APPENDIX

A MATLAB Basics

A.1 INTRODUCTION

MATLAB is an interactive program for scientific and engineering calculations. The MATLAB family of programs includes the base program plus a variety of **toolboxes**, a collection of special files called *m-files* that extend the functionality of the base program [1–8]. Together, the base program plus the *Control System Toolbox* provide the capability to use MATLAB for control system design and analysis. Whenever MATLAB is referenced in this book, it means the base program plus the *Control System Toolbox.*

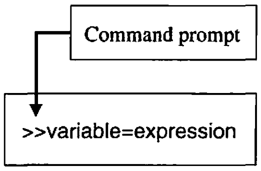
Most of the statements, functions, and commands are computer-platform-independent. Regardless of what particular computer system you use, your interaction with MATLAB is basically the same. This appendix concentrates on this computer platform–independent interaction. A typical session will utilize a variety of objects that allow you to interact with the program: (1) statements and variables, (2) matrices, (3) graphics, and (4) scripts. MATLAB interprets and acts on input in the form of one or more of these objects. The goal in this appendix is to introduce each of the four objects in preparation for our ultimate goal of using MATLAB for control system design and analysis.

The manner in which MATLAB interacts with a specific computer system is computer-platform-dependent. Examples of computer-dependent functions include installation, the file structure, hard-copy generation of the graphics, the invoking and exiting of a session, and memory allocation. Questions related to platform-dependent issues are not addressed here. This does not mean that they are not important, but rather that there are better sources of information such as the MATLAB *User's Guide* or the local resident expert.

The remainder of this appendix consists of four sections corresponding to the four objects already listed. In the first section, we present the basics of **statements** and **variables**. Following that is the subject of **matrices**. The third section presents an introduction to **graphics**, and the fourth section is a discussion on the important topic of **scripts** and **m-files**. All the figures in this appendix can be constructed using the m-files found at the MCS website.

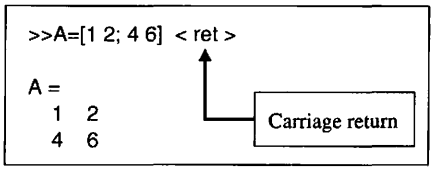
A.2 STATEMENTS AND VARIABLES

Statements have the form shown in Figure A.1. MATLAB uses the assignment so that equals ("=") implies the assignment of the expression to the variable. The command



```
>>variable=expression
```

FIGURE A.1 MATLAB statement form.



```
>>A=[1 2; 4 6] < ret >

A =
  1  2
  4  6
```

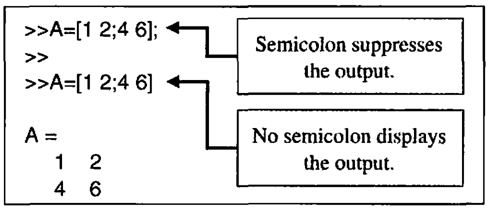
FIGURE A.2 Entering and displaying a matrix **A**.

prompt is two right arrows, " >> ." A typical statement is shown in Figure A.2, where we are entering a 2 × 2 matrix to which we attach the variable name *A*. The statement is executed after the carriage return (or enter key) is pressed. The carriage return is not explicitly denoted in the remaining examples in this appendix.

The matrix **A** is automatically displayed after the statement is executed following the carriage return. If the statement is followed by a semicolon (;), the output matrix **A** is suppressed, as seen in Figure A.3. The assignment of the variable **A** has been carried out even though the output is suppressed by the semicolon. It is often the case that your MATLAB sessions will include intermediate calculations for which the output is of little interest. Use the semicolon whenever you have a need to reduce the amount of output. Output management has the added benefit of increasing the execution speed of the calculations since displaying screen output takes time.

The usual mathematical operators can be used in expressions. The common operators are shown in Table A.1. The order of the arithmetic operations can be altered by using parentheses.

The example in Figure A.4 illustrates that MATLAB can be used in a "calculator" mode. When the variable name and "=" are omitted from an expression, the result is assigned to the generic variable *ans*. MATLAB has available most of the trigonometric and elementary math functions of a common scientific calculator. Type help elfun at the command prompt to view a complete list of available trigonometric and elementary math functions; the more common ones are summarized in Table A.2.


Semicolon suppresses the output.
No semicolon displays the output.

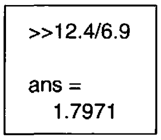
```
>>A=[1 2;4 6];
>>
>>A=[1 2;4 6]

A =
  1  2
  4  6
```

FIGURE A.3 Using semicolons to suppress the output.

Table A.1 Mathematical Operators

+	Addition
−	Subtraction
*	Multiplication
/	Division
^	Power



```
>>12.4/6.9

ans =
  1.7971
```

FIGURE A.4 Using the calculator mode.

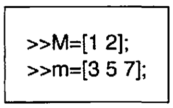
Table A.2 Common Mathematical Functions

sin(x)	Sine	acoth(x)	Inverse hyperbolic cotangent
sinh(x)	Hyperbolic sine	exp(x)	Exponential
asin(x)	Inverse sine	log(x)	Natural logarithm
asinh(x)	Inverse hyperbolic sine	log10(x)	Common (base 10) logarithm
cos(x)	Cosine	log2(x)	Base 2 logarithm and dissect floating point number
cosh(x)	Hyperbolic cosine	pow2(x)	Base 2 power and scale floating point number
acos(x)	Inverse cosine	sqrt(x)	Square root
acosh(x)	Inverse hyperbolic cosine	nextpow2(x)	Next higher power of 2
tan(x)	Tangent	abs(x)	Absolute value
tanh(x)	Hyperbolic tangent	angle(x)	Phase angle
atan(x)	Inverse tangent	complex(x,y)	Construct complex data from real and imaginary parts
atan2(y,x)	Four quadrant inverse tangent	conj(x)	Complex conjugate
atanh(x)	Inverse hyperbolic tangent	imag(x)	Complex imaginary part
sec(x)	Secant	real(x)	Complex real part
sech(x)	Hyperbolic secant	unwrap(x)	Unwrap phase angle
asec(x)	Inverse secant	isreal(x)	True for real array
asech(x)	Inverse hyperbolic secant	cplxpair(x)	Sort numbers into complex conjugate pairs
csc(x)	Cosecant	fix(x)	Round towards zero
csch(x)	Hyperbolic cosecant	floor(x)	Round towards minus infinity
acsc(x)	Inverse cosecant	ceil(x)	Round towards plus infinity
acsch(x)	Inverse hyperbolic cosecant	round(x)	Round towards nearest integer
cot(x)	Cotangent	mod(x,y)	Modulus (signed remainder after division)
coth(x)	Hyperbolic cotangent	rem(x,y)	Remainder after division
acot(x)	Inverse cotangent		

