

Appendix D

Introduction to MATLAB¹

D.1 What Is MATLAB?

MATLAB is a technical computing environment developed by The MathWorks, Inc. for computation and data visualization. It is both an interactive system and a programming language, whose basic data element is an array: scalar, vector, matrix, or multi-dimensional array. Besides basic array operations, it offers programming features similar to those of other computing languages (e.g., functions, control flow, etc.).

In this appendix, we provide a brief summary of MATLAB to help the reader understand the algorithms in the text. We do not claim that this introduction is complete, and we urge the reader to learn more about MATLAB from other sources. The documentation that comes with MATLAB is excellent, and the reader should find the tutorials contained in there to be helpful. For a comprehensive overview of MATLAB, we also recommend Hanselman and Littlefield [1998, 2001]. A new book that introduces numerical computing with MATLAB is by Moler [2004]. This book is available in print version (see [references](#)); an electronic version can be downloaded from

<http://www.mathworks.com/moler>

If the reader needs to understand more about the graphics and GUI capabilities in MATLAB, Marchand and Holland [2003] is the reference to use.

MATLAB will execute on Windows, UNIX/Linux, and Macintosh systems. Here we focus on the Windows version, but most of the information applies to all systems. The main MATLAB software package contains many functions for analyzing data. There are also specialty toolboxes that extend the capabilities of MATLAB. These are available from The MathWorks and

¹Much of this appendix was taken and updated from the corresponding appendix in *Computational Statistics Handbook with MATLAB* [2002].

third party vendors for purchase. Some toolboxes are also on the internet and may be downloaded at no charge. See [Appendix B](#) for a partial list.

We assume that readers know how to start MATLAB for their particular platform. When MATLAB is started, you will have a command window with a prompt where you can enter commands. Other windows are available (help window, history window, etc.), but we do not cover those here.

D.2 Getting Help in MATLAB

One useful and important aspect of MATLAB is the **help** feature. There are many ways to get information about a MATLAB function. Not only does the **help** provide information about the function, but it also gives references for other related functions. We discuss below the various ways to get help in MATLAB.

- *Command Line:* Typing **help** and then the function name at the command line will, in most cases, tell you everything you need to know about the function. In this text, we do not write about all the capabilities or uses of a function. The reader is strongly encouraged to use command line **help** to find out more. As an example, typing **help plot** at the command line provides lots of useful information about the basic **plot** function. Note that the command line **help** works with the EDA Toolbox (and others) as well.
- *Help Menu:* The **help** files can also be accessed via the usual **Help** menu. This opens up a separate **help** window. Information can be obtained by clicking on links or searching the index.

D.3 File and Workspace Management

We can enter commands interactively at the command line or save them in an M-file. So, it is important to know some commands for file management. The commands shown in [Table D.1](#) can be used to list, view, and delete files.

Variables created in a session (and not deleted) live in the MATLAB *workspace*. You can recall the variable at any time by typing in the variable name with no punctuation at the end. Note that MATLAB is case sensitive, so **Temp**, **temp**, and **TEMP** represent different variables.

MATLAB remembers the commands that you enter in the command history. There is a separate command history window available via the **View** menu and certain desktop layouts. One can use this to re-execute old

TABLE D.1
File Management Commands

Command	Usage
dir, ls	Shows the files in the present directory.
delete filename	Deletes filename .
cd, pwd	Show the present directory.
cd dir, chdir	Changes the directory. There is a pop-up menu on the toolbar that allows the user to change directory.
type filename	Lists the contents of filename .
edit filename	Brings up filename in the editor.
which filename	Displays the path to filename . This can help determine whether a file is part of the standard MATLAB package.
what	Lists the .m files and .mat files that are in the current directory.

