not exist the first time you issue the global statement, it is initialized to an empty 0x0 matrix.

If a variable with the same name as the global variable already exists in the current workspace, MATLAB issues a warning and changes the value of that variable and its scope to match the global variable.

Examples

Global Variable Between Functions

Create a function in your current working folder that sets the value of a global variable.

```
function setGlobalx(val)
global x
```

```
x = val;
```

Create a function in your current working folder that returns the value of a global variable. These two functions have separate function workspaces, but they both can access the global variable.

```
function r = getGlobalx
global x
r = x;
```

Set the value of the global variable, x, and obtain it from a different workspace.

```
setGlobalx(1138)
r = getGlobalx
r =

     1138
```

Share Global Variable Between Function and Command Line

Assign a value to the global variable using the function that you defined in the previous example.

```
clear all
setGlobalx(42)
```

Display the value of the global variable, x. Even though the variable is global, it is not accessible at the command line.

```
x
Undefined function or variable 'x'.
```

Declare x as a global variable at the command line, and display its value.

```
global x
x
x =
   42
```

Change the value of x and use the function that you defined in the previous example to return the global value from a different workspace.

```
x = 1701;
r = getGlobalx
r =
```

1701

Conditional Statements:

`for` Loops:

This allows a group of commands to be repeated a fixed, predetermined number of times. The general form of a `for` loop is:

```
for x = array
     Commands …
End
```

For example,

```
>> for n=1:10
        x(n)=sin(n*pi/10);
   end
```

yields the vector x given by,

```
>> x
x =
Columns 1 through 7
0.3090   0.5878   0.8090   0.9511   1.0000  0.9511  0.8090
Columns 8 through 10
0.5878   0.3090   0.0000
```

Let us now use the for loop to generate the first 15 Fibonacci numbers 1,1,2,3,5,8,…

```
>> f=[1 1];
>> for k=1:15
     f(k+2) = f(k+1) + f(k);
   end
>> f
```

`while` Loops

This evaluates a group of commands an infinite number of times unlike the `for` loop that evaluates a group of commands a fixed number of times. The general form for while loop is

```
while expression
```

```
        Commands …
      end
```

For example to generate all the Fibonacci numbers less than 1000, we can do the following:

```
>> f=[1 1];
>> k=1;
>> while f(k) < 1000
     f(k+1) = f(k+1) + f(k);
     k = k +1;
    end
>> f
```

if-else-end Construct

There are times when a sequence of commands must be conditionally evaluated based on a relational test. This is done by the if-else-end construct whose general form is,

```
      if expression
         Commands …
      end
```

For example if we want to give 20% discount for larger purchases of oranges, we say,

```
>> oranges=10;              % number of oranges
>> cost = oranges*25        % Cost of oranges
cost =
     250
>> if oranges > 5
        cost = (1-20/100)*cost;
   end
>> cost
cost =
   200
```

If there are more conditions to be evaluated then one uses the more general if-else-end construct given by,

```
   if expression
```

```
    Commands evaluated if True
  else
    Commands evaluated if False
  end
```

While Loops

If you don't like the *for loop*, you can also use a *while loop*. The *while loop* repeats a sequence of commands as long as some condition is met. This can make for a more efficient algorithm. In the previous example the number of time steps to make may be much larger than 20. In such a case the *for loop* can use up a lot of memory just creating the vector used for the index. A better way of implementing the algorithm is to repeat the same operations but only as long as the number of steps taken is below some threshold. In this example the D.E. y'=x-|y|, y(0)=1, is approximated using Euler's Method:

```
>> h = 0.001;
>> x = [0:h:2];
>> y = 0*x;
>> y(1) = 1;
>> i = 1;
>> size(x)

ans =

           1        2001

>> max(size(x))

ans =

        2001

>> while(i<max(size(x)))
     y(i+1) = y(i) + h*(x(i)-abs(y(i)));
     i = i + 1;
   end
>> plot(x,y,'go')
>> plot(x,y)
```

BREAK

Syntax

break

Description

break terminates the execution of a for or while loop. Statements in the loop after the break statement do not execute.

In nested loops, break exits only from the loop in which it occurs. Control passes to the statement that follows the end of that loop.

Examples

Sum a sequence of random numbers until the next random number is greater than an upper limit. Then, exit the loop using a breakstatement.

```
limit = 0.8;
s = 0;

while 1
tmp = rand;
   if tmp> limit
      break
   end
   s = s + tmp;
end
```

CONTINUE

Pass control to next iteration of for or while loop

Syntax

continue

Description

Continue passes control to the next iteration of a for or while loop. It skips any remaining statements in the body of the loop for the current iteration. The program continues execution from the next iteration.

continue applies only to the body of the loop where it is called. In nested loops, continue skips remaining statements only in the body of the loop in which it occurs.

Examples

Selectively Display Values in Loop

Try this Example

Display the multiples of 7 from 1 through 50. If a number is not divisible by 7, use continue to skip the disp statement and pass control to the next iteration of the for loop.

```
for n = 1:50
   if mod(n,7)
      continue
   end
disp(['Divisible by 7: ' num2str(n)])
end
```

Divisible by 7: 7

Divisible by 7: 14

Divisible by 7: 21

Divisible by 7: 28

Divisible by 7: 35

Divisible by 7: 42

Divisible by 7: 49

switch-case Construct

This is used when sequences of commands must be conditionally evaluated based on repeated use of an equality test with one common argument. In general the form is,

```
switch expression
     case test_expression1
           Commands_1 …
     case test_expression2
           Commands_2 …
      …..
     otherwise
Commands_n ….
end
```

Let us now consider the problem of converting entry in given units to centimeters.

```
% centimeter.m
% This program converts a given measument into the equivalent in cms
% It illustrates the use of SWITCH-CASE control flow
function y=centimeter(A,units)
switch units   % convert A to cms
    case {'inch','in'}
      y = A*2.54;
    case {'feet','ft'}
      y = A*2.54*12;
    case {'meter','m'}
      y = A*100;
    case {'centimeter','cm'}
      y = A;
    otherwise
disp(['Unknown units: ' units])
       y = nan;   %% stands for not a number
end
```

RETURN

Syntax

return

Examples

In your current working folder, create a function, findSqrRootIndex, to find the index of the first occurrence of the square root of a value within an array. If the square root isn't found, the function returns NaN.

```
function idx = findSqrRootIndex(target,arrayToSearch)


idx = NaN;
if target < 0
  return
```

```
end


for idx = 1:length(arrayToSearch)
   if arrayToSearch(idx) == sqrt(target)
      return
   end
end
```

INTERACTIVE INPUT

Syntax

input(<prompt1>)

input(<prompt1>, x1, <prompt2>, x2, …)

Description

input allows interactive input of MuPAD® objects.

input() displays the prompt "Please enter expression:" and waits for input by the user. The input, terminated by pressing the Return key, is parsed and returned *unevaluatedly*.

input(prompt1) uses the character string prompt1 instead of the default prompt "Please enter expression:".

input(prompt1 x1) assigns the input to the identifier or local variable x1. The default prompt is used, if no prompt string is specified.

Several objects can be read with a single input command. Each identifier or variable in the sequence of arguments makes input return a prompt, waiting for input to be assigned to it. A character string preceding an identifier or variable in the argument sequence replaces the default prompt. Arguments that are neither prompt strings nor identifiers or variables are ignored.

The identifiers or variables x1 etc. may have values. These are overwritten by input.

input only parses the input objects for syntactical correctness. It does not evaluate them.

Example

The default prompt is displayed. The input is returned without evaluation:

input()

Please enter expression: << 1 + 2 >>

1 + 2

A character string is used as a prompt:

input("enter a number: ")

enter a number: << 5

>>

5

The input may be assigned to an identifier:

input(x)

Please enter expression: << 5 >>

5

x

5

A user-defined prompt is used, the input is assigned to an identifier:

input("enter a number: ", x)

enter a number: << 6

>>

6

x

6

delete x:

ADVANCED DATA OBJECTS

STRUCTURE

A *structure array* is a data type that groups related data using data containers called *fields*. Each field can contain any type of data. Access data in a field using dot notation of the form structName.fieldName.

Creation

When you have data to put into a new structure, create the structure using dot notation to name its fields one at a time:

s.a = 1;

s.b = {'A','B','C'}

s =

struct with fields:

a: 1

b: {'A' 'B' 'C'}

You also can create a structure array using the struct function, described below. You can specify many fields simultaneously, or create a nonscalar structure array.

Syntax

s = struct

s = struct(field,value)

s = struct(field1,value1,...,fieldN,valueN)

s = struct([])

s = struct(obj)

Description

s = struct creates a scalar (1-by-1) structure with no fields.

s = struct(field,value) creates a structure array with the specified field and values.
The value input argument can be any data type, such as a numeric, logical, character, or cell array.

- If value is *not* a cell array, or if value is a scalar cell array, then s is a scalar structure. For instance, s = struct('a',[1 2 3]) creates a 1-by-1 structure, where s.a = [1 2 3].
- If value is a nonscalar cell array, then s is a structure array with the same dimensions as value. Each element of s contains the corresponding element ofvalue. For example, s = struct('x',{'a','b'},'y','c') returns s(1).x = 'a', s(2).x = 'b', s(1).y = 'c', and s(2).y = 'c'.
- If value is an empty cell array {}, then s is an empty (0-by-0) structure.

s = struct(field1,value1,...,fieldN,valueN) creates multiple fields. Any nonscalar cell arrays in the set value1,...,valueN must have the same dimensions.

- If none of the value inputs are cell arrays, or if all value inputs that are cell arrays are scalars, then s is a scalar structure.
- If any of the value inputs is a nonscalar cell array, then s has the same dimensions as the nonscalar cell array. For any value that is a scalar cell array or an array of any other data type, struct inserts the contents of value in the relevant field for all elements of s.
- If any value input is an empty cell array, {}, then output s is an empty (0-by-0) structure. To specify an empty field and keep the values of the other fields, use [] as a value input instead.

s = struct([]) creates an empty (0-by-0) structure with no fields.

s = struct(obj) creates a scalar structure with field names and values that correspond to properties of obj. The struct function does not convert obj, but rather creates s as a new structure. This structure does not retain the class information, so private, protected, and hidden properties become public fields in s. The struct function issues a warning when you use this syntax.

Examples

Create a nonscalar structure that contains a single field.

```
field = 'f';
value = {'some text';
         [10, 20, 30];
         magic(5)};
s = struct(field,value)
```

s = *3x1 struct array with fields:*

f

View the contents of each element.

```
s.f
ans =
'some text'
ans =

    10    20    30


ans =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Create a nonscalar structure that contains several fields.

```
field1 = 'f1';  value1 = zeros(1,10);
field2 = 'f2';  value2 = {'a', 'b'};
field3 = 'f3';  value3 = {pi, pi.^2};
field4 = 'f4';  value4 = {'fourth'};

s = struct(field1,value1,field2,value2,field3,value3,field4,value4)
```

s = *1x2 struct array with fields:*

f1

f2

f3

f4

The cell arrays for value2 and value3 are 1-by-2, so s is also 1-by-2. Because value1 is a numeric array and not a cell array, both s(1).f1 and s(2).f1 have the same contents. Similarly, because the cell array for value4 has a single element, s(1).f4 and s(2).f4 have the same contents.

```
s(1)
```

ans = *struct with fields:*

f1: [0 0 0 0 0 0 0 0 0 0]

f2: 'a'

f3: 3.1416

f4: 'fourth'

```
s(2)
```

ans = *struct with fields:*

f1: [0 0 0 0 0 0 0 0 0 0]

f2: 'b'

f3: 9.8696

f4: 'fourth'

Create a structure with a field that contains a cell array.

```
field = 'mycell';
value = {{'a','b','c'}};
s = struct(field,value)
```

s = *struct with fields:*

mycell: {'a' 'b' 'c'}

CELL

Description

A *cell array* is a data type with indexed data containers called *cells*, where each cell can contain any type of data. Cell arrays commonly contain either lists of text, combinations of text and numbers, or numeric arrays of different sizes. Refer to sets of cells by enclosing indices in smooth parentheses, (). Access the contents of cells by indexing with curly braces, {}.

Creation

When you have data to put into a cell array, create the array using the cell array construction operator, {}.

```
C = {1,2,3;
    'text',rand(5,10,2),{11; 22; 33}}
```

C =

2x3 cell array

{[1]}	{[2]}	{[3]}
{'text'}	{5x10x2 double}	{3x1 cell}

You also can use {} to create an empty 0-by-0 cell array.

```
C = {}
```

C =

0x0 empty cell array

To create a cell array with a specified size, use the cell function, described below.

You can use cell to preallocate a cell array to which you assign data later. cell also converts certain types of Java, .NET, and Python data structures to cell arrays of equivalent MATLAB objects.

Syntax

C = cell(n)

C = cell(sz1,...,szN)

C = cell(sz)

D = cell(obj)

Description

C = cell(n) returns an n-by-n cell array of empty matrices.

C = cell(sz1,...,szN) returns a sz1-by-...-by-szN cell array of empty matrices where sz1,...,szN indicate the size of each dimension. For example, cell(2,3) returns a 2-by-3 cell array.

C = cell(sz) returns a cell array of empty matrices where size vector sz defines size(C). For example, cell([2 3]) returns a 2-by-3 cell array.

D = cell(obj) converts a Java array, .NET System.String or System.Object array, or Python sequence into a MATLAB cell array.

Examples

Create a 3-by-3 cell array of empty matrices.

C = cell(3)

C = *3x3 cell array*

{0x0 double}	{0x0 double}	{0x0 double}
{0x0 double}	{0x0 double}	{0x0 double}
{0x0 double}	{0x0 double}	{0x0 double}

Create a 3-by-4-by-2 cell array of empty matrices.