Variable names begin with a letter and are followed by any number of letters and numbers (including underscores). Keep the name length to N characters, since MATLAB remembers only the first N characters, where N = namelengthmax. It is a good practice to use variable names that describe the quantity they represent. For example, we might use the variable name *vel* to represent the quantity *aircraft velocity*. Generally, we do not use extremely long variable names even though they may be legal MATLAB names.

Since MATLAB is **case sensitive**, the variables *M* and *m* are not the same. By **case**, we mean upper- and lowercase, as illustrated in Figure A.5. The variables *M* and *m* are recognized as different quantities.

MATLAB has several predefined variables, including *pi*, *Inf*, *NaN*, *i*, and *j*. Three examples are shown in Figure A.6. *NaN* stands for *Not-a-Number* and results from undefined operations. *Inf* represents $+\infty$, and *pi* represents π. The variable $i = \sqrt{-1}$ is used to represent complex numbers. The variable $j = \sqrt{-1}$ can be used for complex arithmetic by those who prefer it over *i*. These predefined variables can be inadvertently overwritten. Of course, they can also be purposely overwritten in order to free the variable name for other uses. For instance, you might want to use *i* as an integer and reserve *j* for complex arithmetic. Be safe and leave these predefined variables alone, as



```
>>M=[1 2];
>>m=[3 5 7];
```

FIGURE A.5 Variables are case sensitive.

```
>>z=3+4*i
z =
  3.0000 + 4.0000i

>>Inf
ans =
  Inf

>>0/0
ans =
  NaN
```

FIGURE A.6 Three predefined variables *i*, *Inf*, and *NaN*.

```
>>who
Your variables are:
A    M    ans    m    z
```

FIGURE A.7 Using the **who** function to display variables.

there are plenty of alternative names that can be used. Predefined variables can be reset to their default values by using clear *name* (e.g., clear *pi*).

The matrix **A** and the variable *ans*, in Figures A.3 and A.4, respectively, are stored in the **workspace.** Variables in the workspace are automatically saved for later use in your session. The who function gives a list of the variables in the workspace, as shown in Figure A.7. MATLAB has a host of built-in functions. Refer to the MATLAB *User's Guide* for a complete list or use the MATLAB help browser. Each function will be described as the need arises.

The whos function lists the variables in the workspace and gives additional information regarding variable dimension, type, and memory allocation. Figure A.8 gives an example of the whos function. The memory allocation information given by the whos function can be interpreted as follows: Each element of the 2 × 2 matrix **A** requires 8 bytes of memory for a total of 32 bytes, the 1 × 1 variable *ans* requires 8 bytes, and so forth. All the variables in the workspace use a total of 96 bytes.

Variables can be removed from the workspace with the clear function. Using the function clear, by itself, removes all items (variables and functions) from the workspace; clear variables removes all variables from the workspace; clear *name*1 *name*2... removes the variables *name1*, *name2*, and so forth. The procedure for removing the matrix **A** from the workspace is shown in Figure A.9.

```
>>whos
          Name    Size    Bytes    Class     Attributes

           A      2x2      32      double
           M      1x2      16      double
          ans     1x1       8      double
           m      1x3      24      double
           z      1x1      16      double    complex
```

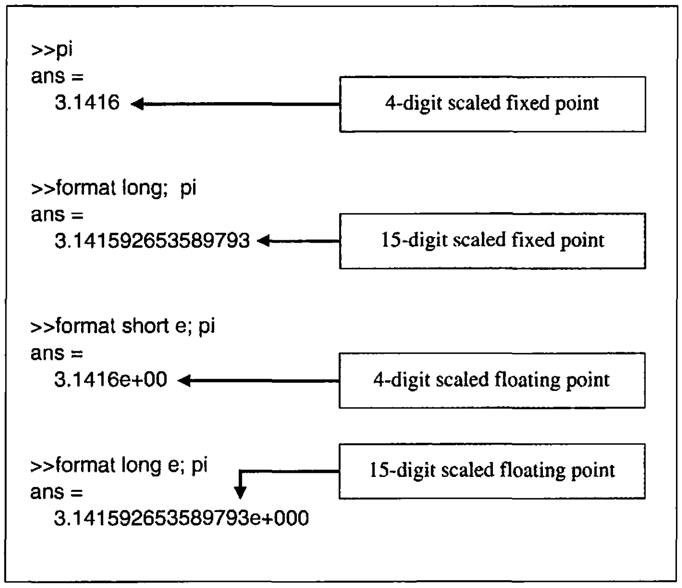
FIGURE A.8 Using the **whos** function to display variables.

```
>>clear A
>>who
Your variables are:
M      ans      m      z
```

FIGURE A.9 Removing the matrix **A** from the workspace.

Computations in MATLAB are performed in **double precision**. However, the screen output can be displayed in several formats. The default output format contains four digits past the decimal point for nonintegers. This can be changed by using the format function shown in Figure A.10. Once a particular format has been specified, it remains in effect until altered by a different format input. The output format does not affect internal MATLAB computations. On the other hand, the number of digits displayed does not necessarily reflect the number of significant digits of the number. This is problem-dependent, and only the user can know the true accuracy of the numbers input and displayed by MATLAB. Other display formats (not shown in Figure A.10) include format long g (best of fixed or floating point format with 14 digits after the decimal point), format short g (same as format long g but with 4 digits after the decimal point), format hex (hexadecimal format), format bank (fixed format for dollars and cents), format rat (ratio of small integers) and format (same as format short).