TABLE D.2
Commands for Workspace Management

Command	Usage
who	Lists all variables in the workspace.
whos	Lists all variables in the workspace along with the size in bytes, array dimensions, and object type.
clear	Removes all variables from the workspace.
clear x y	Removes variables x and y from the workspace.

commands. One way is to highlight the desired commands (hold the **ctrl** key while clicking), right click for a shortcut menu on the highlighted section, and select **Evaluate Selection**. While in the command window, the arrow keys can be used to recall and edit commands. The up-arrow and down-arrow keys scroll through the commands. The left and right arrows move through the present command. By using these keys, the user can recall commands and edit them using common editing keystrokes.

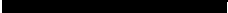
We can view the contents of the current workspace using the **Workspace Browser**. This is accessed through the **File** menu or the toolbar. All variables in the workspace are listed in the window. The variables can be viewed and edited in a spreadsheet-like window format by double-clicking on the variable name.

The commands contained in [Table D.2](#) help manage the workspace. It is important to be able to get data into MATLAB and to save it. We outline below some of the ways to get data in and out of MATLAB. These are not the only options for file I/O. For example, see **help** on **fprintf**, **fscanf**, and **textread** for more possibilities.

- Command Line: The **save** and **load** commands are the main way to perform file I/O in MATLAB. We give some examples of how to use the **save** command. The **load** command works similarly.

Command	Usage
save filename	Saves all variables in filename.mat .
save filename var1 var2	Saves only variables var1 var2 in filename.mat .
save filename var1 -ascii	Saves var1 in ASCII format in filename .

- File Menu: There are commands in the **File** menu for saving and loading the workspace.
- Import Wizard: There is a spreadsheet-like window for inputting data. To execute the wizard, type **uiimport** at the command line.



D.4 Punctuation in MATLAB

[Table D.3](#) contains some of the common punctuation characters in MATLAB, and how they are used.

TABLE D.3
List of Punctuation

Punctuation	Usage
%	A percent sign denotes a comment line. Information after the % is ignored.
,	When used to separate commands on a single line, a comma tells MATLAB to display the results of the preceding command. When used in concatenation (between brackets []), a comma or a blank space concatenates elements along a row. A comma also has other uses, including separating function arguments and array subscripts.
;	When used after a line of input or between commands on a single line, a semicolon tells MATLAB not to display the results of the preceding command. When used in concatenation (between brackets []), a semicolon begins a new row.
...	Three periods denote the continuation of a statement. Comment statements and variable names cannot be continued with this punctuation.
!	An exclamation tells MATLAB to execute the following as an operating system command.
:	The colon specifies a range of numbers. For example, 1:10 means the numbers 1 through 10. A colon in an array dimension accesses all elements in that dimension.
.	The period before an operator tells MATLAB to perform the corresponding operation on each element in the array.

D.5 Arithmetic Operators

Arithmetic operators ($*$, $/$, $+$, $-$, $^$) in MATLAB follow the conventions in linear algebra. If we are multiplying two matrices, **A** and **B**, they must be dimensionally correct. In other words, the number of columns of **A** must be equal to the number of rows of **B**. To multiply two matrices in this manner, we simply use **A*B**. It is important to remember that the default

interpretation of an operation is to perform the corresponding array operation.

MATLAB follows the usual order of operations. The precedence can be changed by using parentheses, as in other programming languages.

It is often useful to operate on an array element-by-element. For instance, we might want to square each element of an array. To accomplish this, we add a period before the operator. As an example, to square each element of array **A**, we use **A.*2**. These operators are summarized in Table D.4.

TABLE D.4
List of Element-by-Element Operators

Operator	Usage
.*	Multiply element-by-element.
./	Divide element-by-element.
.^	Raise elements to powers.

D.6 Data Constructs in MATLAB

Basic Data Constructs

We do not cover the object-oriented aspects of MATLAB here. Thus, we are concerned mostly with data that are floating point (type **double**) or strings (type **char**). The elements in the arrays will be of these two data types.