```
C = cell(3,4,2);
size(C)
ans =

    3    4    2
```

Functions

struct	Structure array
fieldnames	Field names of structure, or public fields of COM or Java object
getfield	Field of structure array
isfield	Determine whether input is structure array field
isstruct	Determine whether input is structure array
orderfields	Order fields of structure array
rmfield	Remove fields from structure
setfield	Assign values to structure array field
arrayfun	Apply function to each element of array
structfun	Apply function to each field of scalar structure
table2struct	Convert table to structure array
struct2table	Convert structure array to table
cell2struct	Convert cell array to structure array
struct2cell	Convert structure to cell array

Part B (5x8=40 Marks)

Possible Questions

1. Describe about script files with examples.
2. Give a brief note on executing a function with examples.
3. Explain about function files with examples.
4. Describe about executing a function inside another function and a function in input list.
5. What is a sub function? Describe about compiled functions and the profiler.
6. Explain the control flow statements with examples.
7. Define global variables.Give example for solving 1^{st} order ODE by using global variables.
8. List out the commands for interactive user input in script file or function file.
9. Explain about the cells with examples.
10. Give a brief notes on structures with an example.
11. Expalin about the
    (i)Switch case
    (ii) Break with examples.
12. Give a brief notes on menu and pause commands in Matlab.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021

Subject: MATLAB programming **Subject Code: 15MMU504**

Class : III - B.Sc. Mathematics **Semester : V**

Unit III

Programming in MATLAB: Scripts and Functions

Part A (20x1=20 Marks)

Possible Questions

Question	Choice 1	Choice 2	Choice 3	Choice 4	Answer
_____is executed by typing name of the file on the commamd line.	Script file	function file	date file	figure file	Script file
A script file is an _____ with a set of vaild MATLAB command	Mex - file	Mat - file	M -file	fig - file	M -file
A ____ is most versatile data object in Matlab	matrix	array	string	cell	cell
Which is these is not an aspect of a for or while loop	update	initialization	runner	condition	runner
To better manage memeory and prevent unnecessary memory allocations, Matlab use,_____	vectors	scalars	matrix math	delayed copy	delayed copy
To proint a new line in a fprintf statement, you must use the following escape character	\t	\n	\nxt	\n1	\n
In Matlab this keyword immediately moves to the next itreation of the loop	continue	update	goto	break	continue

Which of these is the way of access the first element in a vector named v (assuming there is at least one element in the vector) ?	v(0)	v(1)	v	v(: , 0)	v(1)
If I want to save a formatted string to memory, but don't want to print it out, which command should I use ?	fprintf	sprintf	disp	echo	sprintf
When used in the fprintf command ,the %g is used as the	single character display	fixed point display	string natation display	default number display	default number display
When used in the fprintf command ,the \n is used to	add a space between any two character	add a line space	place a number into comment	clear the comment	add a line space
To display 'Question 1' ' in the command window, the correct command is	disp(Question 1)	display('Question 1 ')	disp('Question 1')	Question 1	disp('Question 1')
The num2str command is used to _____	convert a number to string	convert a string to number	concatenates number and string	concatenates string	convert a number to string
To join one or more strings into a single string is known as _____	string convertion	joining	concatenation	string theory	concatenation
In ____ loop the number of repetations is already known	if	while	if else	for	for
In ____ loop execution is repeated until the condition is satisfied	if else	for	while	if	while
In ___ control structure a group of statements are executed only if the condition is true.	if	if else	if elseif	nested if	if
In ___ control structure two group of statements are executed only if one is true and other is false condition.	if	if else	if elseif	nested if	if else
Which is provide a user the option to check a large number of different conditions.	if	if else	if elseif	nested if	if elseif
When one of the if structure lies entirely within the domain of the other if strcture then it is called___	if	if else	if elseif	nested if	nested if

____ command terminate the execution of for or while loop	switch	break	continue	error	break
_____ command display the message and abort function	switch	break	continue	error	error
____ command catch the error generated by MATLAB	try-catch	break	continue	error	try-catch
The first function in the file is called _____ function	Inline	private	primary	nested	primary
Additional functions defined within the same files are called ____	primary function	subfunction	Inline function	nested function	subfunction
_____ functioms are visble only to the functions in their parent folder.	Inline	private	primary	nested	private
_____ function provide a way to pass information without using global variables.	Inline	private	primary	nested	nested
_____ error are caused by grammatical mistake in the statement include in the program	syntax	condition	runtime	statement	syntax
____ error are caused mainly due to wrong logic used by the program.	syntax	condition	runtime	statement	runtime

Enable | Enlighten | Enrich
(Deemed to be University)
(Under Section 3 of UGC Act 1956)

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University Established Under Section 3 of UGC Act 1956)

Pollachi Main Road, Eachanari (Po),

Coimbatore –641 021.

Department of Mathematics

Subject : MATLAB programming	**Subject Code : 15MMU504**	**L T P C**
Class : III – B.Sc. Mathematics	**Semester : V**	**5 0 0 5**

UNIT IV

Applications – Linear Algebra - Solving a linear system – Finding Eigen values and Eigen vectors – Matrix Factorizations.

TEXT BOOK

1. RudraPratap, 2003. Getting Started with MATLAB-A Quick Introduction for Scientists and Engineers, Oxford University Press.

REFERENCES

1. William John Palm, 2005. Introduction to Matlab 7 for Engineers, McGraw-Hill Professional.New Delhi.

2. Dolores M. Etter, David C. Kuncicky, 2004.Introduction to MATLAB 7, Prentice Hall, New Delhi.

3. Kiranisingh.Y,Chaudhuri.B.B, 2007.Matlab Programming, Prentice-Hall Of India Pvt.Ltd, New Delhi.

4. Brian Hahn, 2016. Essential MATLAB for Engineers and Scientists.6thedition , Elsevier publication.

UNIT – IV

APPLICATIONS OF MATLAB

Linear Algebra

Linear algebra functions in MATLAB provide fast, numerically robust matrix calculations. Capabilities include a variety of matrix factorizations, linear equation solving, computation of eigenvalues or singular values, and more

Functions

Linear Equations

mldivide	Solve systems of linear equations Ax = B for x
mrdivide	Solve systems of linear equations xA = B for x
decomposition	Matrix decomposition for solving linear systems
lsqminnorm	Minimum norm least-squares solution to linear equation
linsolve	Solve linear system of equations
inv	Matrix inverse
pinv	Moore-Penrose pseudoinverse
lscov	Least-squares solution in presence of known covariance
lsqnonneg	Solve nonnegative linear least-squares problem
sylvester	Solve Sylvester equation AX + XB = C for X

Eigenvalues and Singular Values

eig	Eigenvalues and eigenvectors
eigs	Subset of eigenvalues and eigenvectors
balance	Diagonal scaling to improve eigenvalue accuracy
svd	Singular value decomposition
svds	Subset of singular values and vectors
gsvd	Generalized singular value decomposition
ordeig	Eigenvalues of quasitriangular matrices
ordqz	Reorder eigenvalues in QZ factorization
ordschur	Reorder eigenvalues in Schur factorization
polyeig	Polynomial eigenvalue problem
qz	QZ factorization for generalized eigenvalues
hess	Hessenberg form of matrix
schur	Schur decomposition
rsf2csf	Convert real Schur form to complex Schur form

cdf2rdf	Convert complex diagonal form to real block diagonal form

Matrix Decomposition

lu	LU matrix factorization
ldl	Block LDL' factorization for Hermitian indefinite matrices
chol	Cholesky factorization
cholupdate	Rank 1 update to Cholesky factorization
qr	Orthogonal-triangular decomposition
qrdelete	Remove column or row from QR factorization
qrinsert	Insert column or row into QR factorization
qrupdate	Rank 1 update to QR factorization
planerot	Givens plane rotation

Matrix Operations

transpose	Transpose vector or matrix
ctranspose	Complex conjugate transpose
mtimes	Matrix Multiplication
mpower	Matrix power
sqrtm	Matrix square root
expm	Matrix exponential
logm	Matrix logarithm
funm	Evaluate general matrix function
kron	Kronecker tensor product
cross	Cross product
dot	Dot product

EXAMPLES
mldivide, \
Solve systems of linear equations $Ax = B$ for x

Syntax
x = A\B
x = mldivide(A,B)
Description

x = A\B solves the system of linear equations A*x = B. The matrices A and B must have the same number of rows. MATLAB® displays a warning message if A is badly scaled or nearly singular, but performs the calculation regardless.

- If A is a scalar, then A\B is equivalent to A.\B.

- If A is a square n-by-n matrix and B is a matrix with n rows, then x = A\B is a solution to the equation A*x = B, if it exists.
- If A is a rectangular m-by-n matrix with m ~= n, and B is a matrix with m rows, then A\B returns a least-squares solution to the system of equations A*x= B.

x = mldivide(A,B) is an alternative way to execute x = A\B, but is rarely used. It enables operator overloading for classes.

Examples

System of Equations

Try this Example

Solve a simple system of linear equations, A*x = B.

```
A = magic(3);
B = [15; 15; 15];
x = A\B
x =

   1.0000
   1.0000
   1.0000
```

Linear System with Singular Matrix

Solve a linear system of equations A*x = b involving a singular matrix, A.

```
A = magic(4);
b = [34; 34; 34; 34];
x = A\b
x =

   1.5000
   2.5000
  -0.5000
   0.5000
```

When rcond is between 0 and eps, MATLAB issues a nearly singular warning, but proceeds with the calculation. When working with ill-conditioned matrices, an unreliable solution can result even though the residual (b-A*x) is relatively small. In this particular example, the norm of the residual is zero, and an exact solution is obtained, although rcond is small.

When rcond is equal to 0, the singular warning appears.

```
A = [1 0; 0 0];
b = [1; 1];
x = A\b
```

Finding Eigenvalues and eigenvectors

Syntax
e = eig(A)
[V,D] = eig(A)
[V,D,W] = eig(A)
e = eig(A,B)
[V,D] = eig(A,B)
[V,D,W] = eig(A,B)
[___] = eig(A,balanceOption)
[___] = eig(A,B,algorithm)
[___] = eig(___,eigvalOption)
Description

e = eig(A) returns a column vector containing the eigenvalues of square matrix A.

[V,D] = eig(A) returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that A*V = V*D.

[V,D,W] = eig(A) also returns full matrix W whose columns are the corresponding left eigenvectors, so that W'*A = D*W'.
The eigenvalue problem is to determine the solution to the equation $Av = \lambda v$, where A is an n-by-n matrix, v is a column vector of length n, and λ is a scalar. The values of λ that satisfy the equation are the eigenvalues. The corresponding values of v that satisfy the equation are the right eigenvectors. The left eigenvectors, w, satisfy the equation $w'A = \lambda w'$.

e = eig(A,B) returns a column vector containing the generalized eigenvalues of square matrices A and B.

[V,D] = eig(A,B) returns diagonal matrix D of generalized eigenvalues and full matrix V whose columns are the corresponding right eigenvectors, so that A*V = B*V*D.
[V,D,W] = eig(A,B) also returns full matrix W whose columns are the corresponding left eigenvectors, so that W'*A = D*W'*B.
The generalized eigenvalue problem is to determine the solution to the equation $Av = \lambda Bv$, where A and B are n-by-n matrices, v is a column vector of length n, and λ is a scalar. The values of λ that satisfy the equation are the generalized eigenvalues. The corresponding values of v are the generalized right eigenvectors. The left eigenvectors, w, satisfy the equation $w'A = \lambda w'B$.
[___] = eig(A,balanceOption), where balanceOption is 'nobalance', disables the preliminary balancing step in the algorithm. The default for balanceOption is 'balance', which enables balancing. The eig function can return any of the output arguments in previous syntaxes.

[___] = eig(A,B,algorithm), where algorithm is 'chol', uses the Cholesky factorization of B to compute the generalized eigenvalues. The default for algorithmdepends on the properties of A and B, but is generally 'qz', which uses the QZ algorithm.
If A is Hermitian and B is Hermitian positive definite, then the default for algorithm is 'chol'.

[___] = eig(___,eigvalOption) returns the eigenvalues in the form specified by eigvalOption using any of the input or output arguments in previous syntaxes. Specify eigvalOption as 'vector' to return the eigenvalues in a column vector or as 'matrix' to return the eigenvalues in a diagonal matrix.

Examples

Eigenvalues of Matrix

Try this Example

Use gallery to create a symmetric positive definite matrix.

```
A = gallery('lehmer',4)
A =

    1.0000    0.5000    0.3333    0.2500
    0.5000    1.0000    0.6667    0.5000
    0.3333    0.6667    1.0000    0.7500
    0.2500    0.5000    0.7500    1.0000
```

Calculate the eigenvalues of A. The result is a column vector.

```
e = eig(A)
e =

    0.2078
    0.4078
    0.8482
    2.5362
```

MATRIX FACTORIZATION

svd

Singular value decomposition

Syntax

```
s = svd(A)
[U,S,V] = svd(A)
[U,S,V] = svd(A,'econ')
[U,S,V] = svd(A,0)
```

Description

s = svd(A) returns the singular values of matrix A in descending order.

[U,S,V] = svd(A) performs a singular value decomposition of matrix A, such that A = U*S*V'.

[U,S,V] = svd(A,'econ') produces an economy-size decomposition of m-by-n matrix A:

- m > n — Only the first n columns of U are computed, and S is n-by-n.
- m = n — svd(A,'econ') is equivalent to svd(A).
- m < n — Only the first m columns of V are computed, and S is m-by-m.

The economy-size decomposition removes extra rows or columns of zeros from the diagonal matrix of singular values, S, along with the columns in either U or V that multiply those zeros in

the expression A = U*S*V'. Removing these zeros and columns can improve execution time and reduce storage requirements without compromising the accuracy of the decomposition.

[U,S,V] = svd(A,0) produces a different economy-size decomposition of m-by-n matrix A:

- m > n — svd(A,0) is equivalent to svd(A,'econ').
- m <= n — svd(A,0) is equivalent to svd(A).

Examples

Singular Values of Matrix

Try this Example

Compute the singular values of a full rank matrix.