Since MATLAB is case sensitive, the functions who and WHO are not the same functions. The first function, who, is a built-in function, and typing who lists the variables in the workspace. On the other hand, typing the uppercase WHO results in the error message shown in Figure A.11. Case sensitivity applies to all functions.



```
>>pi
ans =
  3.1416

>>format long; pi
ans =
  3.141592653589793

>>format short e; pi
ans =
  3.1416e+00

>>format long e; pi
ans =
  3.141592653589793e+000
```

4-digit scaled fixed point
15-digit scaled fixed point
4-digit scaled floating point
15-digit scaled floating point

FIGURE A.10 Output format control illustrates the four forms of output.

```
>>WHO
??? Undefined
function or variable 'WHO'.

>>Who
??? Undefined function or
variable 'Who'.
```

FIGURE A.11 Function names are case sensitive.

A.3 MATRICES

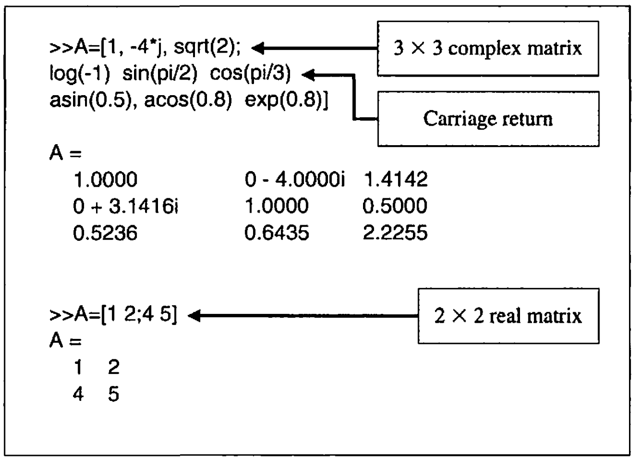
MATLAB is short for **matrix laboratory**. Although we will not emphasize the matrix routines underlying our calculations, we will learn how to use the interactive capability to assist us in the control system design and analysis. We begin by introducing the basic concepts associated with manipulating matrices and vectors.

The basic computational unit is the matrix. Vectors and scalars can be viewed as special cases of matrices. A typical matrix expression is enclosed in square brackets, [•]. The column elements are separated by blanks or commas, and the rows are separated by semicolons or carriage returns. Suppose we want to input the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & -4j & \sqrt{2} \\ \log(-1) & \sin(\pi/2) & \cos(\pi/3) \\ \text{asin}(0.8) & \text{acos}(0.8) & \exp(0.8) \end{bmatrix}.$$

One way to input **A** is shown in Figure A.12. The input style in Figure A.12 is not unique.

Matrices can be input across multiple lines by using a carriage return following the semicolon or in place of the semicolon. This practice is useful for entering large matrices. Different combinations of spaces and commas can be used to separate the columns, and different combinations of semicolons and carriage returns can be used to separate the rows, as illustrated in Figure A.12.



```
>>A=[1, -4*j, sqrt(2);
log(-1) sin(pi/2) cos(pi/3)
asin(0.5), acos(0.8) exp(0.8)]

A =
  1.0000          0 - 4.0000i  1.4142
  0 + 3.1416i     1.0000       0.5000
  0.5236          0.6435       2.2255


>>A=[1 2;4 5]
A =
  1  2
  4  5
```

FIGURE A.12 Complex and real matrix input with automatic dimension and type adjustment.

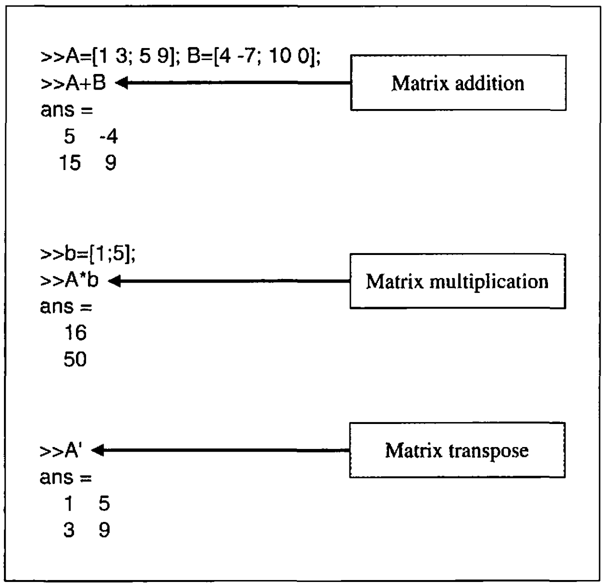
No dimension statements or type statements are necessary when using matrices; memory is allocated automatically. Notice in the example in Figure A.12 that the size of the matrix **A** is automatically adjusted when the input matrix is redefined. Also notice that the matrix elements can contain trigonometric and elementary math functions, as well as complex numbers.

The important basic matrix operations are addition and subtraction, multiplication, transpose, powers, and the so-called array operations, which are element-to-element operations. The mathematical operators given in Table A.1 apply to matrices. We will not discuss matrix division, but be aware that MATLAB has a left- and right-matrix division capability.

Matrix operations require that the matrix dimensions be compatible. For matrix addition and subtraction, this means that the matrices must have the same dimensions. If **A** is an $n \times m$ matrix and **B** is a $p \times r$ matrix, then **A** ± **B** is permitted only if $n = p$ and $m = r$. Matrix multiplication, given by **A** * **B**, is permitted only if $m = p$. Matrix-vector multiplication is a special case of matrix multiplication. Suppose **b** is a vector of length p. Multiplication of the vector **b** by the matrix **A**, where **A** is an $n \times m$ matrix, is allowed if $m = p$. Thus, **y** = **A** * **b** is the $n \times 1$ vector solution of **A** * **b**. Examples of three basic matrix-vector operations are given in Figure A.13.