The fundamental data element in MATLAB is an array. Arrays can be:

- The 0×0 empty array created using **[]**.
- A 1×1 scalar array.
- A row vector, which is a $1 \times n$ array.
- A column vector, which is an $n \times 1$ array.
- A matrix with two dimensions, say $m \times n$ or $n \times n$.
- A multi-dimensional array, say $m \times \dots \times n$.

Arrays must always be dimensionally conformal and all elements must be of the same data type. In other words, a 2×3 matrix must have 3 elements (e.g., numbers) on each of its 2 rows. Table D.5 gives examples of how to access elements of arrays.

Building Arrays

In most cases, the statistician or engineer will need to import data into MATLAB using **load** or some other method described previously. However, sometimes we might need to type in simple arrays for testing code or entering parameters, etc. Here we cover some of the ways to build small arrays. Note that this can also be used to combine separate arrays into one large array.

Commas or spaces concatenate elements (which can be arrays) as columns. Thus, we get a row vector from the following

```
temp = [1, 4, 5];
```

or we can concatenate two column vectors **a** and **b** into one matrix, as follows

```
temp = [a b];
```

The semi-colon tells MATLAB to concatenate elements as rows. So, we would get a column vector from this command:

```
temp = [1; 4; 5];
```

We note that when concatenating arrays, the sizes of each array element must be conformal. The ideas presented here also apply to cell arrays, discussed below.

Before we continue with cell arrays, we cover some of the other useful functions in MATLAB for building arrays. These are summarized here.

Function	Usage
zeros, ones	These build arrays containing all 0's or all 1's, respectively.
rand, randn	These build arrays containing uniform (0,1) random variables or standard normal random variables, respectively.
eye	This creates an identity matrix.

Cell Arrays

Cell arrays and structures allow for more flexibility. Cell arrays can have elements that contain any data type (even other cell arrays), and they can be of different sizes. The cell array has an overall structure that is similar to the basic data arrays. For instance, the cells are arranged in dimensions (rows, columns, etc.). If we have a 2×3 cell array, then each of its 2 rows has to have 3 cells. However, the *content* of the cells can be different sizes and can contain

different types of data. One cell might contain **char** data, another **double**, and some can be empty. Mathematical operations are not defined on cell arrays.

TABLE D.5
Examples of Accessing Elements of Arrays

Notation	Usage
a(i)	Denotes the <i>i</i> -th element (cell) of a row or column vector array (cell array).
A(:,i)	Accesses the <i>i</i> -th column of a matrix or cell array. In this case, the colon in the row dimension tells MATLAB to access all rows.
A(i,:)	Accesses the <i>i</i> -th row of a matrix or cell array. The colon tells MATLAB to gather all of the columns.
A(1,3,4)	This accesses the element in the first row, third column on the fourth entry of dimension 3 (sometimes called the page).

In Table D.5, we show some of the common ways to access elements of arrays, which can be cell arrays or basic arrays. With cell arrays, this accesses the cell element, but not the contents of the cells. Curly braces, **{ }**, are used to get to the elements inside the cell. For example, **A{1,1}** would give us the contents of the cell (type **double** or **char**). Whereas, **A(1,1)** is the cell itself and has data type **cell**. The two notations can be combined to access part of the contents of a cell. To get the first two elements of the contents of **A{1,1}**, assuming it contains a vector, we can use

A{1,1} (1:2).

Cell arrays are very useful when using strings in plotting functions such as **text**.

Structures

Structures are similar to cell arrays in that they allow one to combine collections of dissimilar data into a single variable. Individual structure elements are addressed by names called *fields*. We use the *dot notation* to access the fields. Each element of a structure is called a *record*.

For example, suppose we had a structure called **data** that had fields called **name**, **dob**, **test**. Then we could obtain the information in the tenth record using

```
data(10).name  
data(10).dob  
data(10).test
```

Here **name** and **dob** are vectors of type **char**, and **test** is a numeric. Note that fields may also be structures with their own fields. We may access the elements of the fields using the element accessing syntax discussed earlier.

