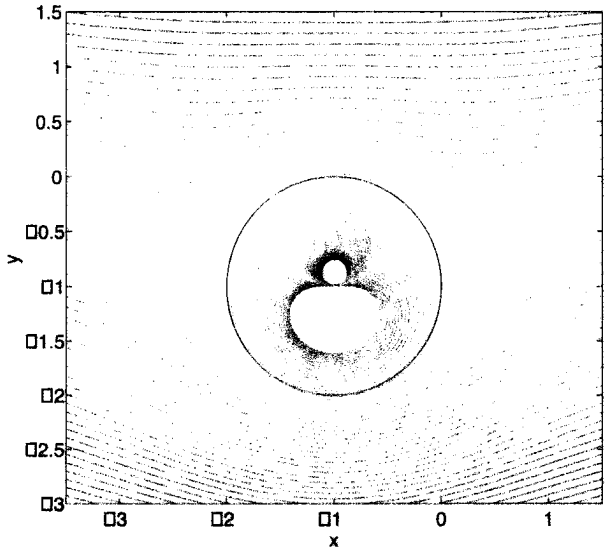
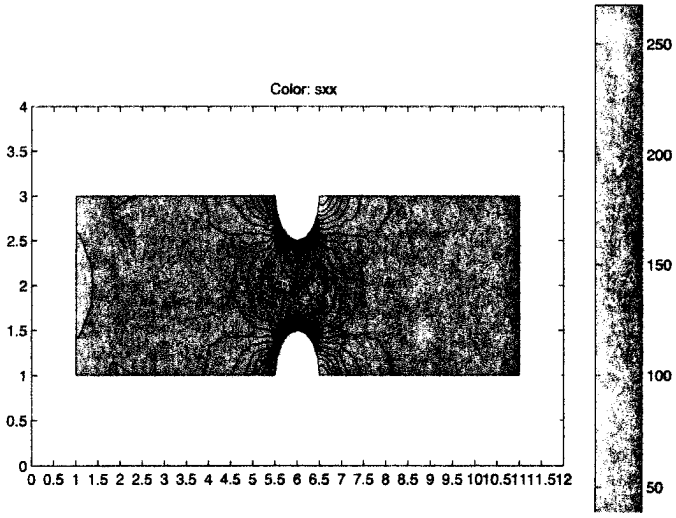
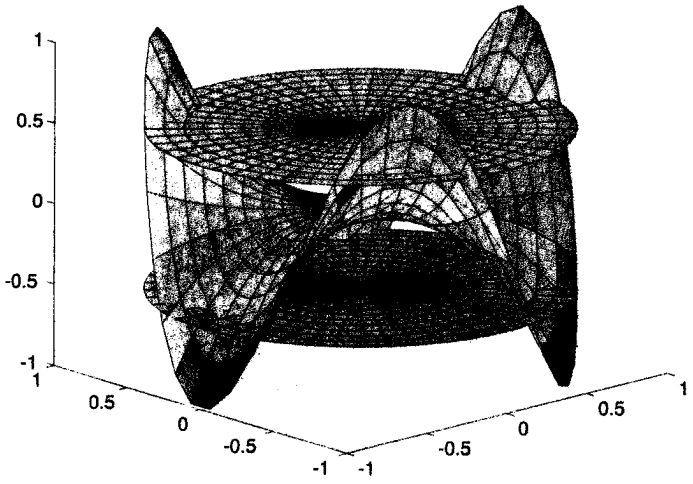
An Engineer's Guide to MATLAB®







EDWARD B. MAGRAB

SHAPOUR AZARM

BALAKUMAR BALACHANDRAN

JAMES DUNCAN

KEITH HEROLD

GREGORY WALSH

1.7.1 Online Help

MATLAB has a complete online help capability, which can be accessed several ways. One way is to click on the question mark (?) icon on the toolbar. This opens the *Help* window, which should be minimized after each use so as to be readily available from both the MATLAB command window and from the word processor/text editor window. Another way is to type in the MATLAB command window

```
help FunctionName
```

where *FunctionName* is the name of the function about which information is sought.

A third way is to select *Help Desk (HTML)* from the *Help* pull-down menu, which opens the computer system's web browser to display a more complete description of the command(s) in question. This form of information can also be accessed by clicking on the *Go to Help Desk* icon in the MATLAB help window that is accessed with the question mark (?) icon in the MATLAB command window.