The matrix transpose is formed with the apostrophe ('). We can use the matrix transpose and multiplication operation to create a vector **inner product** in the following manner. Suppose **w** and **v** are $m \times 1$ vectors. Then the inner product (also known as the dot product) is given by **w**' * **v**. The inner product of two vectors is a scalar. The **outer product** of two vectors can similarly be computed as **w** * **v**'. The outer product of two $m \times 1$ vectors is an $m \times m$ matrix of rank 1. Examples of inner and outer products are given in Figure A.14.

The basic matrix operations can be modified for element-by-element operations by preceding the operator with a period. The modified matrix operations are known

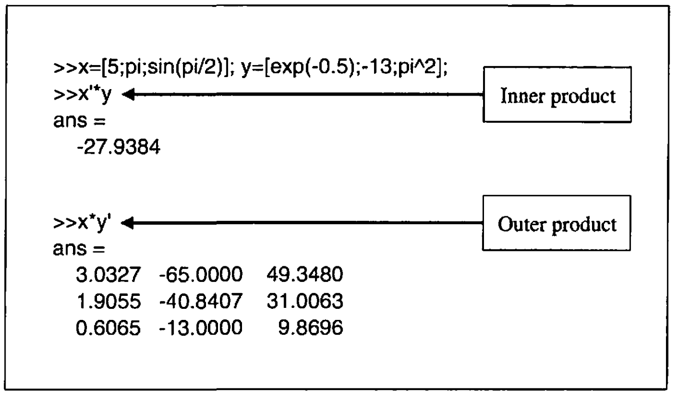


```
>>A=[1 3; 5 9]; B=[4 -7; 10 0];
>>A+B
ans =
  5  -4
  15  9

>>b=[1;5];
>>A*b
ans =
  16
  50

>>A'
ans =
  1  5
  3  9
```

FIGURE A.13 Three basic matrix operations: addition, multiplication, and transpose.



```
>>x=[5;pi;sin(pi/2)]; y=[exp(-0.5);-13;pi^2];
>>x'*y
ans =
  -27.9384

>>x*y'
ans =
  3.0327  -65.0000  49.3480
  1.9055  -40.8407  31.0063
  0.6065  -13.0000   9.8696
```

FIGURE A.14 Inner and outer products of two vectors.

Table A.3 Mathematical Array Operators

+	Addition
–	Subtraction
.*	Multiplication
./	Division
.^	Power

as **array operations**. The commonly used array operators are given in Table A.3. Matrix addition and subtraction are already element-by-element operations and do not require the additional period preceding the operator. However, array multiplication, division, and power do require the preceding dot, as shown in Table A.3.

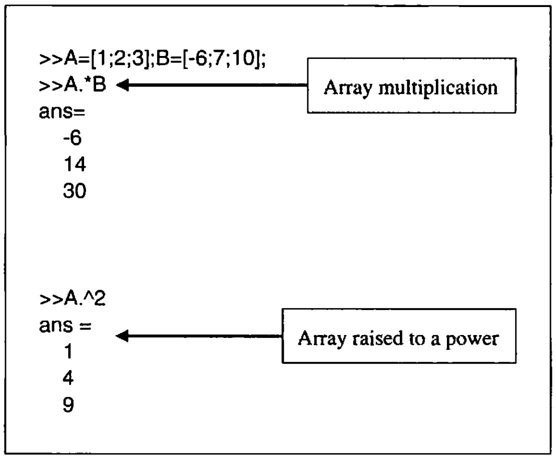
Consider **A** and **B** as 2 × 2 matrices given by

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}.$$

Then, using the array multiplication operator, we have

$$\mathbf{A}.{*}\,\mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}.$$

The elements of **A**.* **B** are the products of the corresponding elements of **A** and **B**. A numerical example of two array operations is given in Figure A.15.



```
>>A=[1;2;3];B=[-6;7;10];
>>A.*B
ans=
  -6
  14
  30

>>A.^2
ans =
  1
  4
  9
```

FIGURE A.15 Array operations.

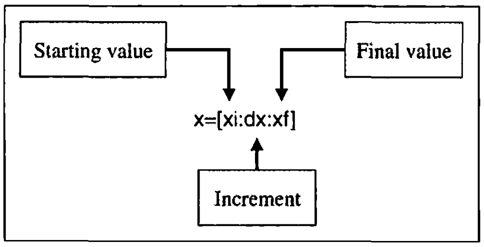


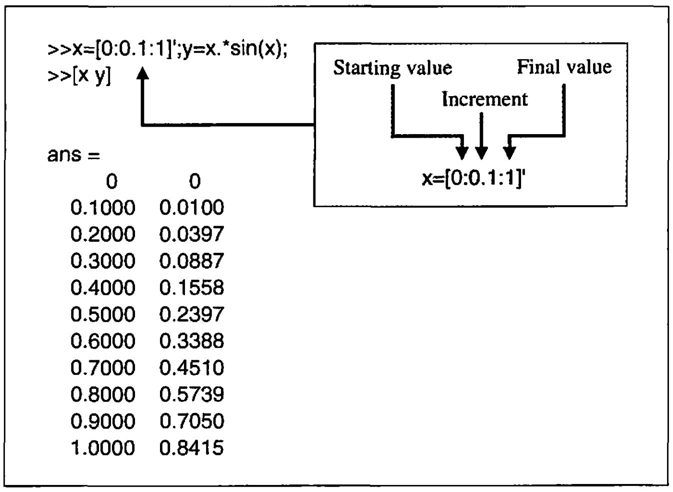
FIGURE A.16
The colon notation.

Before proceeding to the important topic of graphics, we need to introduce the notion of **subscripting using colon notation**. The colon notation, shown in Figure A.16, allows us to generate a row vector containing the numbers from a given starting value, x_i, to a final value, x_f, with a specified increment, dx.