D.7 Script Files and Functions

MATLAB programs are saved in M-files. These are text files that contain MATLAB commands, and they are saved with the **.m** extension. Any text editor can be used to create them, but the one that comes with MATLAB is recommended. This editor can be activated using the **File** menu or the toolbar.

When script files are executed, the commands are implemented just as if you typed them in interactively. The commands have access to the workspace and any variables created by the script file are in the workspace when the script finishes executing. To execute a script file, simply type the name of the file at the command line or use the option in the **File** menu.

Script files and functions both have the same **.m** extension. However, a function has a special syntax for the first line. In the general case, this syntax is

```
function [out1,...,outM] = func_name(in1,...,inN)
```

A function does not have to be written with input or output arguments. Whether you have these or not depends on the application and the purpose of the function. The function corresponding to the above syntax would be saved in a file called **func_name.m**. These functions are used in the same way any other MATLAB function is used.

It is important to keep in mind that functions in MATLAB are similar to those in other programming languages. The function has its own workspace. So, communicating information between the function workspace and the main workspace is done via input and output variables.

It is always a good idea to put several comment lines at the beginning of your function. These are returned by the **help** command.

D.8 Control Flow

Most computer languages provide features that allow one to control the flow of execution depending on certain conditions. MATLAB has similar constructs:

- **for** loops
- **while** loops
- **if-else** statements
- **switch** statement

These should be used sparingly. In most cases, it is more efficient in MATLAB to operate on an entire array rather than looping through it.

for Loop

The basic syntax for a **for** loop is

```
for i = array
    commands
end
```

Each time through the loop, the loop variable **i** assumes the next value in **array**. The colon notation is usually used to generate a sequence of numbers that **i** will take on. For example,

```
for i = 1:10
```

The commands between the **for** and the **end** statements are executed once for every value in the array. Several **for** loops can be nested, where each loop is closed by **end**.

while Loop

A **while** loop executes an indefinite number of times. The general syntax is:

```
while expression
    commands
end
```

The commands between the **while** and the **end** are executed as long as **expression** is true. Note that in MATLAB a scalar that is nonzero evaluates to true. Usually a scalar entry is used in the **expression**, but an array can

be used also. In the case of arrays, all elements of the resulting array must be true for the commands to execute.

if-else Statements

Sometimes, commands must be executed based on a relational test. The **if-else** statement is suitable here. The basic syntax is

```
if expression
    commands
elseif expression
    commands
else
    commands
end
```

Only one **end** is required at the end of the sequence of **if**, **elseif** and **else** statements. Commands are executed only if the corresponding **expression** is true.

switch Statement

The **switch** statement is useful if one needs a lot of **if**, **elseif** statements to execute the program. This construct is very similar to that in the C language. The basic syntax is:

```
switch expression
case value1
    commands execute if expression is value1
case value2
    commands execute if expression is value2
...
otherwise
    commands
end
```

The **expression** must be either a scalar or a character string.

D.9 Simple Plotting

For more information on some of the plotting capabilities of MATLAB, the reader is referred to the MATLAB documentation *Using MATLAB Graphics and Graphics and GUIs with MATLAB* [Marchand and Holland, 2003]. In this appendix, we briefly describe some of the basic uses of **plot** for plotting 2-D

graphics and **plot3** for plotting 3-D graphics. The reader is strongly urged to view the **help** file for more information and options for these functions.

When the function **plot** is called, it opens a **Figure** window (if one is not already there) scales the axes to fit the data, and plots the points. The default is to plot the points and connect them using straight lines. For example,

plot(x,y)

plots the values in vector **x** on the horizontal axis and the values in vector **y** on the vertical axis, connected by straight lines. These vectors must be the same size or you will get an error.

Any number of pairs can be used as arguments to **plot**. For instance, the following command plots two curves,

plot(x,y1,x,y2)

on the same axes. If only one argument is supplied to **plot**, then MATLAB plots the vector versus the index of its values.