1.8 BASIC MATLAB SYNTAX

MATLAB requires that all variable names (except those used by the Symbolic Toolbox) be assigned numerical values prior to being used in an expression. Typing the variable name, an equal sign, the numerical value(s), and then *Enter* performs the assignment. Thus, if we let $p = 7.1$, $x = 4.92$ and $k = -1.7$, then the following interaction in the MATLAB command window is obtained.

```
» p = 7.1          <---- User types
p =
   7.1000          <---- System responds
» x = 4.92         <---- User types
x =
   4.9200          <---- System responds
» k = -1.7         <---- User types
k =
  -1.7000          <---- System responds
```

If one wants to suppress the system's response, then a semicolon (;) is placed as the last character of the expression. Thus,

```
» p = 7.1;
» x = 4.92;
» k = -1.7;
»
```

MATLAB also lets one place several expressions on one line, a line being terminated by *Enter*. In this case, each expression is separated by either a comma (,) or a semicolon (;). When a comma is used, the system echoes the input. Thus, if the following is typed,

```
» p = 7.1, x = 4.92, k = –1.7
```

then the system responds with

```
p =
   7.1000
x =
   4.9200
k =
  –1.7000
»
```

The use of semicolons instead of the commas would have suppressed this output.

The five arithmetic operators to perform scalar addition, subtraction, multiplication, division and exponentiation are +, –, *, /, and ^, respectively. For example, the mathematical expression

$$t = \left(\frac{1}{1 + px} \right)^k$$

is written in MATLAB as

```
t = (1/(1+p*x))^k
```

The quantities p, x, and k must be assigned numerical values by the user prior to the execution of this statement. If this has not been done, then an error message to that effect will appear. Assuming that the quantities p, x and k are those entered previously, the system returns[3]

```
440.8779
```

The parentheses in the MATLAB expression for t have to be used so that the mathematical operations are performed on the proper collections of quantities in their proper order within each set of parentheses. There is a hierarchy that MATLAB uses to compute arithmetic statements so that the number of parentheses can be minimized. However, parentheses that are unnecessary from MATLAB's point of view can still be used to remove visual ambiguity and make the expression easier to understand. The highest level is exponentiation, followed by multiplication and division, and then addition and subtraction. Within each set of parentheses and with all expressions in general, MATLAB performs its operations from left to right. Consider the examples shown in Table 1.1 involving the scalar quantities c, d, g, and x. The MATLAB function

[3] From this point forward we shall not always reproduce the command window literally, and instead just indicate the output (answers) in a format appropriate to its context.

TABLE 1.1
Examples of MATLAB Syntax

Mathematical expression	MATLAB expression
$1-dc^{x+2}$	1-d*c^(x+2)
$dc^x + 2$	d*c^x+2 or 2+d*c^x
$(2/d)c^{x+2}$	(2/d)*c^(x+2) or 2/d*c^(x+2) or 2*c^(x+2)/d
$(dc^x + 2)/g^{2.7}$	(d*c^x+2)/g^2.7
$\sqrt{dc^x + 2}$	(d*c^x+2)^0.5 or `sqrt(d*c^x+2)`

`sqrt`

takes the square root of its argument.

MATLAB also includes a large set of elementary, and not so elementary, functions. Some of the elementary ones are listed in Tables 1.2 and 1.3. The arguments to these functions can be scalars, vectors, or matrices. The definitions of vectors and matrices and their creation in MATLAB are given in Sections 2.3 and 2.3.

An example of the use of MATLAB's built-in functions is illustrated by considering the following expression:

$$y = \sqrt{|\pi - \sin(x)/\cosh(a) - \ln_e(x+a)|}$$

In MATLAB this expression is written as

```
y = sqrt(abs(pi-sin(x)/cosh(a)-log(x+a)))
```

where the built-in function `pi` = π. It is assumed that x and a have been assigned numerical values prior to the execution of this statement.

1.9 SOME SUGGESTIONS ON HOW TO USE MATLAB

Listed below are some suggestions on how to use the MATLAB environment to efficiently create MATLAB scripts and functions.

- *Use the Help files extensively.* This will minimize errors caused by incorrect syntax and by incorrect or inappropriate application of a MATLAB function.
- *Write scripts and functions in a text editor and save them as m files.* This will save time, save the code, and greatly facilitate the debugging process, especially if the MATLAB editor/debugger is used.
- *Attempt to minimize the number of expressions comprising scripts and functions.* This usually leads to a tradeoff between readability and compactness, but it can encourage the search for MATLAB functions and procedures that can perform some of the steps faster and more directly.