We can easily generate vectors using the colon notation, and as we shall soon see, this is quite useful for developing ***x*-*y* plots**. Suppose our objective is to generate a plot of $y = x\sin(x)$ versus x for $x = 0, 0.1, 0.2, \ldots, 1.0$. Our first step is to generate a table of x-y data. We can generate a vector containing the values of x at which the values of $y(x)$ are desired using the colon notation, as illustrated in Figure A.17. Given the desired x vector, the vector $y(x)$ is computed using the multiplication array operation. Creating a plot of $y = x\sin(x)$ versus x is a simple step once the table of x-y data is generated.

A.4 GRAPHICS

Graphics plays an important role in both the design and analysis of control systems. An important component of an **interactive** control system design and analysis tool is an effective graphical capability. A complete solution to the control system design and analysis will eventually require a detailed look at a multitude of data types in many formats. The objective of this section is to acquaint the reader with the basic



```
>>x=[0:0.1:1]';y=x.*sin(x);
>>[x y]

ans =
      0         0
 0.1000    0.0100
 0.2000    0.0397
 0.3000    0.0887
 0.4000    0.1558
 0.5000    0.2397
 0.6000    0.3388
 0.7000    0.4510
 0.8000    0.5739
 0.9000    0.7050
 1.0000    0.8415
```

FIGURE A.17
Generating vectors using the colon notation.

Table A.4 Plot Formats

plot(x,y)	Plots the vector **x** versus the vector **y**.
semilogx(x,y)	Plots the vector **x** versus the vector **y**. The x-axis is $\log_{10}$; the y-axis is linear.
semilogy(x,y)	Plots the vector **x** versus the vector **y**. The x-axis is linear; the y-axis is $\log_{10}$.
loglog(x,y)	Plots the vector **x** versus the vector **y**. Creates a plot with $\log_{10}$scales on both axes.

x-y plotting capability of MATLAB. More advanced graphics topics are addressed in the chapter sections on MATLAB.

MATLAB uses a **graph display** to present plots. The graph display is activated automatically when a plot is generated using any function that generates a plot (e.g., the plot function). The plot function opens a graph display, called a **FIGURE** window. You can also create a new figure window with the figure function. Multiple figure windows can exist in a single MATLAB session; the function figure (n) makes n the current figure. The plot in the graph display is cleared by the clf function at the command prompt. The shg function brings the current figure window forward.

There are two basic groups of graphics functions. The first group, shown in Table A.4, specifies the type of plot. The list of available plot types includes the x-y plot, semilog plots, and log plots. The second group of functions, shown in Table A.5, allows us to customize the plots by adding titles, axis labels, and text to the plots and to change the scales and display multiple plots in subwindows.

The standard x-y plot is created using the plot function. The x-y data in Figure A.17 are plotted using the plot function, as shown in Figure A.18. The axis scales and line types are automatically chosen. The axes are labeled with the xlabel and ylabel functions; the title is applied with the title function. The legend function puts a legend on the current figure. A grid can be placed on the plot by using the grid on function. A basic x-y plot is generated with the combination of functions plot, legend, xlabel, ylabel, title, and grid on.

Multiple lines can be placed on the graph by using the plot function with multiple arguments, as shown in Figure A.19. The default line types can also be altered. The available line types are shown in Table A.6. The line types will be automatically

Table A.5 Functions for Customized Plots

title('text')	Puts 'text' at the top of the plot
legend (string1, string2,...)	Puts a legend on current plot using specified strings as labels
xlabel('text')	Labels the x-axis with 'text'
ylabel('text')	Labels the y-axis with 'text'
text(p1,p2, 'text')	Adds 'text' to location (p1,p2), where (p1,p2) is in units from the current plot
subplot	Subdivides the graphics window
grid on	Adds grid lines to the current figure
grid off	Removes grid lines from the current figure
grid	Toggles the grid state

Table A.6 Commands for Line Types for Customized Plots

-	Solid line
- -	Dashed line
:	Dotted line
-.	Dashdot line

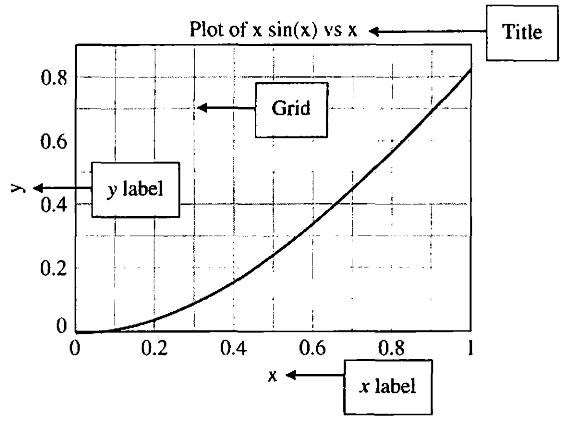
chosen unless specified by the user. The use of the text function and the changing of line types are illustrated in Figure A.19.

The other graphics functions—loglog, semilogx, and semilogy—are used in a fashion similar to that of plot. To obtain an x-y plot where the x-axis is a linear scale and the y-axis is a $\log_{10}$ scale, you would use the semilogy function in place of the plot function. The customizing features listed in Table A.5 can also be utilized with the loglog, semilogx, and semilogy functions.

The graph display can be subdivided into smaller subwindows. The function subplot(m,n,p) subdivides the graph display into an $m \times n$ grid of smaller subwindows. The integer p specifies the window, numbered left to right, top to bottom, as illustrated in Figure A.20, where the graphics window is subdivided into four subwindows.

```
>>x=[0:0.1:1]';
>>y=x.*sin(x);
>>plot(x,y)
>>title('Plot of x sin(x) vs x ')
>>xlabel('x')
>>ylabel('y')
>>grid on
```

(a)



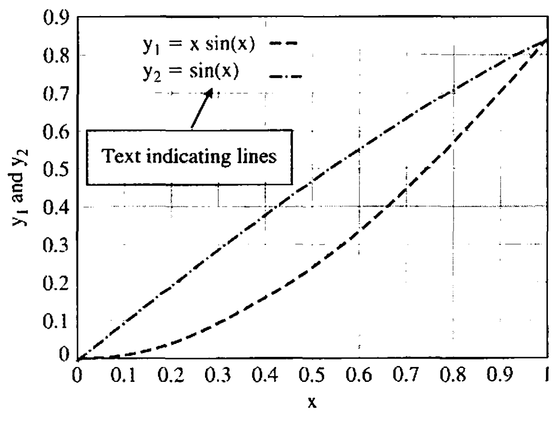
(b)