```
A = [1 0 1; -1 -2 0; 0 1 -1]
A =

     1     0     1
    -1    -2     0
     0     1    -1

s = svd(A)
s =

    2.4605
    1.6996
    0.2391
```

LU matrix factorization

Syntax

```
Y = lu(A)
[L,U] = lu(A)
[L,U,P] = lu(A)
[L,U,P,Q] = lu(A)
[L,U,P,Q,R] = lu(A)
[...] = lu(A,'vector')
[...] = lu(A,thresh)
[...] = lu(A,thresh,'vector')
```

Description

The lu function expresses a matrix A as the product of two essentially triangular matrices, one of them a permutation of a lower triangular matrix and the other an upper triangular matrix. The factorization is often called the *LU*, or sometimes the *LR*, factorization. A can be rectangular.

Y = lu(A) returns matrix Y that contains the strictly lower triangular L, i.e., without its unit diagonal, and the upper triangular U as submatrices. That is, if [L,U,P] = lu(A), then Y = U+L-eye(size(A)). The permutation matrix P is not returned.

[L,U] = lu(A) returns an upper triangular matrix in U and a permuted lower triangular matrix in L such that A = L*U. Return value L is a product of lower triangular and permutation matrices.
[L,U,P] = lu(A) returns an upper triangular matrix in U, a lower triangular matrix L with a unit diagonal, and a permutation matrix P, such that L*U = P*A. The statement lu(A,'matrix') returns identical output values.
[L,U,P,Q] = lu(A) for sparse nonempty A, returns a unit lower triangular matrix L, an upper triangular matrix U, a row permutation matrix P, and a column reordering matrix Q, so that P*A*Q = L*U. If A is empty or not sparse, lu displays an error message. The statement lu(A,'matrix') returns identical output values.
[L,U,P,Q,R] = lu(A) returns unit lower triangular matrix L, upper triangular matrix U, permutation matrices P and Q, and a diagonal scaling matrix R so that P*(R\A)*Q = L*U for sparse non-empty A. Typically, but not always, the row-scaling leads to a sparser and more stable factorization. The statement lu(A,'matrix') returns identical output values.
[...] = lu(A,'vector') returns the permutation information in two row vectors p and q. You can specify from 1 to 5 outputs. Output p is defined as A(p,:)=L*U, output q is defined as A(p,q)=L*U, and output R is defined as R(:,p)\A(:,q)=L*U.
[...] = lu(A,thresh) controls pivoting. This syntax applies to sparse matrices only. The thresh input is a one- or two-element vector of type single or double that defaults to [0.1, 0.001]. If A is a square matrix with a mostly symmetric structure and mostly nonzero diagonal, MATLAB® uses a symmetric pivoting strategy. For this strategy, the diagonal where
A(i,j) >= thresh(2) * max(abs(A(j:m,j)))
is selected. If the diagonal entry fails this test, a pivot entry below the diagonal is selected, using thresh(1). In this case, L has entries with absolute value 1/min(thresh) or less.
If A is not as described above, MATLAB uses a nonsymmetric strategy. In this case, the sparsest row i where
A(i,j) >= thresh(1) * max(abs(A(j:m,j)))
is selected. A value of 1.0 results in conventional partial pivoting. Entries in L have an absolute value of 1/thresh(1) or less. The second element of the thresh input vector is not used when MATLAB uses a nonsymmetric strategy.
Smaller values of thresh(1) and thresh(2) tend to lead to sparser LU factors, but the solution can become inaccurate. Larger values can lead to a more accurate solution (but not always), and usually an increase in the total work and memory usage. The statement lu(A,thresh,'matrix') returns identical output values.
[...] = lu(A,thresh,'vector') controls the pivoting strategy and also returns the permutation information in row vectors, as described above. The thresh input must precede 'vector' in the input argument list.

Note

In rare instances, incorrect factorization results in P*A*Q ≠ L*U. Increase thresh, to a maximum of 1.0 (regular partial pivoting), and try again.

Arguments

A	Rectangular matrix to be factored.
thresh	Pivot threshold for sparse matrices. Valid values are in the interval [0,1]. If you specify the fourth output Q, the default is 0.1. Otherwise, the default is 1.0.
L	Factor of A. Depending on the form of the function, L is either a unit lower triangular

	matrix, or else the product of a unit lower triangular matrix with P'.
U	Upper triangular matrix that is a factor of A.
P	Row permutation matrix satisfying the equation L*U = P*A, or L*U = P*A*Q. Used for numerical stability.
Q	Column permutation matrix satisfying the equation P*A*Q = L*U. Used to reduce fill-in in the sparse case.
R	Row-scaling matrix

Examples

Start with

```
A = [ 1    2    3
      4    5    6
      7    8    0 ];
```

To see the LU factorization, call lu with two output arguments.

```
[L1,U] = lu(A)

L1 =
   0.1429    1.0000         0
   0.5714    0.5000    1.0000
   1.0000         0         0

U =
   7.0000    8.0000         0
        0    0.8571    3.0000
        0         0    4.5000
```

Notice that L1 is a permutation of a lower triangular matrix: if you switch rows 2 and 3, and then switch rows 1 and 2, the resulting matrix is lower triangular and has 1s on the diagonal. Notice also that U is upper triangular. To check that the factorization does its job, compute the product

```
L1*U
```

which returns the original A. The inverse of the example matrix, X = inv(A), is actually computed from the inverses of the triangular factors

```
X = inv(U)*inv(L1)
```

Using three arguments on the left side to get the permutation matrix as well,

```
[L2,U,P] = lu(A)
```

returns a truly lower triangular L2, the same value of U, and the permutation matrix P.

```
L2 =

   1.0000         0         0
   0.1429    1.0000         0
   0.5714    0.5000    1.0000

U =
   7.0000    8.0000         0
        0    0.8571    3.0000
        0         0    4.5000
```

```
P =
   0   0   1
   1   0   0
   0   1   0
```

Note that L2 = P*L1.

```
P*L1

ans =

   1.0000         0         0
   0.1429    1.0000         0
   0.5714    0.5000    1.0000
```

To verify that L2*U is a permuted version of A, compute L2*U and subtract it from P*A:

```
P*A - L2*U

ans =
    0    0    0
    0    0    0
    0    0    0
```

In this case, inv(U)*inv(L) results in the permutation of inv(A) given by inv(P)*inv(A).

The determinant of the example matrix is

```
d = det(A)

d =    27
```

It is computed from the determinants of the triangular factors

d = det(L)*det(U)

The solution to $Ax = b$ is obtained with matrix division

x = A\b

The solution is actually computed by solving two triangular systems

y = L\b

x = U\y

Cholesky factorization

Syntax

```
R = chol(A)
L = chol(A,'lower')
R = chol(A,'upper')
[R,p] = chol(A)
[L,p] = chol(A,'lower')
[R,p] = chol(A,'upper')
[R,p,S] = chol(A)
[R,p,s] = chol(A,'vector')
```

[L,p,s] = chol(A,'lower','vector')
[R,p,s] = chol(A,'upper','vector')

Description

R = chol(A) produces an upper triangular matrix R from the diagonal and upper triangle of matrix A, satisfying the equation R'*R=A. The chol function assumes that A is (complex Hermitian) symmetric. If it is not, chol uses the (complex conjugate) transpose of the upper triangle as the lower triangle. Matrix A must be positive definite.

L = chol(A,'lower') produces a lower triangular matrix L from the diagonal and lower triangle of matrix A, satisfying the equation L*L'=A. The chol function assumes that A is (complex Hermitian) symmetric. If it is not, chol uses the (complex conjugate) transpose of the lower triangle as the upper triangle. When A is sparse, this syntax of chol is typically faster. Matrix A must be positive definite. R = chol(A,'upper') is the same as R = chol(A).

[R,p] = chol(A) for positive definite A, produces an upper triangular matrix R from the diagonal and upper triangle of matrix A, satisfying the equation R'*R=A and p is zero. If A is not positive definite, then p is a positive integer and MATLAB® does not generate an error. When A is full, R is an upper triangular matrix of order q=p-1 such that R'*R=A(1:q,1:q). When A is sparse, R is an upper triangular matrix of size q-by-n so that the L-shaped region of the first q rows and first q columns of R'*R agree with those of A.

[L,p] = chol(A,'lower') for positive definite A, produces a lower triangular matrix L from the diagonal and lower triangle of matrix A, satisfying the equation L*L'=A and p is zero. If A is not positive definite, then p is a positive integer and MATLAB does not generate an error. When A is full, L is a lower triangular matrix of order q=p-1 such that L*L'=A(1:q,1:q). When A is sparse, L is a lower triangular matrix of size q-by-n so that the L-shaped region of the first q rows and first q columns of L*L' agree with those of A. [R,p] = chol(A,'upper') is the same as [R,p] = chol(A).

The following three-output syntaxes require sparse input A.

[R,p,S] = chol(A), when A is sparse, returns a permutation matrix S. Note that the preordering S may differ from that obtained from amd since chol will slightly change the ordering for increased performance. When p=0, R is an upper triangular matrix such that R'*R=S'*A*S. When p is not zero, R is an upper triangular matrix of size q-by-n so that the L-shaped region of the first q rows and first q columns of R'*R agree with those of S'*A*S. The factor of S'*A*S tends to be sparser than the factor of A.

[R,p,s] = chol(A,'vector'), when A is sparse, returns the permutation information as a vector s such that A(s,s)=R'*R, when p=0. You can use the 'matrix' option in place of 'vector' to obtain the default behavior.

[L,p,s] = chol(A,'lower','vector'), when A is sparse, uses only the diagonal and the lower triangle of A and returns a lower triangular matrix L and a permutation vector ssuch that A(s,s)=L*L', when p=0. As above, you can use the 'matrix' option in place of 'vector' to obtain a permutation matrix. [R,p,s] = chol(A,'upper','vector') is the same as [R,p,s] = chol(A,'vector').

Example 1

The gallery function provides several symmetric, positive, definite matrices.

```
A=gallery('moler',5)

A =

     1    -1    -1    -1    -1
    -1     2     0     0     0
    -1     0     3     1     1
    -1     0     1     4     2
    -1     0     1     2     5

C=chol(A)

ans =

     1    -1    -1    -1    -1
     0     1    -1    -1    -1
     0     0     1    -1    -1
     0     0     0     1    -1
     0     0     0     0     1
isequal(C'*C,A)

ans =

     1
```

For sparse input matrices, chol returns the Cholesky factor.

```
N = 100;
```

```
A = gallery('poisson', N);
```

N represents the number of grid points in one direction of a square N-by-N grid.
Therefore, A is N^2 by N^2.

```
L = chol(A, 'lower');
```

```
D = norm(A - L*L', 'fro');
```

The value of D will vary somewhat among different versions of MATLAB but will be on order of 10^{-14}

Orthogonal-triangular decomposition

Syntax

```
[Q,R] = qr(A)
[Q,R] = qr(A,0)
[Q,R,E] = qr(A)
[Q,R,E] = qr(A,'matrix')
[Q,R,e] = qr(A,'vector')
[Q,R,e] = qr(A,0)
X = qr(A)
X = qr(A,0)
R = qr(A)
R = qr(A,0)
[C,R] = qr(A,B)
[C,R,E] = qr(A,B)
[C,R,E] = qr(A,B,'matrix')
[C,R,e] = qr(A,B,'vector')
[C,R] = qr(A,B,0)
[C,R,e] = qr(A,B,0)
```

Description

[Q,R] = qr(A), where A is m-by-n, produces an m-by-n upper triangular matrix R and an m-by-m unitary matrix Q so that A = Q*R.

[Q,R] = qr(A,0) produces the economy-size decomposition. If m > n, only the first n columns of Q and the first n rows of R are computed. If m<=n, this is the same as [Q,R] = qr(A).

If A is full:

[Q,R,E] = qr(A) or [Q,R,E] = qr(A,'matrix') produces unitary Q, upper triangular R and a permutation matrix E so that A*E = Q*R. The column permutation E is chosen so that abs(diag(R)) is decreasing.

[Q,R,e] = qr(A,'vector') returns the permutation information as a vector instead of a matrix. That is, e is a row vector such that A(:,e) = Q*R.

[Q,R,e] = qr(A,0) produces an economy-size decomposition in which e is a permutation vector, so that A(:,e) = Q*R.

X = qr(A) and X = qr(A,0) return a matrix X such that triu(X) is the upper triangular factor R.

If A is sparse:

R = qr(A) computes a Q-less QR decomposition and returns the upper triangular factor R. Note that R = chol(A'*A). Since Q is often nearly full, this is preferred to [Q,R] = QR(A).

R = qr(A,0) produces economy-size R. If m>n, R has only n rows. If m<=n, this is the same as R = qr(A).

[Q,R,E] = qr(A) or [Q,R,E] = qr(A,'matrix') produces unitary Q, upper triangular R and a permutation matrix E so that A*E = Q*R. The column permutation E is chosen to reduce fill-in in R.

[Q,R,e] = qr(A,'vector') returns the permutation information as a vector instead of a matrix. That is, e is a row vector such that A(:,e) = Q*R.

[Q,R,e] = qr(A,0) produces an economy-size decomposition in which e is a permutation vector, so that A(:,e) = Q*R.

[C,R] = qr(A,B), where B has as many rows as A, returns C = Q'*B. The least-squares solution to A*X = B is X = R\C.

[C,R,E] = qr(A,B) or [C,R,E] = qr(A,B,'matrix'), also returns a fill-reducing ordering. The least-squares solution to A*X = B is X = E*(R\C).

[C,R,e] = qr(A,B,'vector') returns the permutation information as a vector instead of a matrix. That is, the least-squares solution to A*X = B is X(e,:) = R\C.

[C,R] = qr(A,B,0) produces economy-size results. If m>n, C and R have only n rows. If m<=n, this is the same as [C,R] = qr(A,B).

[C,R,e] = qr(A,B,0) additionally produces a fill-reducing permutation vector e. In this case, the least-squares solution to A*X = B is X(e,:) = R\C.

Examples

Find the least squares approximate solution to A*x = b with the Q-less QR decomposition and one step of iterative refinement:

ifissparse(A), R = qr(A);

else R = triu(qr(A)); end

x = R\(R'\(A'*b));

r = b - A*x;

err = R\(R'\(A'*r));

x = x + err;

Matrix Operations

There are also routines that let you find solutions to equations. For example, if Ax=b and you want to find x, a slow way to find x is to simply invert A and perform a left multiply on both sides (more on that later). It turns out that there are more efficient and more stable methods to do this (L/U decomposition with pivoting, for example). Matlab has special commands that will do this for you.

Before finding the approximations to linear systems, it is important to remember that if A and B are both matrices, then AB is not necessarily equal to BA. To distinguish the difference between solving systems that have a right or left multiply, Matlab uses two different operators, "/" and "\". Examples of their use are given below. It is left as an exercise for you to figure out which one is doing what.