TABLE 1.2
Some Elementary MATLAB Functions

Mathematical function	MATLAB expression
e^x	`exp(x)`
$\sqrt{x}$	`sqrt(x)`
$\ln(x)$ or $\log_e(x)$	`log(x)`
$\log_{10}(x)$	`log10(x)`
$\lvert x \rvert$	`abs(x)`
signum	`sign(x)`

TABLE 1.3
MATLAB's Trigonometric and Hyperbolic Functions

	Trigonometric		Hyperbolic	
Function	Function	Inverse function	Function	Inverse function
sine	`sin(x)`	`asin(x)`	`sinh(x)`	`asinh(x)`
cosine	`cos(x)`	`acos(x)`	`cosh(x)`	`acosh(x)`
tangent	`tan(x)`	`atan(x)` ⊠	`tanh(x)`	`atanh(x)`
secant	`sec(x)`	`asec(x)`	`sech(x)`	`asech(x)`
cosecant	`csc(x)`	`acsc(x)`	`csch(x)`	`acsch(x)`
cotangent	`cot(x)`	`acot(x)`	`coth(x)`	`acoth(x)`

⊠ `atan2(y,x)` is the four quadrant version.

- *When practical use graphical output as the script or function is being developed.* This usually shortens the code development process by identifying potential coding errors and can facilitate the understanding of the physical process being modeled or analyzed.
- *Most important, verify by independent means that the outputs from the scripts and functions are correct.*

EXERCISES

1.1 The following expressions[4] describe the principal contact stresses in the x-, y-, and z-directions, respectively, when two spheres are pressed together with a force F.

[4] J. E. Shigley and C. R. Mischke, *Mechanical Engineering Design*, 5th ed., McGraw-Hill, New York, 1989.

APPENDIX A

Summary of MATLAB Special Characters

TABLE A.1
MATLAB's Special Characters and a Summary of Their Usage

Character	Name	Usage
.	Period	(a) Decimal point.
		(b) Part of arithmetic operators to indicate a special type of vector or matrix operation, called the dot operation, such as $c = a.*b$.
,	Comma	(a) Separator within parentheses of matrix elements such as $b(2,7)$ and functions such as `besselj(1, x)` or brackets creating vectors such as v = [1,x] or the output of function arguments such as [x,s] = `max(a)`.
		(b) Placed at the end of an expression when several expressions appear on one line.
;	Semicolon	(a) Suppresses display of the results when placed at end of an expression, or series of expressions, appearing on one line.
		(b) Indicates the end of a row in matrix creation statement such as $m = [x\ y\ z;\ a\ b\ c]$.
:	Colon	(a) Separator in the vector creation expression x = a:b:c.
		(b) For a matrix z it indicates "all rows" when written as $z(:,k)$ or "all columns" when written as $z(k,:)$.
( )	Parentheses	(a) Denote subscript of an element of matrix z, where $z(j, k)$ is the element in row j and column k.
		(b) Delimiters in mathematical expressions such as $a^{(b+c)}$.
		(c) Delimiters for the arguments of functions, such as `sin`(x).
[ ]	Brackets	Creates an array of numbers, either a vector or a matrix, or strings (literals).
{ }	Braces	Creates a cell matrix or structure.
%	Percentage	Comment delimiter; used to indicate the beginning of a comment wherein the MATLAB compiler ignores everything to its right. The exception is when it is used inside a pair of quotes to define a string such as a = 'p1 = 14 % of the total'.
'	Quote or Apostrophe	(a) '*Expression*' indicates that *Expression* is a string (literal)
		(b) Indicates the transpose of a vector or matrix.
...	Ellipsis	Continuation of a MATLAB expression to the next line. Used to create more readable code.
	Blank	Context dependent: either ignored, indicates a delimiter in a data creation statement such as c = [a b] or is a character in a string statement.

These expressions could have been written compactly as

```
y = [-1 6 15 -7 31 2 -4 -5];
s = y(find(y<=0))
```

See Section 6.3.8 for another application of `find`.

One of the great advantages of MATLAB's implicit vector and matrix notation is that it provides the user with a compact way of performing a series of operations on an array of values. For example, suppose that we would like to determine $\sin(x)$ when x varies by $\pi/5$ from $-\pi \le x \le \pi$. Then the MATLAB statements

```
x = -pi:pi/5:pi;
y = sin(x)
```

yield the following vector of n = `length(y)` = 11 elements:

```
y→[0.0000 -0.5878 -0.9511 -0.9511 -0.5878 0.0000 0.5878  0.9511 0.9511
   0.5878  0.0000]
```