FIGURE A.18 (a) MATLAB commands. (b) A basic x-y plot of x sin(x) versus x.

```
>> x=[0:0.1:1]';
>> y1=x.*sin(x); y2=sin(x);
>> plot(x,y1,'--',x,y2,'-.')
>> text(0.1,0.85,'y_1 = x sin(x) ---')
>> text(0.1,0.80,'y_2 = sin(x) .\_.\_')
>> xlabel('x'), ylabel('y_1 and y_2'), grid on
```

Dashed line for y1
Dashed-dot line for y2

(a)



(b)

FIGURE A.19 (a) MATLAB commands. (b) A basic x-y plot with multiple lines.

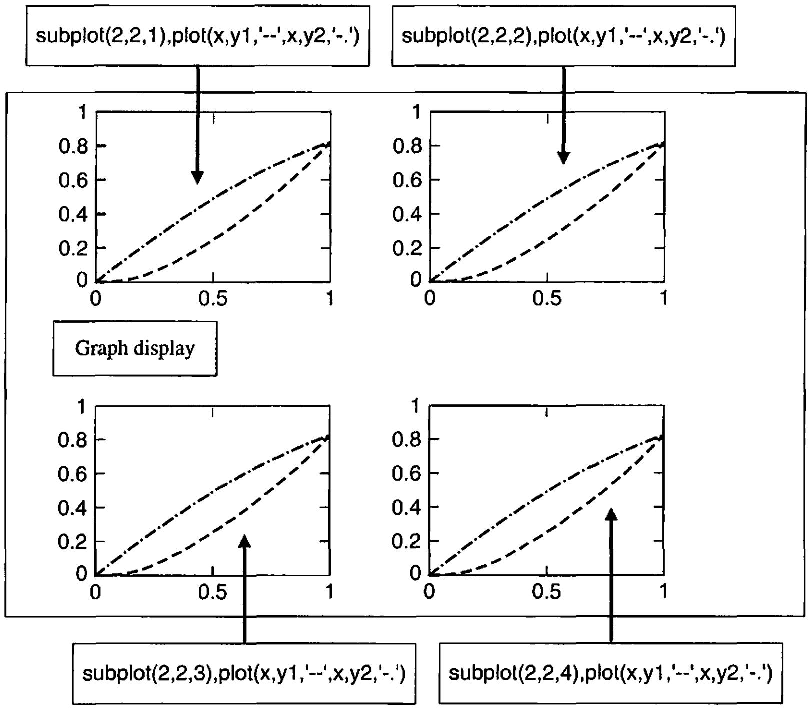


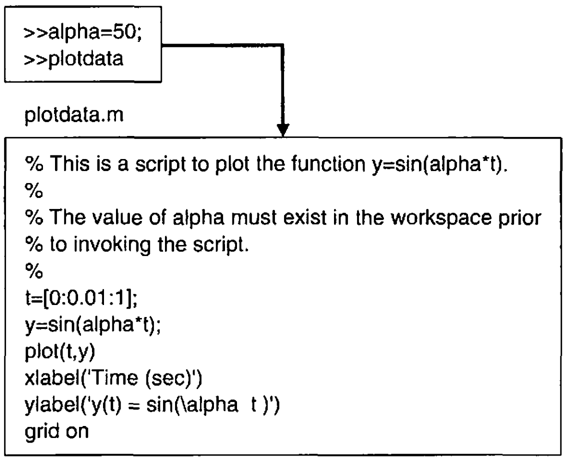
FIGURE A.20 Using **subplot** to create a 2 × 2 partition of the graph display.

A.5 SCRIPTS

Up to this point, all of our interaction with MATLAB has been at the command prompt. We entered statements and functions at the command prompt, and MATLAB interpreted our input and took the appropriate action. This is the preferable mode of operation whenever the work sessions are short and non-repetitive. However, the real power of MATLAB for control system design and analysis derives from its ability to execute a long sequence of commands stored in a file. These files are called **m-files**, since the filename has the form *filename.m*. A **script** is one type of m-file. The *Control System Toolbox* is a collection of m-files designed specifically for control applications. In addition to the preexisting m-files delivered with MATLAB and the toolboxes, we can develop our own scripts for our applications. Scripts are ordinary ASCII text files and are created using a text editor.

A script is a sequence of ordinary statements and functions used at the command prompt level. A script is invoked at the command prompt level by typing in the filename or by using the pull-down menu. Scripts can also invoke other scripts. When the script is invoked, MATLAB executes the statements and functions in the file without waiting for input at the command prompt. The script operates on variables in the workspace.

Suppose we want to plot the function $y(t) = \sin \alpha t$, where α is a variable that we want to vary. Using a text editor, we write a script that we call plotdata.m, as



```
>>alpha=50;
>>plotdata
```

plotdata.m

```
% This is a script to plot the function y=sin(alpha*t).
%
% The value of alpha must exist in the workspace prior
% to invoking the script.
%
t=[0:0.01:1];
y=sin(alpha*t);
plot(t,y)
xlabel('Time (sec)')
ylabel('y(t) = sin(\alpha  t )')
grid on
```

FIGURE A.21
A simple script to plot the function $y(t) = \sin \alpha t$.

shown in Figure A.21, then input a value of α at the command prompt, placing α in the workspace. Then we execute the script by typing in plotdata at the command prompt; the script plotdata.m will use the most recent value of α in the workspace. After executing the script, we can enter another value of α at the command prompt and execute the script again.