```
>> v = [1 3 5]'

v =

     1
     3
     5

>> x = A\v

Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 4.565062e-18


x =

   1.0e+15 *

    1.8014
   -3.6029
    1.8014

>> x = B\v
```

```
x =

     2
     1
    -1

>> B*x

ans =

     1
     3
     5

>>x1 = v'/B

x1 =

    4.0000   -3.0000    1.0000


>>x1*B

ans =

    1.0000    3.0000    5.0000
```

Finally, sometimes you would like to clear all of your data and start over. You do this with the "clear" command. Be careful though, it does not ask you for a second opinion and its results are **final**.

```
>>clear

>>whos
```

Part B (5x8=40 Marks)

Possible Questions

1. Mention the procedure for solving a linear system of equation with example.
2. Describe in detail finding the eigen values and eigen vectors with example.
3. Explain about matrix factorization with example.
4. Describe about matrix operation in linear algebra with example.
5. Write a short note on straight line fit with example.
6. Define Gauss elimination with examples.
7. Describe about the curve fitting with polynomial functions
8. Describe in detail about curve fitting with polynomial functions.
9. Describe in detail solving a system of equations in MAT LAB with example.
10. Describe about the interpretation with example

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021

Subject: MATLAB programming **Subject Code: 15MMU504**

Class : III - B.Sc. Mathematics **Semester : V**

Unit IV

Applications

Part A (20x1=20 Marks)
(Question Nos. 1 to 20 Online Examinations)

Possible Questions

Question	Choice 1	Choice 2	Choice 3	Choice 4	Answer
____ command is used to solve linear systems if matrix A has exploitable structure	dsolve	linsolve	solve	linear	linsolve
linsolve command is used to solve linear systems if matrix A has _____ structure	exploitable	decimal	complex	triangular	exploitable
The Command for determinant is ____	dsolve	det	mod	eye	det
To evaluates f(x) at x =5 is executed in MATLAB as	x=f(5)	f(x)=5	fx(5)	f=5	fx(5)
To evaluates f(x,y) at x =2 and y = 3 is executed in MATLAB as	fxy=(2,3)	fxy(2,3)	fx(2)y(3)	f(2,3)	fxy(2,3)
_____ command is used to produce stunning surface plots in 3 D	ezpolar	ezcontour	ezcontourf	ezsurf	ezsurf
To solve the matrix equation in Mat lab we use _____ command	x = A\b	x = A*b	$x = A^{-1}b$	x = A/b	x = A\b

To determine the eigen value for matrix A	A = eigen (A)	eig(A)	eign(A)	Eig(A,:)	eig(A)
To solve the LU factorization by using the command	[L,U] = lu(A);	{LU}=LU(A);	[l,u]=A;	lu=[A];	[L,U] = lu(A);
To solve the QR factorization by using the command	{q,r} = [A]	{Q,R} = [A]	[Q,R] = qr(A)	[Q,R] = A	[Q,R] = qr(A)
____ bulit in function is used to solve cholesky factorization	ch	chol	chl	chlf	chol
_____ command is used to solve singular value decomposition	[U,D,V] = svd(A)	svd = [A]	[U,D,V] = A	{u,d,v} = SVD(A)	[U,D,V] = svd(A)
interp1 command is used ____ data interpolation	1D	2D	3D	fourier transform	1D
Gauss jordan reduction procedure is then used to transform the augmented matrix is called ___	row reduced echelon form	gauss seidal	gauss elimination	augmented matrix	row reduced echelon form
_____ reduction procedure is then used to transform the augmented matrix is called row reduced echelon form	augmented matrix	gauss elimination	gauss seidal	Gauss jordan	Gauss jordan
Gauss jordan reduction procedure is then used to transform the _____ is called row reduced echelon form	gauss seidal	square matrix	augmented matrix	identity matrix	augmented matrix
R = chol(A) this command is used to solve	LU factorization	QR factorization	Cholesky factorization	Singular value decomposition	Cholesky factorization
[U, D, V] = svd(A) is use to solve	LU factorization	QR factorization	Cholesky factorization	Singular value decomposition	Singular value decomposition
[Q,R] = qr(A) this command is used to solve	LU factorization	QR factorization	Cholesky factorization	Singular value decomposition	QR factorization
[L,U] = lu(A) this command is used to solve	LU factorization	QR factorization	Cholesky factorization	Singular value decomposition	LU factorization

Which of the following command is used to solve linear algebra ?	nzmax	eigs	speye	spy	eigs
which of the following function is used for graphs ?	gplot	etree	speye	spy	gplot
interp2 command is used ____ data interpolation	1D	2D	3D	fourier transform	2D
interp3 command is used ____ data interpolation	1D	2D	3D	fourier transform	3D
interpft command is used ____ data interpolation	1D	2D	3D	fourier transform	fourier transform
spline command is used _____ interpolation that uses cubic spline fit.	1D	2D	3D	fourier transform	1D
spline command is used 1-D interpolation that uses_____.	square spline	cubic spline fit	least square fit	straight-line fit	cubic spline fit
_____ command is used 1-D interpolation that uses cubic spline fit.	interp3	interpft	interp2	spline	spline
____ error are caused mainly due to wrong logic used by the program.	syntax	condition	runtime	statement	runtime

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University Established Under Section 3 of UGC Act 1956)

Pollachi Main Road, Eachanari (Po),

Coimbatore –641 021.

Department of Mathematics

Subject : MATLAB programming	**Subject Code : 15MMU504**	**L T P C**
Class : III – B.Sc. Mathematics	**Semester : V**	**5 0 0 5**

UNIT V

Applications – Data Analysis and Statistics – Numerical Integration – ordinary differential equations – Nonlinear Algebraic Equations.

TEXT BOOK

1. RudraPratap, 2003. Getting Started with MATLAB-A Quick Introduction for Scientists and Engineers, Oxford University Press.

REFERENCES

1. William John Palm, 2005. Introduction to Matlab 7 for Engineers, McGraw-Hill Professional.New Delhi.

2. Dolores M. Etter, David C. Kuncicky, 2004.Introduction to MATLAB 7, Prentice Hall, New Delhi.

3. Kiranisingh.Y,Chaudhuri.B.B, 2007.Matlab Programming, Prentice-Hall Of India Pvt.Ltd, New Delhi.

4. Brian Hahn, 2016. Essential MATLAB for Engineers and Scientists.6thedition , Elsevier publication.

UNIT – V

Application of MATLAB

Data Analysis and Statistics

For performing simple data analysis tasks, such as finding mean, median, and standard deviation, MATLAB provides an easy graphical interface that you can activatefrom the Tools menu of the figure window. First , you should plot your data inthe form you wish (e.g. , scatter plot , line plot)_ Then, go to the figure window an select Data Statistics from the Too ls pull-down menu. MATLAB shows you thebasic statistics of your data in a separate window marked Data Statistics. You canshow any of the statistical measures on your plot by checking the appropriate box. However, you are not limited to this simple interface for your statistical needs.Several built-in functions are at your disposal for statistical calculations. Thesefunctions are briefly discussed later.All data analysis functions take both vectors and matrices as arguments. Whena vector is given as an argument , it does not matter whether it is a row vector ora column vector. However, when a matrix is used as an argument , the functionsoperatecolumnwise on the matrix and output a row vector that contains results ofthe operation on each column.

Statistics Function Summary

Function	Description
max	Maximum value
mean	Average or mean value
median	Median value
min	Smallest value
mode	Most frequent value
std	Standard deviation
var	Variance, which measures the spread or dispersion of the values

Examples

1. Calculating Maximum, Mean, and Standard Deviation

This example shows how to use MATLAB functions to calculate the maximum, mean, and standard deviation values for a 24-by-3 matrix called count. MATLAB computes these statistics independently for each column in the matrix.

```
% Load the sample data
load count.dat
% Find the maximum value in each column
mx = max(count)
% Calculate the mean of each column
mu = mean(count)
% Calculate the standard deviation of each column
sigma = std(count)
```

The results are

```
mx =
        114         145         257

mu =
    32.0000     46.5417     65.5833

sigma =
    25.3703     41.4057     68.0281
```

To get the row numbers where the maximum data values occur in each data column, specify a second output parameter indx to return the row index. For example:

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

These results are

```
mx =
    114     145     257

indx =
    20    20    20
```

Here, the variable mx is a row vector that contains the maximum value in each of the three data columns. The variable indx contains the row indices in each column that correspond to the maximum values.

To find the minimum value in the entire count matrix, 24-by-3 matrix into a 72-by-1 column vector by using the syntax count(:). Then, to find the minimum value in the single column, use the following syntax:

```
min(count(:))

ans =
    7
```

2. Subtracting the Mean

Subtract the mean from each column of the matrix by using the following syntax:

```
% Get the size of the count matrix
[n,p] = size(count)
% Compute the mean of each column
mu = mean(count)
% Create a matrix of mean values by
% replicating the mu vector for n rows
MeanMat = repmat(mu,n,1)
% Subtract the column mean from each element
% in that column
x = count - MeanMat
```

Numerical Integration

Functions

integral	Numerical integration
integral2	Numerically evaluate double integral
integral3	Numerically evaluate triple integral
quadgk	Numerically evaluate integral, adaptive Gauss-Kronrod quadrature
quad2d	Numerically evaluate double integral, tiled method
cumtrapz	Cumulative trapezoidal numerical integration
trapz	Trapezoidal numerical integration
polyint	Polynomial integration
del2	Discrete Laplacian
diff	Differences and Approximate Derivatives
gradient	Numerical gradient
polyder	Polynomial differentiation

Numerical integration

Syntax

```
q = integral(fun,xmin,xmax)

q = integral(fun,xmin,xmax,Name,Value)
```

Description

q = integral(fun,xmin,xmax) numerically integrates function fun from xmin to xmax using global adaptive quadrature and default error tolerances.

q = integral(fun,xmin,xmax,Name,Value) specifies additional options with one or more Name,Value pair arguments. For example, specify 'WayPoints' followed by a vector of real or complex numbers to indicate specific points for the integrator to use.

Examples

Improper Integral

Try this Example

Create the function $f(x) = e^{-x^2}(\ln x)^2$.

```
fun = @(x) exp(-x.^2).*log(x).^2;
```

Evaluate the integral from x=0 to x=Inf.

```
q = integral(fun,0,Inf)
q = 1.9475
```

Parameterized Function

Try this Example

Create the function $f(x) = 1/(x^3 - 2x - c)$ with one parameter, c.

```
fun = @(x,c) 1./(x.^3-2*x-c);
```

Evaluate the integral from x=0 to x=2 at c=5.

```
q = integral(@(x)fun(x,5),0,2)
q = -0.4605
```

Singularity at Lower Limit

Try this Example

Create the function $f(x) = \ln(x)$.

```
fun = @(x)log(x);
```

Evaluate the integral from x=0 to x=1 with the default error tolerances.

```
format long
q1 = integral(fun,0,1)
q1 =
  -1.000000010959678
```

Evaluate the integral again, specifying 12 decimal places of accuracy.

```
q2 = integral(fun,0,1,'RelTol',0,'AbsTol',1e-12)
q2 =
  -1.000000000000010
```

Numerically evaluate double integral

Syntax

```
q = integral2(fun,xmin,xmax,ymin,ymax)

q = integral2(fun,xmin,xmax,ymin,ymax,Name,Value)
```

Description

q = integral2(fun,xmin,xmax,ymin,ymax) approximates the integral of the function z = fun(x,y) over the planar region xmin ≤ x ≤ xmax and ymin(x) ≤ y ≤ ymax(x).

q = integral2(fun,xmin,xmax,ymin,ymax,Name,Value) specifies additional options with one or more Name,Value pair arguments.

Examples

Integrate Triangular Region with Singularity at the Boundary

The function

$$f(x,y)=\frac{1}{(\sqrt{x+y})(1+x+y)}$$

is undefined when x and y are zero. integral2 performs best when singularities are on the integration boundary.

Create the anonymous function.