The default is a solid line, but MATLAB allows other choices. These are given in Table D.6.

TABLE D.6
Line Styles for Plots

Notation	Line Type
-	Solid Line
:	Dotted Line
-.	Dash-dot Line
--	Dashed Line

If several lines are plotted on one set of axes, then MATLAB plots them in different colors. The predefined colors are listed in Table D.7.

TABLE D.7
Line Colors for Plots

Notation	Color
b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
k	black
w	white

Plotting symbols (e.g., *****, **x**, **o**, etc.) can be used for the points. Since the list of plotting symbols is rather long, we refer the reader to the online **help** for **plot** for more information. To plot a curve where both points and a connected curve are displayed, use

```
plot(x, y, x, y, 'b*')
```

or

```
plot(x,y,'b*-')
```

This command first plots the points in **x** and **y**, connecting them with straight lines. It then plots the points in **x** and **y** using the symbol ***** and the color blue. See the **help** on **graph2d** for more 2-D plots.

The **plot3** function works the same as **plot**, except that it takes three vectors for plotting:

```
plot3(x, y, z)
```

All of the line styles, colors, and plotting symbols apply to **plot3**. Other forms of 3-D plotting (e.g., **surf** and **mesh**) are available. See the **help** on **graph3d** for more capabilities. Titles and axes labels can be created for all plots using **title**, **xlabel**, **ylabel**, and **zlabel**.

Before we finish this discussion on simple plotting techniques in MATLAB, we present a way to put several axes or plots in one **figure** window. This is through the use of the **subplot** function. This creates an $m \times n$ matrix of plots (or axes) in the current **figure** window. We provide an example below, where we show how to create two plots side-by-side.

```
% Create the left-most plot.  
subplot(1,2,1)  
plot(x,y)  
% Create the right-most plot  
subplot(1,2,2)  
plot(x,z)
```

The first two arguments to **subplot** tell MATLAB about the layout of the plots within the **figure** window. The third argument tells MATLAB which plot to work with. The plots are numbered from top to bottom and left to right. The most recent plot that was created or worked on is the one affected by any subsequent plotting commands. To access a previous plot, simply use the **subplot** function again with the proper value for the third argument. You can think of the **subplot** function as a *pointer* that tells MATLAB what set of axes to work with.

Through the use of MATLAB's low-level Handle Graphics functions, the data analyst has complete control over graphical output. We do not present any of that here, because we make limited use of these capabilities. However, we urge the reader to look at the online **help** for **propedit**. This graphical user interface allows the user to change many aspects or properties of the plots without resorting to Handle Graphics.

D.10 Where to get MATLAB Information

For MATLAB product information, please contact:

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA, 01760-2098 USA
Tel: 508-647-7000
Fax: 508-647-7101
E-mail: info@mathworks.com
Web: www.mathworks.com

There are two useful resources that describe new products, programming tips, algorithm development, upcoming events, etc. One is the monthly electronic newsletter called the *MATLAB Digest*. Another is called *MATLAB News & Notes*, published quarterly. You can subscribe to both of these at www.mathworks.com or send an email request to

subscribe@mathworks.com

Back issues of these documents are available on-line.

The MathWorks has an educational area that contains many resources for professors and students. You can find contributed course materials, product recommendations by curriculum, tutorials on MATLAB, and more at:

www.mathworks.com/products/education/

The MathWorks also provides free live (and archived) webinars that demonstrate their products. See

www.mathworks.com/webinars

for a schedule and registration information.

There is an active news group to discuss MATLAB. Sign up at

comp.soft-sys.matlab

This is a good place to ask questions and share ideas. Another useful source for MATLAB users is MATLAB Central. This is on the main website at The MathWorks:

www.mathworks.com/matlabcentral/

It has an area for sharing MATLAB files, a web interface to the news group, and other features. It is always a good idea to check here first before writing your own functions, since what you need might be there already.