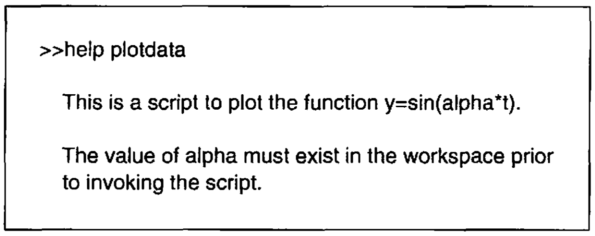
Your scripts should be well documented with **comments**, which begin with a %. Put a **header** in the script; make sure the header includes several descriptive comments regarding the function of the script, and then use the help function to display the header comments and describe the script to the user, as illustrated in Figure A.22.

Use plotdata.m to develop an interactive capability with α as a variable, as shown in Figure A.23. At the command prompt, input a value of $\alpha = 10$ followed by the script filename, which in this case is plotdata. The graph of $y(t) = \sin \alpha t$ is automatically generated. You can now go back to the command prompt, enter a value of $\alpha = 50$, and run the script again to obtain the updated plot.

A limited subset of TeX[1] characters are available to allow you to annotate plots with symbols and mathematical characters. Table A.7 shows the available symbols. Figure A.21 illustrates the use of '\alpha' to generate the α character in the y-axis label. The '\' character preceeds all TeX sequences. Also, you can modify the characters with the following modifiers:

- \bf—bold font
- \it—italics font
- \rm—normal font
- \fontname—specify the name of the font family to use
- \fontsize—specify the font size
- \color—specify color for succeeding characters

[1]TeX is a trademark of the American Mathematical Society.



```
>>help plotdata

  This is a script to plot the function y=sin(alpha*t).

  The value of alpha must exist in the workspace prior
  to invoking the script.
```

FIGURE A.22 Using the **help** function.

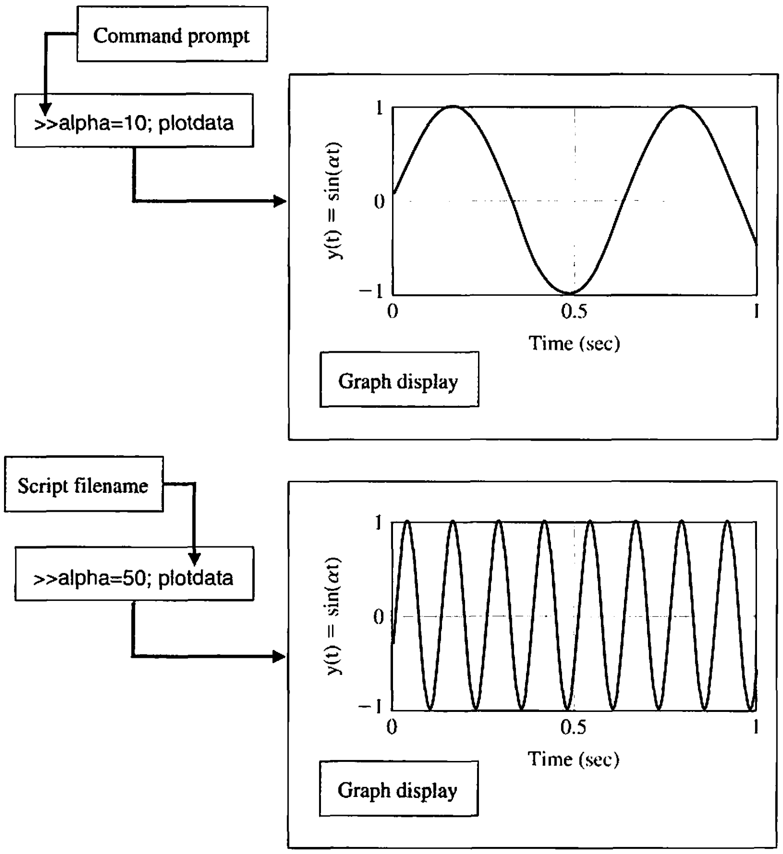


FIGURE A.23 An interactive session using a script to plot the function $y(t) = \sin \alpha t$.

Subscripts and superscripts are obtained with "_" and "^", respectively. For example, ylabel('y_1 and y_2') generates the y-axis label shown in Figure A.19.

The graphics capability of MATLAB extends beyond the introductory material presented here. A table of MATLAB functions used in this book is provided in Table A.8.

Table A.7 TeX Symbols and Mathematics Characters

Character Sequence	Symbol	Character Sequence	Symbol	Character Sequence	Symbol
\alpha	α	\upsilon	υ	\sim	~
\beta	β	\phi	φ	\leq	≤
\gamma	γ	\chi	χ	\infty	∞
\delta	δ	\psi	ψ	\clubsuit	♣
\epsilon	ϵ	\omega	ω	\diamondsuit	♦
\zeta	ζ	\Gamma	Γ	\heartsuit	♥
\eta	η	\Delta	Δ	\spadesuit	♠
\theta	θ	\Theta	Θ	\leftrightarrow	↔
\vartheta	ϑ	\Lambda	Λ	\leftarrow	←
\iota	ι	\Xi	Θ	\uparrow	↑
\kappa	κ	\Pi	Π	\rightarrow	→
\lambda	λ	\Sigma	Σ	\downarrow	↓
\mu	μ	\Upsilon	ϒ	\circ	∘
\nu	ν	\Phi	Φ	\pm	±
\xi	ξ	\Psi	Ψ	\geq	≥
\pi	π	\Omega	Ω	\propto	∝
\rho	ρ	\forall	∀	\partial	∂
\sigma	σ	\exists	∃	\bullet	•
\varsigma	ς	\ni	∍	\div	÷
\tau	τ	\cong	≅	\neq	≠
\equiv	≡	\approx	≈	\aleph	ℵ
\Im	ℑ	\Re	ℜ	\wp	℘
\otimes	⊗	\oplus	⊕	\oslash	∅
\cap	∩	\cup	∪	\supseteq	⊇
\supset	⊃	\subseteq	⊆	\subset	⊂
\int	∫	\in	∋	\o	o