```
fun = @(x,y) 1./( sqrt(x + y) .* (1 + x + y).^2 )
fun = function_handle with value:
   @(x,y)1./(sqrt(x+y).*(1+x+y).^2)
```

Integrate over the triangular region bounded by $0 \le x \le 1$ and $0 \le y \le 1-x$.

```
ymax = @(x) 1 - x;
q = integral2(fun,0,1,0,ymax)
q = 0.2854
```

Evaluate Double Integral in Polar Coordinates

Try this Example

Define the function

$$f(\theta,r)=\frac{r}{\sqrt{r\cos\theta+r\sin\theta}\,(1+r\cos\theta+r\sin\theta)^2}$$

```
fun = @(x,y) 1./( sqrt(x + y) .* (1 + x + y).^2 );
polarfun = @(theta,r) fun(r.*cos(theta),r.*sin(theta)).*r;
```

Define a function for the upper limit of r.

```
rmax = @(theta) 1./(sin(theta) + cos(theta));
```

Integrate over the region bounded by $0 \le \theta \le \pi/2$ and $0 \le r \le r_{max}$.

```
q = integral2(polarfun,0,pi/2,0,rmax)
q = 0.2854
```

Evaluate Double Integral of Parameterized Function with Specific Method and Error Tolerance

Try this Example

Create the anonymous parameterized function $f(x,y)=ax^2+by^2$ with parameters $a=3$ and $b=5$.

```
a = 3;
b = 5;
fun = @(x,y) a*x.^2 + b*y.^2;
```

Evaluate the integral over the region $0 \le x \le 5$ and $-5 \le y \le 0$. Specify the 'iterated' method and approximately 10 significant digits of accuracy.

```
format long
q = integral2(fun,0,5,-5,0,'Method','iterated',...
```

'AbsTol',0,'RelTol',1e-10)
q =
1.666666666666666e+03

Numerically evaluate double integral

Syntax

q = quad2d(fun,a,b,c,d)
[q,errbnd] = quad2d(...)
q = quad2d(fun,a,b,c,d,param1,val1,param2,val2,...)

Description

q = quad2d(fun,a,b,c,d) approximates the integral of fun(x,y) over the planar region $a \le x \le b$ and $c(x) \le y \le d(x)$. fun is a function handle, c and d may each be a scalar or a function handle.

All input functions must be vectorized. The function Z=fun(X,Y) must accept 2-D matrices X and Y of the same size and return a matrix Z of corresponding values. The functions ymin=c(X) and ymax=d(X) must accept matrices and return matrices of the same size with corresponding values.

[q,errbnd] = quad2d(...). errbnd is an approximate upper bound on the absolute error, |Q - I|, where I denotes the exact value of the integral.

q = quad2d(fun,a,b,c,d,param1,val1,param2,val2,...) performs the integration as above with specified values of optional parameters:

AbsTol	absolute error tolerance
RelTol	relative error tolerance

quad2d attempts to satisfy ERRBND <= max(AbsTol,RelTol*|Q|). This is absolute error control when |Q| is sufficiently small and relative error control when |Q| is larger. A default tolerance value is used when a tolerance is not specified. The default value of AbsTol is 1e-5. The default value of RelTol is 100*eps(class(Q)). This is also the minimum value of RelTol. Smaller RelTol values are automatically increased to the default value.

MaxFunEvals	Maximum allowed number of evaluations of fun reached.

The MaxFunEvals parameter limits the number of vectorized calls to fun. The default is 2000.

FailurePlot	Generate a plot if MaxFunEvals is reached.

Setting FailurePlot to true generates a graphical representation of the regions needing further refinement when MaxFunEvals is reached. No plot is generated if the integration succeeds before reaching MaxFunEvals. These (generally) 4-sided regions are mapped to rectangles internally. Clusters of small regions indicate the areas of difficulty. The default is false.

Singular	Problem may have boundary singularities

With Singular set to true, quad2d will employ transformations to weaken boundary singularities for better performance. The default is true. Setting Singular to false will turn these transformations off, which may provide a performance benefit on some smooth problems.

Examples

Evaluate Double Integral

Integrate

$y\sin(x) + x\cos(y)$

over $-\pi \le x \le 2\pi$ and $0 \le y \le \pi$.

```
fun = @(x,y) y.*sin(x)+x.*cos(y);
Q = quad2d(fun,pi,2*pi,0,pi)
Q = -9.8696
```

Compare the result to the true value of the integral, $-\pi^2$.

```
-pi^2
ans = -9.8696
```

Integrand with Singularity on Integration Boundary

Integrate the function

$[(x+y)^{1/2}(1+x+y)^2]^{-1}$

over the region $0 \le x \le 1$ and $0 \le y \le 1-x$. This integrand is infinite at the origin (0,0), which lies on the boundary of the integration region.

```
fun = @(x,y) 1./(sqrt(x + y) .* (1 + x + y).^2 );
ymax = @(x) 1 - x;
Q = quad2d(fun,0,1,0,ymax)
Q = 0.2854
```

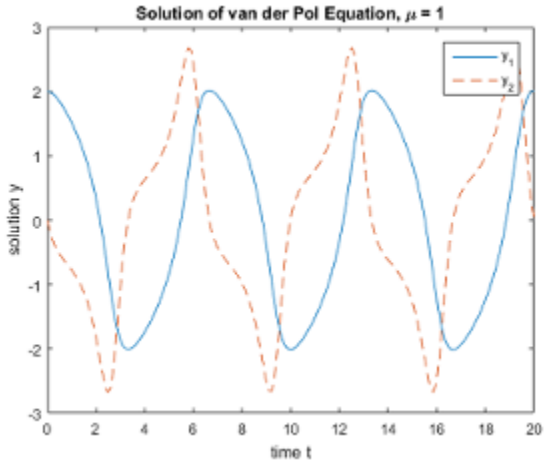
The true value of the integral is $\pi/4 - 1/2$.

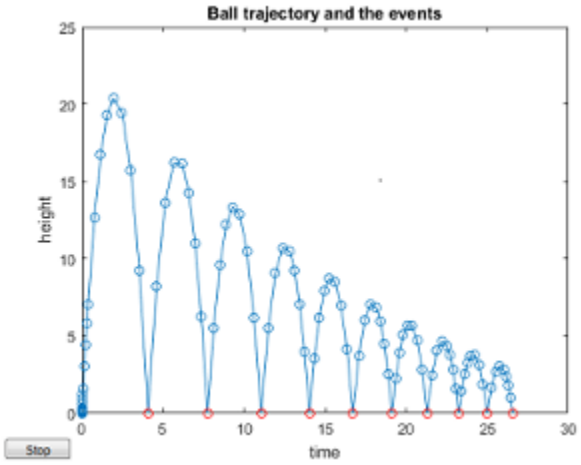
```
pi/4 - 0.5
ans = 0.2854
```

Ordinary Differential Equations

Ordinary differential equation initial value problem solvers

The Ordinary Differential Equation (ODE) solvers in MATLABsolve initial value problems with a variety of properties. The solvers can work on stiff or nonstiff problems, problems with a mass matrix, differential algebraic equations (DAEs), or fully implicit problems





Functions

Nonstiff Solvers

ode45	Solve nonstiff differential equations — medium order method
ode23	Solve nonstiff differential equations — low order method
ode113	Solve nonstiff differential equations — variable order method

ode23
Solve nonstiff differential equations — low order method

Syntax
[t,y] = ode23(odefun,tspan,y0)
[t,y] = ode23(odefun,tspan,y0,options)
[t,y,te,ye,ie] = ode23(odefun,tspan,y0,options)
sol = ode23(___)
Description

[t,y] = ode23(odefun,tspan,y0), where tspan = [t0 tf], integrates the system of differential equations $y'=f(t,y)$ from t0 to tf with initial conditions y0. Each row in the solution array y corresponds to a value returned in column vector t.
All MATLAB® ODE solvers can solve systems of equations of the form $y'=f(t,y)$, or problems that involve a mass matrix, $M(t,y)y'=f(t,y)$. The solvers all use similar syntaxes.
The ode23s solver only can solve problems with a mass matrix if the mass matrix is constant. ode15s and ode23t can solve problems with a mass matrix that is singular, known as differential-algebraic equations (DAEs). Specify the mass matrix using the Mass option of odeset.

[t,y] = ode23(odefun,tspan,y0,options) also uses the integration settings defined by options, which is an argument created using the odeset function. For example, use the AbsTol and RelTol options to specify absolute and relative error tolerances, or the Mass option to provide a mass matrix.
[t,y,te,ye,ie] = ode23(odefun,tspan,y0,options) additionally finds where functions of (t,y), called event functions, are zero. In the output, te is the time of the event, ye is the solution at the time of the event, and ie is the index of the triggered event.
For each event function, specify whether the integration is to terminate at a zero and whether the direction of the zero crossing matters. Do this by setting the 'Events'property to a function, such as myEventFcn or @myEventFcn, and creating a corresponding function: [value,isterminal,direction] = myEventFcn(t,y). For more information, see ODE Event Location.
sol = ode23(___) returns a structure that you can use with deval to evaluate the solution at any point on the interval [t0 tf]. You can use any of the input argument combinations in previous syntaxes.

Examples

ODE with Single Solution Component

Try this Example

Simple ODEs that have a single solution component can be specified as an anonymous function in the call to the solver. The anonymous function must accept two inputs (t,y) even if one of the inputs is not used.

Solve the ODE

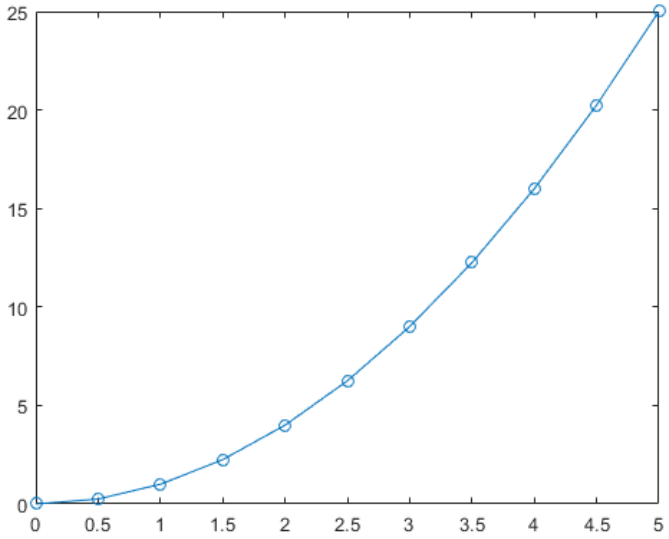
$y' = 2t.$

Use a time interval of [0,5] and the initial condition y0 = 0.

```
tspan = [0 5];
y0 = 0;
[t,y] = ode23(@(t,y) 2*t, tspan, y0);
```

Plot the solution.

```
plot(t,y,'-o')
```



Solve Nonstiff Equation

The van der Pol equation is a second order ODE

$y_1'' - \mu\left(1 - y_1^2\right) y_1' + y_1 = 0,$

where $\mu > 0$ is a scalar parameter. Rewrite this equation as a system of first-order ODEs by making the substitution $y_1' = y_2$. The resulting system of first-order ODEs is

$y_1' = y_2$

$y_2' = \mu(1 - y_1^2)y_2 - y_1.$

The function file vdp1.m represents the van der Pol equation using $\mu = 1$. The variables y_1 and y_2 are the entries y(1) and y(2) of a two-element vector, dydt.

```
function dydt = vdp1(t,y)
%VDP1  Evaluate the van der Pol ODEs for mu = 1
%
%   See also ODE113, ODE23, ODE45.
```

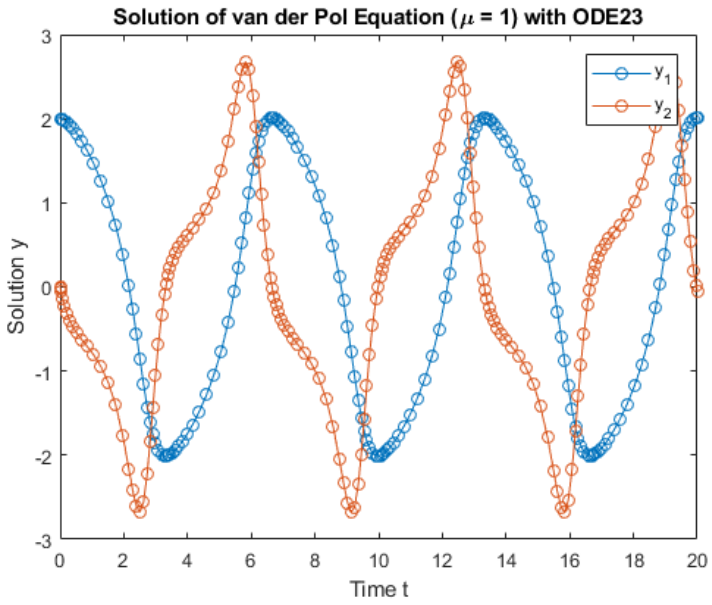
```
%   JacekKierzenka and Lawrence F. Shampine
%   Copyright 1984-2014 The MathWorks, Inc.

dydt = [y(2); (1-y(1)^2)*y(2)-y(1)];
```

Solve the ODE using the ode23 function on the time interval [0 20] with initial values [2 0]. The resulting output is a column vector of time points t and a solution array y. Each row in y corresponds to a time returned in the corresponding row of t. The first column of y corresponds to y_1, and the second column to y_2.