2.4 CREATION OF MATRICES

The (4×3) matrix a

$$a = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix} \rightarrow (4 \times 3)$$

is created in any of the following ways. Consider the three quantities a_{11}, a_{12}, and a_{13}. These can be combined to create a vector v_1 by

$$v_1 = [\, a_{11}\ a_{12}\ a_{13}]$$

In a similar manner, we can create three more vectors v_2, v_3, and v_4:

$$v_2 = [\, a_{21}\ a_{22}\ a_{23}]$$
$$v_3 = [\, a_{31}\ a_{32}\ a_{33}]$$
$$v_4 = [\, a_{41}\ a_{42}\ a_{43}]$$

We now use these four vectors to create a matrix a as follows:

$$a = [\, v_1;\ v_2;\ v_3;\ v_4]$$

where the semicolons are used to indicate the end of a row. Each row must have the *same* number of columns. This expression could have also been created directly with

$$a = [\, a_{11}\ a_{12}\ a_{13};\ a_{21}\ a_{22}\ a_{23};\ a_{31}\ a_{32}\ a_{33};\ a_{41}\ a_{42}\ a_{43}]$$

6.2 BASIC 2D PLOTTING COMMANDS

The basic 2D plotting command is

```
plot(u1,v1,c1,u2,v2,c2, ...)
```

where u_j and v_j are the x and y coordinates, respectively, of a point or a series of points. They are either a pair of numbers, vectors of the same length, matrices of the same order, or expressions that, when evaluated, result in one of these three quantities. The quantity c_j is string of characters: one character specifies the line/point color, one character specifies the point type if points are to be plotted, and up to two characters are used to specify the line characteristics. When a series of points are to be plotted, one of the characters of c_j can be, for example, an 's' to plot a square or an asterisk '*' to plot an asterisk. When the points, whether or not they are to be displayed, are to be connected with (straight) lines, the characters of c_j can be, for example, a '-' for a solid line and a '--' for a dashed line. When both the lines and points are to be plotted with the same color, the c_j contains both descriptors. For example, if we were to plot blue dashed lines connecting blue diamonds, c_j would be 'b--d'. The order of the three sets of characters within the single quotes is not important. When both lines and points are to be plotted but the number of points defining the line is different from the number of points that are to be plotted, c_1 contains the symbol for the line type and c_2 contains the symbol for the point type, or *vice versa*. See the Help file for `plot` for a list of the colors and line and point types provided. If c_j is omitted, then the system's default values are used. If more than one curve is drawn, then the line colors change according to the default sequence.

We now describe the statements used to draw points, lines, circles, expressions, families of curves, and curves described by multiple functions.

6.2.1 Points

To place a red asterisk at the location (2,4), the plotting instruction is

```
plot(2,4,'r*')
```

6.2.2 Lines

To draw a straight line that goes from (0,0) to (1,2) using the default line type (solid) and the default color (blue), the plotting instruction is

```
plot([0 1],[0 2])
```

Notice that the first two-element vector [0 1] represents the values of the x coordinates, and the second two-element vector [0 2] represents the values of the y coordinates. Thus, the first element of each vector defines the (x,y) coordinates of one endpoint of the line, and the second elements of these vectors are the coordinates of the other endpoint.

Suppose that we want to draw a set of n unconnected lines whose end points are (x_{1n},y_{1n}) and (x_{2n},y_{2n}). To accomplish this we create four vectors:

6.3 GRAPH ANNOTATION AND VISUAL ENHANCEMENT

6.3.1 Axes and Curve Labels, Figure Titles, Legends, Text, and Other Attributes

We shall now illustrate, through an example, how to enhance a graph:

- With axis labels, figure titles, labeled curves, legends, filled areas, and placement of text
- By altering the attributes of the axes, curve lines, and the text
- By using Greek letters, mathematical symbols, and subscripts and superscripts

Let us draw, label, title and annotate the relationship of two intersecting curves, $\cos(x)$ and $1/\cosh(x)$, over the range $0 \leq x \leq 6$. In this range, these two curves intersect at $x = 4.73$. Recall the example in Section 5.5.1. We shall also draw a vertical line through the intersecting point and denote the value of x near this intersection. The script to perform this is

```
x = 0:.05:6;
plot(x,cos(x),'k',x,1./cosh(x),'k',[4.73 4.73],[-1 1],'k')
xlabel('x')
ylabel('Value of functions')
title('Visualization of two intersecting curves')
text(4.8,-.1,'x = 4.73')
text(2.1,.3,'1/cosh(x)')
text(1.2,-.4,'cos(x)')
```

Execution of the script results in Figure 6.8. The coordinate values for the location of the various texts are chosen after only the `plot` function is executed—that is, after only the first two lines of the script have been written and the resulting figure examined. Then the `text` functions are added.