Table A.8 MATLAB Functions

Function Name	Function Description
abs	Computes the absolute value
acos	Computes the arccosine
ans	Variable created for expressions
asin	Computes the arcsine
atan	Computes the arctangent (2 quadrant)
atan2	Computes the arctangent (4 quadrant)
axis	Specifics the manual axis scaling on plots
bode	Generates Bode frequency response plots
c2d	Converts a continuous-time state variable system representation to a discrete-time system representation
clear	Clears the workspace
clf	Clears the graph window
conj	Computes the complex conjugate
conv	Multiplies two polynomials (convolution)
cos	Computes the cosine
ctrb	Computes the controllability matrix
diary	Saves the session in a disk file
d2c	Converts a discrete-time state variable system representation to a continuous-time system representation
eig	Computes the eigenvalues and eigenvectors
end	Terminates control structures
exp	Computes the exponential with base e
expm	Computes the matrix exponential with base e
eye	Generates an identity matrix
feedback	Computes the feedback interconnection of two systems
for	Generates a loop
format	Sets the output display format
grid on	Adds a grid to the current graph
help	Prints a list of HELP topics
hold on	Holds the current graph on the screen
i	$\sqrt{-1}$
imag	Computes the imaginary part of a complex number
impulse	Computes the unit impulse response of a system
inf	Represents infinity
j	$\sqrt{-1}$
legend	Puts a legend on the current plot
linspace	Generates linearly spaced vectors
load	Loads variables saved in a file
log	Computes the natural logarithm
log10	Computes the logarithm base 10
loglog	Generates log-log plots
logspace	Generates logarithmically spaced vectors
lsim	Computes the time response of a system to an arbitrary input and initial conditions
margin	Computes the gain margin, phase margin, and associated crossover frequencies from frequency response data
max	Determines the maximum value
mesh	Creates three-dimensional mesh surfaces
meshgrid	Generates arrays for use with the mesh function
min	Determines the minimum value
minreal	Transfer function pole–zero cancellation

Table A.8 continued

Table A.8 *Continued*

Function Name	Function Description
NaN	Representation for Not-a-Number
ngrid	Draws grid lines on a Nichols chart
nichols	Computes a Nichols frequency response plot
num2str	Converts numbers to strings
nyquist	Calculates the Nyquist frequency response
obsv	Computes the observability matrix
ones	Generates a matrix of integers where all the integers are 1
pade	Computes an nth-order Padé approximation to a time delay
parallel	Computes a parallel system connection
plot	Generates a linear plot
pole	Computes the poles of a system
poly	Computes a polynomial from roots
polyval	Evaluates a polynomial
printsys	Prints state variable and transfer function representations of linear systems in a pretty form
pzmap	Plots the pole–zero map of a linear system
rank	Calculates the rank of a matrix
real	Computes the real part of a complex number
residue	Computes a partial fraction expansion
rlocfind	Finds the gain associated with a given set of roots on a root locus plot
rlocus	Computes the root locus
roots	Determines the roots of a polynomial
semilogx	Generates an x-y plot using semilog scales with the x-axis $\log_{10}$ and the y-axis linear
semilogy	Generates an x-y plot using semilog scales with the y-axis $\log_{10}$ and the x-axis linear
series	Computes a series system connection
shg	Shows graph window
sin	Computes the sine
sqrt	Computes the square root
ss	Creates a state-space model object
step	Calculates the unit step response of a system
subplot	Splits the graph window into subwindows
tan	Computes the tangent
text	Adds text to the current graph
title	Adds a title to the current graph
tf	Creates a transfer function model object
who	Lists the variables currently in memory
whos	Lists the current variables and sizes
xlabel	Adds a label to the x-axis of the current graph
ylabel	Adds a label to the y-axis of the current graph
zero	Computes the zeros of a system
zeros	Generates a matrix of zeros

MATLAB BASICS: PROBLEMS

A.1 Consider the two matrices

$$A = \begin{bmatrix} 4 & 2\pi \\ 6j & 10 + \sqrt{2}j \end{bmatrix}$$

$$B = \begin{bmatrix} 6j & -13\pi \\ \pi & 16 \end{bmatrix}.$$

Using MATLAB, compute the following:

(a) $\boldsymbol{A} + \boldsymbol{B}$ (b) $\boldsymbol{AB}$
(c) $\boldsymbol{A}^2$ (d) $\boldsymbol{A}'$
(e) $\boldsymbol{B}^{-1}$ (f) $\boldsymbol{B}'\boldsymbol{A}'$
(g) $\boldsymbol{A}^2 + \boldsymbol{B}^2 - \boldsymbol{AB}$

A.2 Consider the following set of linear algebraic equations:

$$5x + 6y + 10z = 4,$$
$$-3x + 14z = 10,$$
$$-7y + 21z = 0.$$

Determine the values of x, y, and z so that the set of algebraic equations is satisfied. (*Hint:* Write the equations in matrix vector form.)

A.3 Generate a plot of

$$y(x) = e^{-0.5x} \sin \omega x,$$

where $\omega = 10$ rad/s, and $0 \leq x \leq 10$. Utilize the colon notation to generate the x vector in increments of 0.1.

A.4 Develop a MATLAB script to plot the function

$$y(x) = \frac{4}{\pi} \cos \omega x + \frac{4}{9\pi} \cos 3\omega x.$$

where ω is a variable input at the command prompt. Label the x-axis with *time (sec)* and the y-axis with $y(x) = (4/\pi) * \cos(\omega x) + (4/9\pi) * \cos(3\omega x)$. Include a descriptive header in the script, and verify that the help function will display the header. Choose $\omega = 1, 3, 10$ rad/s and test the script.

A.5 Consider the function

$$y(x) = 10 + 5e^{-x} \cos(\omega x + 0.5).$$

Develop a script to co-plot $y(x)$ for the three values of $\omega = 1, 3, 10$ rad/s with $0 \leq x \leq 5$ seconds. The final plot should have the following attributes:

Title	$y(x) = 10 + 5 \exp(-x) * \cos(\omega x + 0.5)$
x-axis label	Time (sec)
y-axis label	$y(x)$
Line type	$\omega = 1$: solid line
	$\omega = 3$: dashed line
	$\omega = 10$: dotted line
Grid	grid on