```
[t,y] = ode23(@vdp1,[0 20],[2; 0]);
```

Plot the solutions for y_1 and y_2 against t.

```
plot(t,y(:,1),'-o',t,y(:,2),'-o')
title('Solution of van der Pol Equation (\mu = 1) with ODE23');
xlabel('Time t');
ylabel('Solution y');
legend('y_1','y_2')
```



ode45

Solve nonstiff differential equations — medium order method

Syntax

```
[t,y] = ode45(odefun,tspan,y0)
[t,y] = ode45(odefun,tspan,y0,options)
[t,y,te,ye,ie] = ode45(odefun,tspan,y0,options)
sol = ode45(___)
```

Description

[t,y] = ode45(odefun,tspan,y0), where tspan = [t0 tf], integrates the system of differential equations $y'=f(t,y)$ from t0 to tf with initial conditions y0. Each row in the solution array y corresponds to a value returned in column vector t.

All MATLAB® ODE solvers can solve systems of equations of the form $y'=f(t,y)$, or problems that involve a mass matrix, $M(t,y)y'=f(t,y)$. The solvers all use similar syntaxes.
The ode23s solver only can solve problems with a mass matrix if the mass matrix is constant. ode15s and ode23t can solve problems with a mass matrix that is singular, known as differential-algebraic equations (DAEs). Specify the mass matrix using the Mass option of odeset.
ode45 is a versatile ODE solver and is the first solver you should try for most problems. However, if the problem is stiff or requires high accuracy, then there are other ODE solvers that might be better suited to the problem. See Choose an ODE Solver for more information.

[t,y] = ode45(odefun,tspan,y0,options) also uses the integration settings defined by options, which is an argument created using the odeset function. For example, use the AbsTol and RelTol options to specify absolute and relative error tolerances, or the Mass option to provide a mass matrix.
[t,y,te,ye,ie] = ode45(odefun,tspan,y0,options) additionally finds where functions of (t,y), called event functions, are zero. In the output, te is the time of the event, ye is the solution at the time of the event, and ie is the index of the triggered event.
For each event function, specify whether the integration is to terminate at a zero and whether the direction of the zero crossing matters. Do this by setting the 'Events'property to a function, such as myEventFcn or @myEventFcn, and creating a corresponding function: [value,isterminal,direction] = myEventFcn(t,y). For more information, see ODE Event Location.

sol = ode45(___) returns a structure that you can use with deval to evaluate the solution at any point on the interval [t0 tf]. You can use any of the input argument combinations in previous syntaxes.

Examples

ODE with Single Solution Component

Simple ODEs that have a single solution component can be specified as an anonymous function in the call to the solver. The anonymous function must accept two inputs (t,y) even if one of the inputs is not used.
Solve the ODE

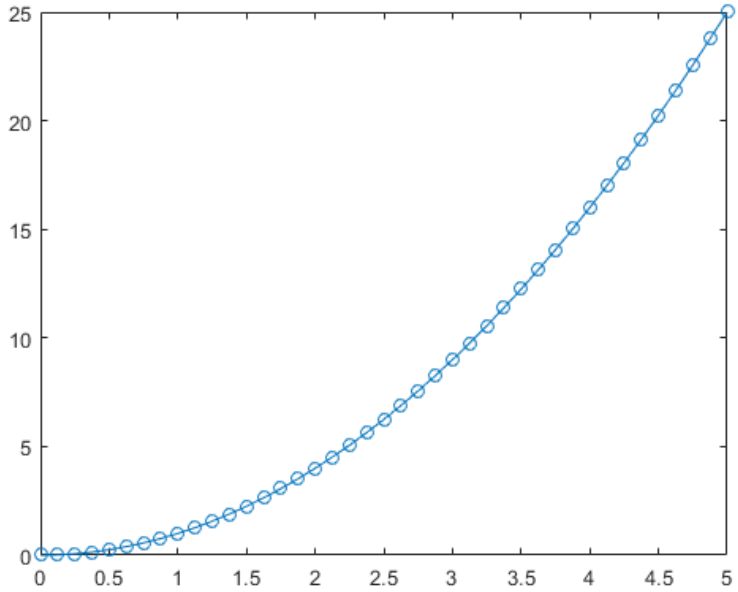
$$y' = 2t.$$

Use a time interval of [0,5] and the initial condition y0 = 0.

```
tspan = [0 5];
y0 = 0;
[t,y] = ode45(@(t,y) 2*t, tspan, y0);
```

Plot the solution.

```
plot(t,y,'-o')
```

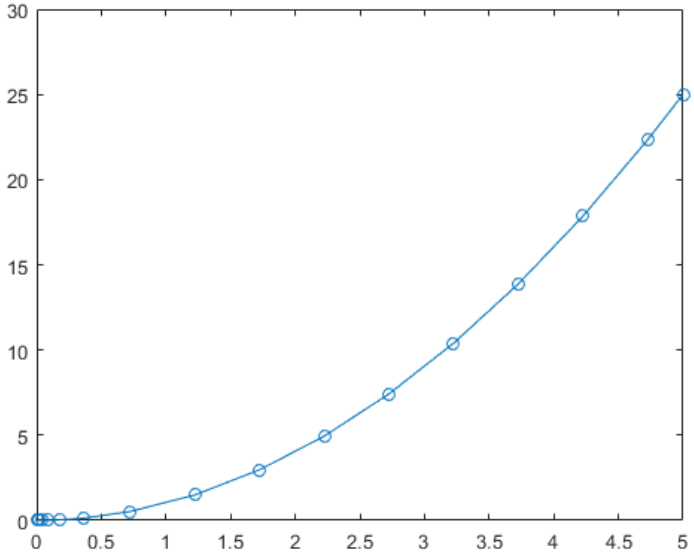


ode113
Solve nonstiff differential equations — variable order method

Syntax
[t,y] = ode113(odefun,tspan,y0)
[t,y] = ode113(odefun,tspan,y0,options)
[t,y,te,ye,ie] = ode113(odefun,tspan,y0,options)
sol = ode113(___)

Examples

ODE with Single Solution Component
Simple ODEs that have a single solution component can be specified as an anonymous function in the call to the solver. The anonymous function must accept two inputs (t,y) even if one of the inputs is not used.
Solve the ODE
$y' = 2t.$
Use a time interval of [0,5] and the initial condition y0 = 0.
tspan = [0 5];
y0 = 0;
[t,y] = ode113(@(t,y) 2*t, tspan, y0);
Plot the solution.
plot(t,y,'-o')



Stiff Solvers

ode15s	Solve stiff differential equations and DAEs — variable order method
ode23s	Solve stiff differential equations — low order method
ode23t	Solve moderately stiff ODEs and DAEs — trapezoidal rule
ode23tb	Solve stiff differential equations — trapezoidal rule + backward differentiation formula

ode15s
Solve stiff differential equations and DAEs — variable order method

Syntax
[t,y] = ode15s(odefun,tspan,y0)
[t,y] = ode15s(odefun,tspan,y0,options)
[t,y,te,ye,ie] = ode15s(odefun,tspan,y0,options)
sol = ode15s(___)
Description

[t,y] = ode15s(odefun,tspan,y0), where tspan = [t0 tf], integrates the system of differential equations $y'=f(t,y)$ from t0 to tf with initial conditions y0. Each row in the solution array y corresponds to a value returned in column vector t.
All MATLAB® ODE solvers can solve systems of equations of the form $y'=f(t,y)$, or problems that involve a mass matrix, $M(t,y)y'=f(t,y)$. The solvers all use similar syntaxes. The ode23s solver only can solve problems with a mass matrix if the mass matrix is constant. ode15s and ode23t can solve problems with a mass matrix that is singular, known as differential-algebraic equations (DAEs). Specify the mass matrix using the Mass option of odeset.

Examples

ODE With Single Solution Component

Try this Example

Simple ODEs that have a single solution component can be specified as an anonymous function in the call to the solver. The anonymous function must accept two inputs (t,y) even if one of the inputs is not used.

Solve the ODE

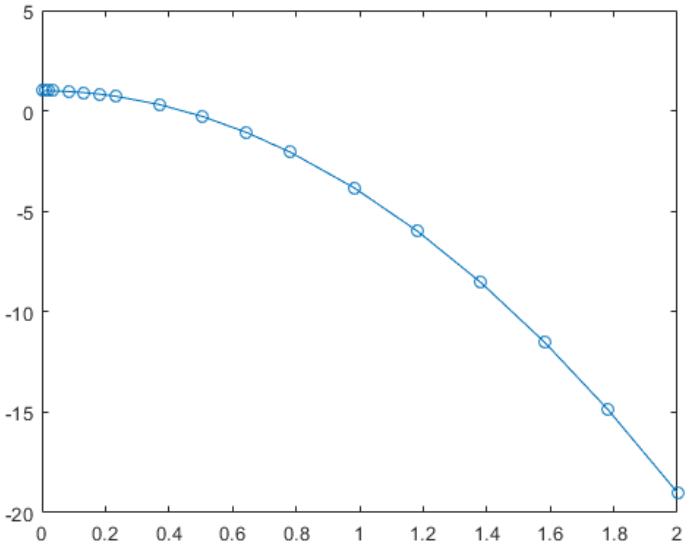
$y' = -10t.$

Use a time interval of [0,2] and the initial condition y0 = 1.

```
tspan = [0 2];
y0 = 1;
[t,y] = ode15s(@(t,y) -10*t, tspan, y0);
```

Plot the solution.

```
plot(t,y,'-o')
```



ode23t

Solve moderately stiff ODEs and DAEs — trapezoidal rule

Syntax

```
[t,y] = ode23t(odefun,tspan,y0)
[t,y] = ode23t(odefun,tspan,y0,options)
[t,y,te,ye,ie] = ode23t(odefun,tspan,y0,options)
sol = ode23t(___)
```

Description

[t,y] = ode23t(odefun,tspan,y0), where tspan = [t0 tf], integrates the system of differential equations $y'=f(t,y)$ from t0 to tf with initial conditions y0. Each row in the solution array y corresponds to a value returned in column vector t.

All MATLAB® ODE solvers can solve systems of equations of the form $y'=f(t,y)$, or problems that involve a mass matrix, $M(t,y)y'=f(t,y)$. The solvers all use similar syntaxes. The ode23s solver only can solve problems with a mass matrix if the mass matrix is constant. ode15s and ode23t can solve problems with a mass matrix that is singular, known as differential-algebraic equations (DAEs). Specify the mass matrix using the Mass option of odeset.

[t,y] = ode23t(odefun,tspan,y0,options) also uses the integration settings defined by options, which is an argument created using the odeset function. For example, use the AbsTol and RelTol options to specify absolute and relative error tolerances, or the Mass option to provide a mass matrix.
[t,y,te,ye,ie] = ode23t(odefun,tspan,y0,options) additionally finds where functions of (t,y), called event functions, are zero. In the output, te is the time of the event, ye is the solution at the time of the event, and ie is the index of the triggered event.
For each event function, specify whether the integration is to terminate at a zero and whether the direction of the zero crossing matters. Do this by setting the 'Events'property to a function, such as myEventFcn or @myEventFcn, and creating a corresponding function: [value,isterminal,direction] = myEventFcn(t,y). For more information, see ODE Event Location.
sol = ode23t(___) returns a structure that you can use with deval to evaluate the solution at any point on the interval [t0 tf]. You can use any of the input argument combinations in previous syntaxes.

Examples

ODE With Single Solution Component

Simple ODEs that have a single solution component can be specified as an anonymous function in the call to the solver. The anonymous function must accept two inputs (t,y) even if one of the inputs is not used.
Solve the ODE

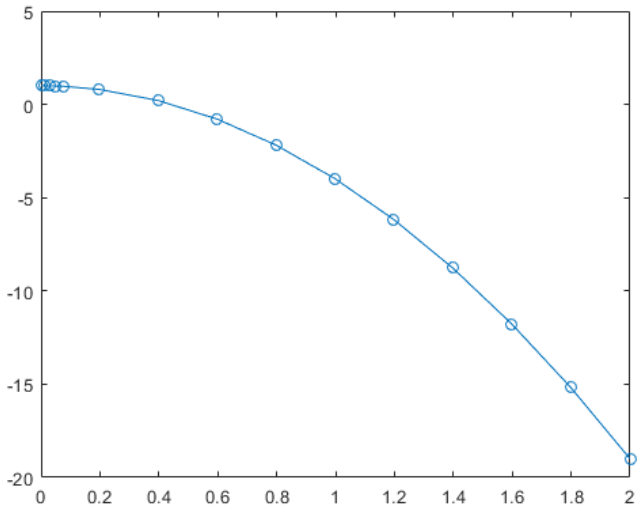
$$y' = -10t.$$

Use a time interval of [0,2] and the initial condition y0 = 1.

```
tspan = [0 2];
y0 = 1;
[t,y] = ode23t(@(t,y) -10*t, tspan, y0);
```

Plot the solution.