We can modify these results so that the area between the two curves in the range $0 \leq x \leq 4.73$ is filled with the color cyan. To fill the area contained within these two curves we use

```
fill
```

which requires that these two curves be turned into a connected polygon. In doing this, we must also create a new range for x: $0 \leq x \leq 4.73$. To accomplish this, the following expressions are appended to the above script:

```
xn = linspace(0,4.73,50);
hold on
fill([xn fliplr(xn)], [1./cosh(xn) fliplr(cos(xn))],'c');
```

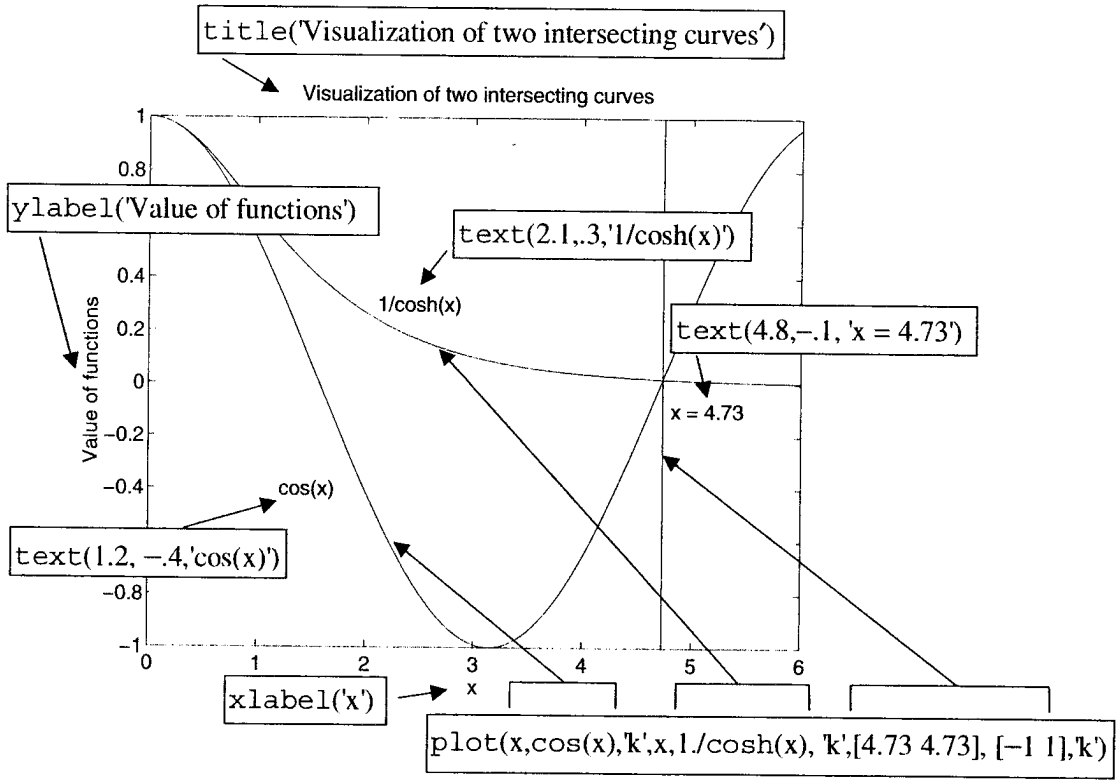


FIGURE 6.8.
Expressions that create and annotate the figure shown.

The result of appending this script to the previous one gives Figure 6.9. The connected polygon is created by forming the vector [1./cosh(xn) fliplr(cos(xn))], which is the concatenation of the top curve 1/cosh(x) and the reversal of the vector of cos(x), the bottom curve. Corresponding to this new vector is the new x-coordinate vector [xn fliplr(xn)], which is formed by the concatenation of the new values of x and its reversed values.

If, instead of using fill, the area between these two curves is delineated by a series of 20 equally spaced vertical lines, then the script is

```
x = 0:.05:6;
plot(x,cos(x),'k',x,1./cosh(x),'k',[4.73 4.73],[−1 1],'k')
hold on
xx = linspace(0,4.73,20);
plot([xx;xx],[cos(xx);1./cosh(xx)],'k-')
```

When executed, this script produces the results shown in Figure 6.10a.

To create 20 equally spaced horizontal lines, we have to work with the inverse functions $\cos^{-1}(x)$ and $\cosh^{-1}(x)$. For this case the script is