```
plot(t,y,'-o')
```



Fully Implicit Solvers

ode15i	Solve fully implicit differential equations — variable order method
decic	Compute consistent initial conditions for ode15i

ode15i
Solve fully implicit differential equations — variable order method

Syntax
[t,y] = ode15i(odefun,tspan,y0,yp0)
[t,y] = ode15i(odefun,tspan,y0,yp0,options)
[t,y,te,ye,ie] = ode15i(odefun,tspan,y0,yp0,options)
sol = ode15i(___)
Description

[t,y] = ode15i(odefun,tspan,y0,yp0), where tspan = [t0 tf], integrates the system of differential equations $f(t,y,y')=0$ from t0 to tf with initial conditions y0 and yp0. Each row in the solution array y corresponds to a value returned in column vector t.

[t,y] = ode15i(odefun,tspan,y0,yp0,options) also uses the integration settings defined by options, which is an argument created using the odeset function. For example, use the AbsTol and RelTol options to specify absolute and relative error tolerances, or the Jacobian option to provide the Jacobian matrix.
[t,y,te,ye,ie] = ode15i(odefun,tspan,y0,yp0,options) additionally finds where functions of (t,y,y'), called event functions, are zero. In the output, te is the time of the event, ye is the solution at the time of the event, and ie is the index of the triggered event.
For each event function, specify whether the integration is to terminate at a zero and whether the direction of the zero crossing matters. Do this by setting the 'Events'property to a function, such as myEventFcn or @myEventFcn, and creating a corresponding function: [value,isterminal,direction] = myEventFcn(t,y,yp). For more information, see ODE Event Location.

sol = ode15i(___) returns a structure that you can use with deval to evaluate the solution at any point on the interval [t0 tf]. You can use any of the input argument combinations in previous syntaxes.

Examples

collapse all

Solve Weissinger Implicit ODE

Try this Example

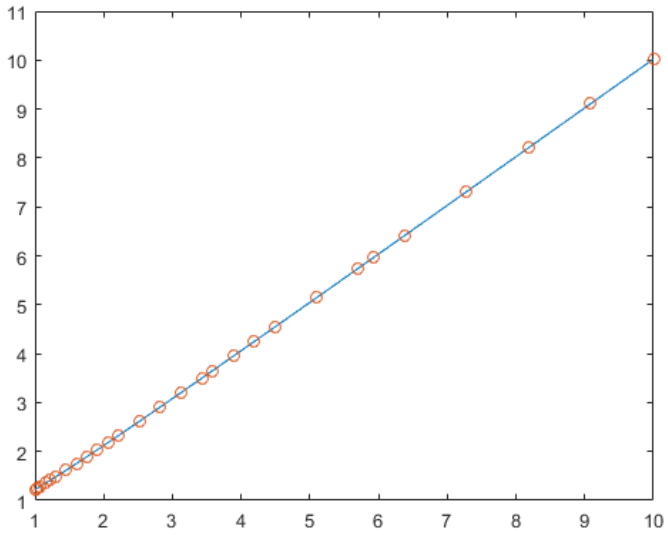
Use decic to compute consistent initial conditions for the Weissinger implicit ODE. decic holds fixed the initial value for *y(t0)* and computes a consistent initial value for *y'(t0)*.

The weissinger function evaluates the residual of the implicit ODE.

```
t0 = 1;
y0 = sqrt(3/2);
yp0 = 0;
[y0,yp0] = decic(@weissinger,t0,y0,1,yp0,0);
```

Use the result returned by decic with ode15i to solve the ODE. Plot the numerical solution, y, against the analytical solution, ytrue.

```
[t,y] = ode15i(@weissinger,[1 10],y0,yp0);
ytrue = sqrt(t.^2 + 0.5);
plot(t,y,t,ytrue,'o')
```



Get/Set Options

odeget	Extract ODE option values
odeset	Create or modify options structure for ODE s
deval	Evaluate differential equation solution structure
odextend	Extend solution to ODE

Non Linear Algebraic Equations

Solve system of nonlinear equations

Nonlinear system solver
Solves a problem specified by
$F(x) = 0$
for x, where $F(x)$ is a function that returns a vector value.
x is a vector or a matrix; see Matrix Arguments.

Syntax

```
x = fsolve(fun,x0)
x = fsolve(fun,x0,options)
x = fsolve(problem)
[x,fval] = fsolve(___)
[x,fval,exitflag,output] = fsolve(___)
[x,fval,exitflag,output,jacobian] = fsolve(___)
```

Description

x = fsolve(fun,x0) starts at x0 and tries to solve the equations fun(x) = **0**, an array of zeros.

x = fsolve(fun,x0,options) solves the equations with the optimization options specified in options. Use optimoptions to set these options.

x = fsolve(problem) solves problem, where problem is a structure described in Input Arguments. Create the problem structure by exporting a problem from Optimization app, as described in Exporting Your Work.

[x,fval] = fsolve(___), for any syntax, returns the value of the objective function fun at the solution x.
example
[x,fval,exitflag,output] = fsolve(___) additionally returns a value exitflag that describes the exit condition of fsolve, and a structure output with information about the optimization process.
[x,fval,exitflag,output,jacobian] = fsolve(___) returns the Jacobian of fun at the solution x.

Examples

Solution of 2-D Nonlinear System

This example shows how to solve two nonlinear equations in two variables. The equations are

$$e^{-e^{-(x_1+x_2)}} = x_2\left(1+x_1^2\right)$$
$$x_1\cos(x_2) + x_2\sin(x_1) = \tfrac{1}{2}.$$

Convert the equations to the form $F(x) = \mathbf{0}$.

$$e^{-e^{-(x_1+x_2)}} - x_2\left(1+x_1^2\right) = 0$$
$$x_1\cos(x_2) + x_2\sin(x_1) - \tfrac{1}{2} = 0.$$

Write a function that computes the left-hand side of these two equations.

```
function F = root2d(x)
```

```
F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
```

Save this code as a file named root2d.m on your MATLAB® path.
Solve the system of equations starting at the point [0,0].

```
fun = @root2d;
x0 = [0,0];
x = fsolve(fun,x0)
```

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

x =

0.3532 0.6061

Solution with Nondefault Options

Examine the solution process for a nonlinear system.
Set options to have no display and a plot function that displays the first-order optimality, which should converge to 0 as the algorithm iterates.

```
options = optimoptions('fsolve','Display','none','PlotFcn',@optimplotfirstorderopt);
```

The equations in the nonlinear system are

$$e^{-e^{-(x_1+x_2)}} = x_2\left(1+x_1^2\right)$$
$$x_1\cos(x_2) + x_2\sin(x_1) = \tfrac{1}{2}.$$

Convert the equations to the form $F(x) = \mathbf{0}$.

$$e^{-e^{-(x_1+x_2)}} - x_2\left(1+x_1^2\right) = 0$$
$$x_1\cos(x_2) + x_2\sin(x_1) - \tfrac{1}{2} = 0.$$

Write a function that computes the left-hand side of these two equations.

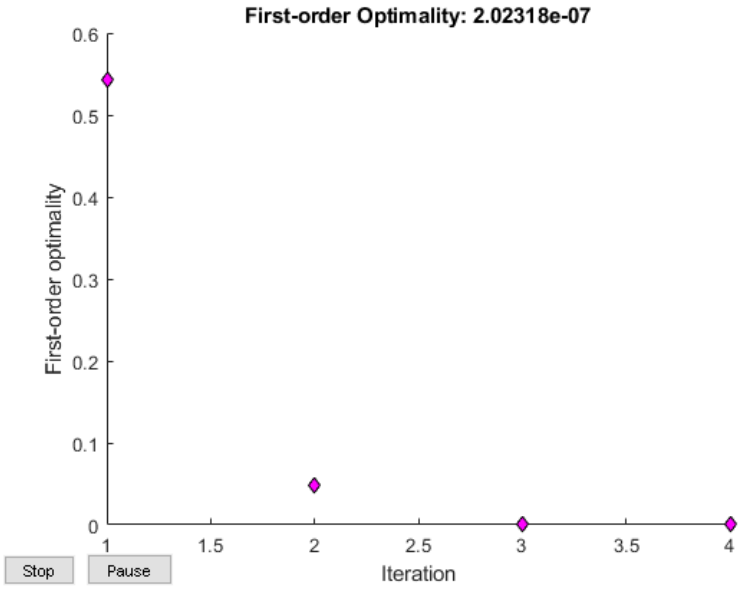
```
function F = root2d(x)

F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
```

Save this code as a file named root2d.m on your MATLAB® path.
Solve the nonlinear system starting from the point [0,0] and observe the solution process.

```
fun = @root2d;
x0 = [0,0];
x = fsolve(fun,x0,options)
```

x =

0.3532 0.6061



Solve a Problem Structure

Create a problem structure for fsolve and solve the problem.
Solve the same problem as in Solution with Nondefault Options, but formulate the problem using a problem structure.
Set options for the problem to have no display and a plot function that displays the first-order optimality, which should converge to 0 as the algorithm iterates.
problem.options = optimoptions('fsolve','Display','none','PlotFcn',@optimplotfirstorderopt);
The equations in the nonlinear system are

$$e^{-e^{-(x_1+x_2)}} = x_2\left(1+x_1^2\right)$$
$$x_1\cos\left(x_2\right) + x_2\sin\left(x_1\right) = \tfrac{1}{2}.$$

Convert the equations to the form $F(x) = \mathbf{0}$.

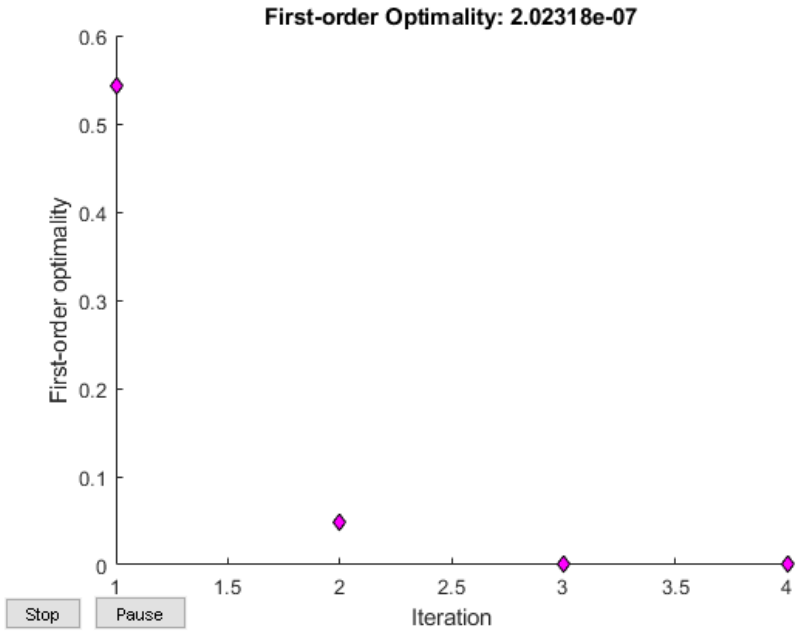
$$e^{-e^{-(x_1+x_2)}} - x_2\left(1+x_1^2\right) = 0$$
$$x_1\cos\left(x_2\right) + x_2\sin\left(x_1\right) - \tfrac{1}{2} = 0.$$

Write a function that computes the left-hand side of these two equations.
function F = root2d(x)

F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;

Save this code as a file named root2d.m on your MATLAB® path.
Create the remaining fields in the problem structure.
problem.objective = @root2d;
problem.x0 = [0,0];
problem.solver = 'fsolve';
Solve the problem.
x = fsolve(problem)

```
x =
    0.3532    0.6061
```



Root of nonlinear function

Syntax

```
x = fzero(fun,x0)
x = fzero(fun,x0,options)
x = fzero(problem)
[x,fval,exitflag,output] = fzero(___)
```

Description

x = fzero(fun,x0) tries to find a point x where fun(x) = 0. This solution is where fun(x) changes sign—fzero cannot find a root of a function such as x^2.

x = fzero(fun,x0,options) uses options to modify the solution process.

x = fzero(problem) solves a root-finding problem specified by problem.

[x,fval,exitflag,output] = fzero(___) returns fun(x) in the fval output, exitflag encoding the reason fzero stopped, and an output structure containing information on the solution process.

Examples

Root Starting From One Point

Calculate π by finding the zero of the sine function near 3.

```
fun = @sin; % function
x0 = 3; % initial point
x = fzero(fun,x0)
x = 3.1416
```

Root Starting From an Interval

Find the zero of cosine between 1 and 2.
```
fun = @cos; % function
x0 = [1 2]; % initial interval
x = fzero(fun,x0)
x = 1.5708
```
Note that $\cos(1)$ and $\cos(2)$ differ in sign.

Root of a Function Defined by a File

Find a zero of the function $f(x) = x^3 - 2x - 5$.

First, write a file called f.m.
```
function y = f(x)
y = x.^3 - 2*x - 5;
```
Save f.m on your MATLAB® path.

Find the zero of $f(x)$ near 2.
```
fun = @f; % function
x0 = 2; % initial point
z = fzero(fun,x0)
z =
   2.0946
```
Since f(x) is a polynomial, you can find the same real zero, and a complex conjugate pair of zeros, using the roots command.
```
roots([1 0 -2 -5])
ans =
  2.0946
 -1.0473 + 1.1359i
 -1.0473 - 1.1359i
```

Advanced Topics

The topics covered in the preceding sections of this chapter are far from exhaustivein what you can do readily with MATLAB . These topics have been selected carefullyto introduce you to various applications that you are likely to use frequently in yourwork. Once you gain a little bit of experience and some confidence, you can exploremost of the advanced features of the functions introduced as well as several functionsfor more complex applications on your own, with the help of on-line documentation.MATLAB provides some new functions for solving two-point boundary valueproblems, simple partial differential equations, and nonlinear function minimizationproblems. In particular, we mention the following functions.

dde23 added in MATLAB 7, this function solves delay differential equations (DAEs) with constant delays

ode15isolves implicit ODEs and DAEs of index 1 with the helper function ode i c for evaluating consistent initial conditions.

bvp4c solves two-point boundary value problems (BVP) defined by a set of ODEsof the form y' = f (x , y), and its boundary conditions y (a) and y (b) over theinterval [a, b] . The user has to write two functions-one that specifies theequations and the other that specifies the boundary

conditions. Users can setseveral options for initiating the solutions. To see an example, try executingthe built-in example t wobvp and learn how to program your BVP by following this example (to see the functions required for twobvp , type typetwoode . mand t ypetwobc . m).
pdepe solves simple parabolic and elliptic partial differential equations (PDEs) of a single dependent variable. This is a rather restricted utility function.However, for those who need to solve PDEs frequently, there is the PartialDifferential Equation Toolbox.

Part B (5x8=40 Marks)

Possible Questions

1. Explain statistical tool in mat lab with examples.
2. Illustrate the numerical integration with examples
3. Describe about the double integration by using dblquad function with example
4. How to solve the ordinary differential equations with example.
5. Write mat lab commands for solving a first order linear ODE with example.
6. Write a procedure for solve the second order linear differential equation with examples.
7. Describe about the ODE suite.
8. Solve the transcendental equation $\sin x = e^x - 5$ using mat lab
9. Explain about finding roots of polynomial equation in mat lab.
10. Describe about solving Non- linear algebraic equation with example.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari (Po),
Coimbatore –641 021

Subject: MATLAB programming **Subject Code: 15MMU504**

Class : III - B.Sc. Mathematics **Semester : V**

Unit V

Applications

Part A (20x1=20 Marks)
(Question Nos. 1 to 20 Online Examinations)

Possible Questions

Question	Choice 1	Choice 2	Choice 3	Choice 4	Answer
___ command is used to determine the largest value in the data set	sum	max	cumsum	maxm	max
Which command is used to find the standard deviation based on sample.	stade	SD	std	sdev	std
If A = [8 3 0] then median(A) =	1.5	3	8	0	3
Numerical evaluation of the integral ∫ f(x) dx is called ___	quadrature	integration	function	differentiation	quadrature
Which is the function for double intergration	int2	dblint	quad	dblquad	dblquad
_____ function is used as nonstiff solver based on 3rd order Runge-Kutta Method	ode15s	ode23	ode45	ode113	ode23
_____ function is used as stiff solver based on variable order	ode15s	ode23	ode45	ode113	ode113

Which function that get various options parameters?	odeset	odeget	get	odegt	odeget
_____ function that is specified in options for 2-D plots.	odephas2	odeplot	odeset	odeget	odephas2
Which function is stiff solver and DAE solver based on a variable order method	ode23tb	ode15tb	ode15s	ode23s	ode15s
ode23tb is stiff solver based on _____	Lower order method	Adams method	Runge-Kutta Method	Trapezoidal rule	Lower order method
_____ this function solve delay differential equations	dde113	dde15	dde23	dde45	dde23
To solve two-point boundary value problem by using _____ function	bvp4c	bvp2c	bcp23c	bvp5c	bvp4c
____ this function is used to solves implicit ODEs	ode113i	ode45i	ode15i	ode23i	ode15i
Which function is used to solve simple parabolic PDEs ?	pdepe	pdep	pde23	pdeep	pdepe
ode23s is stiff solver based on _____	Adams method	Runge-Kutta Method	Trapezoidal rule	numerical differentiation formula	numerical differentiation formula
ode45 is nonstiff solver based on _____	Runge-Kutta Method	Trapezoidal rule	Adams method	numerical differentiation formula	Runge-Kutta Method
The syntax for ODE solver is	[time,sol] = ode23('function';tspan; x_0)	[time,solution] = ode23('function', tspan)	[time,solution] = ode23('function', tspan, x_0)	[time,solution] = ode45('function', tspan, x_0)	[time,solution] = ode23('function', tspan, x_0)
ode23t is stiff solver based on _____	Adams method	Runge-Kutta Method	Trapezoidal rule	numerical differentiation formula	Trapezoidal rule

ode113 is stiff solver based on _____	Adams method	Runge-Kutta Method	Trapezoidal rule	numerical differentiation formula	Adams method
odephasa2 function that is specified in options for____phase plots.	1-D	binary	2-D	3-D	2-D
Which of the following solver based on fifth order Runge-Kutta method ?	ode15s	ode23	ode45	ode113	ode45
Which function that set various options for the solver?	odeset	odeget	get	odegt	odeset

Reg. No ----------

(15MMU504)

KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE - 21

Department of Mathematics

Fifth Semester

First Internal Test – July - 2017

MATLAB Programming

Date : 20.07.2017 (FN) **Time: 2 Hours**

Class : III - B.Sc. Mathematics **Maximum: 50 Marks**

PART-A (20 x 1 =20 Marks)

Answer All the Questions

1. Which symbol precedes all comments in Mat lab?
   a) " b) { c) >> d) [[
2. The fundamental data type of Matlab is _____
   a) matrix b) array c) list d) string
3. Which of the following is not pre-defined variable in Mat lab?
   a) pi b) inf c) i d) gravity
4. Which of the following command is used to clear all data and variables in memory?
   a) clc b) clear c) delete d) deallocate
5. ____ character in Mat lab are represented in their value in memory.
   a) decimal b) hex c) ASCII d) string
6. In Mat lab, this keyword immediately moves to the next iteration of the loop
   a) update b) goto c) move d) continue
7. To add a comment to the m file, the Mat lab command is _____
   a) % b) @ c) & d) (' ')
8. The clc command is used to ____
   a) erase everything in m file b) clear the command window
   c) clean the desktop d) save the existing mfile
9. The basic building block of Mat lab is the _____
   a) array b) string c) matrix d) list
10. To find the dimension of an existing matrix in Mat lab with _____ command
    a) size b) length c) dim d) eye
11. To suppresses the screen output we use _____ at the end of the command
    a) dot b) camma c) colon d) semicolon
12. Mat-files can be loaded into MATLAB with the_____ command
    a) load b) lod c) list d) update
13. Typing _____ at the Mat lab prompt to print the content of the figure window
    a) eye b) print c) copy d) prt
14. mkdir command is used to create a _____
    a) script file b) directory c) function file d) m file
15. If we give square brackets with no elements between them then it creates
    a) identity matrix b) square matrix c) diagonal matrix d) null matrix
16. The three consecutive periods using in matrices are also called as _____
    a) ellipse b) hyperbola c) parabola d) circle
17. _____ in MATLAB refers to the element aij of matrix A
    a) Aij b) A{ij} c) A (i , j) d) A (j, i)
18. The transpose of matrix A is obtained by typing
    a) A" b) (A) c) A' d) A;
19. _____ is the same as u=a : (b-a) / (n-1) : b
    a) u = log(a,b,n) b) u=line (a , b , n)
    c) u=linspace (a , b , n) d) u = logspace(a,b,n)
20. _____ command is round the output towards 0
    a) ceil b) fix c) floor d) round

PART-B (3 x 10 =30 Marks)

Answer All the Questions

21. a) Discuss about the Mat lab Windows

(OR)

b) List out the mat lab general commands with explanation.

22. a) Explain in detail about Mat lab‘s features of input and output.

(OR)

b) Describe about elementary math functions.

23. a) Explain in detail about Mat lab file types.

(OR)

b) Explain about Matrix Manipulation with example.

Reg. No ----------

(15MMU504)

KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE - 21

Department of Mathematics

Fifth Semester

II – Internal Test – August - 2017

MATLAB Programming

Date : 10.08.2017 (FN) **Time: 2 Hours**

Class : III - B.Sc. Mathematics **Maximum: 50 Marks**

PART-A (20 x 1 =20 Marks)

Answer All the Questions

1. To converts characters to their ASCII numeric values by using ____
a) strcat b) abs c) char d) ischar
2. To converts any uppercase letters in the string to lowercase by using ___
a) lower b) strcat c) ischar d) char
3. _____ executes the string as a command
a) strcat b) abs c) eval d) lower
4. The syntax for creating an inline function is particularly simple
a) F = inline ('function') b) F = inline ('function formula')
c) F = in('function formula') d) F = inline ('formula')
5. An _____is created by the command
f = @(inputlist) mathematical expression
a) vector function b) argument function
c) inline functions d) anonymous function
6. To evaluates f (x) at x = 5 by giving
a) f(x)=5 b) fx(5) c) f(5) d) x(5)
7. ____loop is conditionally execute statements
a) else b) If c) If else d) for
8. _____ command is terminate scope of control statements.
a) If b) end c) for d) else
9. To produce stunning surface plots in 3-D by using ____ command
a) ezsurf b) ezpolar c) ezplot d) ezcontour
10.In Matlab command _____ does not need brackets
a) vector b) string c) scalar d) matrix
11. _____is executed by typing name of the file on the commamd line.
a) script file b) function file c) date file d) figure file
12. A ____ is most versatile data object in Matlab
a) matrix b) array c) string d) cell
13. In ___ control structure a group of statements are executed only if the condition is true.
a) if b) if else c) if elseif d) nested if
14. ____ command terminate the execution of for or while loop
a) switch b) break c) continue d) error
15. A script file is an _____ with a set of vaild MATLAB command
a) Mex - file b) Mat - file c) M -file d) fig - file
16. Which is these is not an aspect of a for or while loop?
a) update b) initialization c) runner d) condition
17. In ___ control structure two group of statements are executed only if one is true and other is false condition.
a) if b) if else c) if else if d) nested if
18. _____ command display the message and abort function
a) switch b) break c) continue d) error
19. ____ command catch the error generated by MATLAB
a) try-catch b) break c) continue d) error
20. To print a new line in a fprintf statement, you must use the following escape character
a) \t b) \n c) \nxt d) \n1

PART-B (3 x 10 =30 Marks)

Answer All the Questions

21. a) Explain about plotting simple graph with example.

(OR)

b) Explain in detail about character strings with example.

22. a) Describe about script files with examples.

(OR)

b) List out the commands for interactive user input in script file or function file.

23. a) Explain the control flow statements with examples.

(OR)

b) Give a brief notes on structures with an example.

Reg. No ----------

(15MMU504)

KARPAGAM ACADEMY OF HIGHER EDUCATION

COIMBATORE-21

MODEL EXAMINATION- SEP 2017

Fifth Semester

Mathematics

MATLAB Programming

Date : 14.09.2017 (FN) **Time: 3 Hours**

Class : III - B.Sc. Mathematics **Maximum: 60 Marks**

PART - A (20 x 1 =20 Marks)

Answer All the Questions:

1. _____ character in Matlab are represented in their value in memory.
a) decimal b) hex c) ASCII d) string

2. In Matlab, this keyword immediately moves to the next iterartion of the loop
a) update b) goto c) into d) continue

3. To suppresses the screen output we use _____ at the end of the command
a) dot b) camma c) colon d) semicolon

4. Mat-files can be loaded into MATLAB with the_____command
a) load b) lod c) list d) update

5. The 0- 1 vector created by you is converted into a logical array with the command _____
a) array b) logical c) ones d) zeros

6. All the elements of matrix A can be strung into a single-column vector b by the comman
a) b = A(:) b) b = A(\) c) b = A[ ] d) b = A(")

7. The command eye(2) produce a 2x2 _____ matrix
a) zero b) square c) identity d) null

8. To produce a = [0 0.5 1 1.5 2 2.5 3] by giving a command as ___
a) a = 0 … 3 b) a = 0 : 0.5 : 3 c) a = 0.5 : 3 d) a = 0 : 3

9. If I want to save a formatted string to memory, but don't want to print it out, which command should I use ?
a) fprintf b) sprintf c) disp d) echo

10. When used in the fprintf command ,the %g is used as the ____
a) single character display b) fixed point display
c) string natation display d) default number display

11. When used in the fprintf command ,the \n is used to
a) add a space between any two character b) add a line space
c) place a number into comment d) clear the comment

12. The first function in the file is called _____ function
a) Inline b) private c) primary d) nested

13. To solve the QR factorization by using the command
a) {q,r} = [A] b) {Q,R} = [A] c) [Q,R] = qr(A) d) [Q,R] = A

14. ____ bulit in function is used to solve cholesky factorization
a) ch b) chol c) chl d) chlf

15. To determine the eigen value for matrix A
a) A = eigen (A) b) eig(A) c) eign(A) d) Eig(A,:)

16. spline command is used 1-D interpolation that uses_____.
a) square spline b) cubic spline fit
c) least square fit d) straight-line fit

17. Which command is used to find the standard deviation based on sample?
a) stade b) SD c) std d) sdev

18. _____ function that is specified in options for 2-D plots.
a) odephas2 b) odeplot c) odeset d) odeget

19. To solve two-point boundary value problem by using _____ function
a) bvp4c b) bvp2c c) bcp23c d) bvp5c

20. Which function is used to solve simple parabolic PDEs ?
a) pdepe b) pdep c) pde23 d) pdeep

PART-B (5 x 8 = 40 Marks)

Answer All the Questions:

21. a) Explain about Matlab environment with diagrammatic representation.

(OR)

b) Write a short note on (i) Matlab Desktop
(ii) Figure window
(iii)Editor window

22. a) Write a short note on
(i) Appending row or column
(ii) Deleting a row or column
(iii) Utility matrices

(OR)

b) Explain about plotting simple graph with example.

23. a) Define global variables. Give example for solving 1^{st} order ODE by using global variables.

(OR)

b) Explain about the Switch case and Break with examples.

24. a) Describe about matrix operation in linear algebra with example.

(OR)

b) Explain about matrix factorization with example.

25. a) Explain statistical tool in matlab with examples.

(OR)

b) Write a procedure for solve the first order linear differential equation with examples