The McGraw-Hill Companies

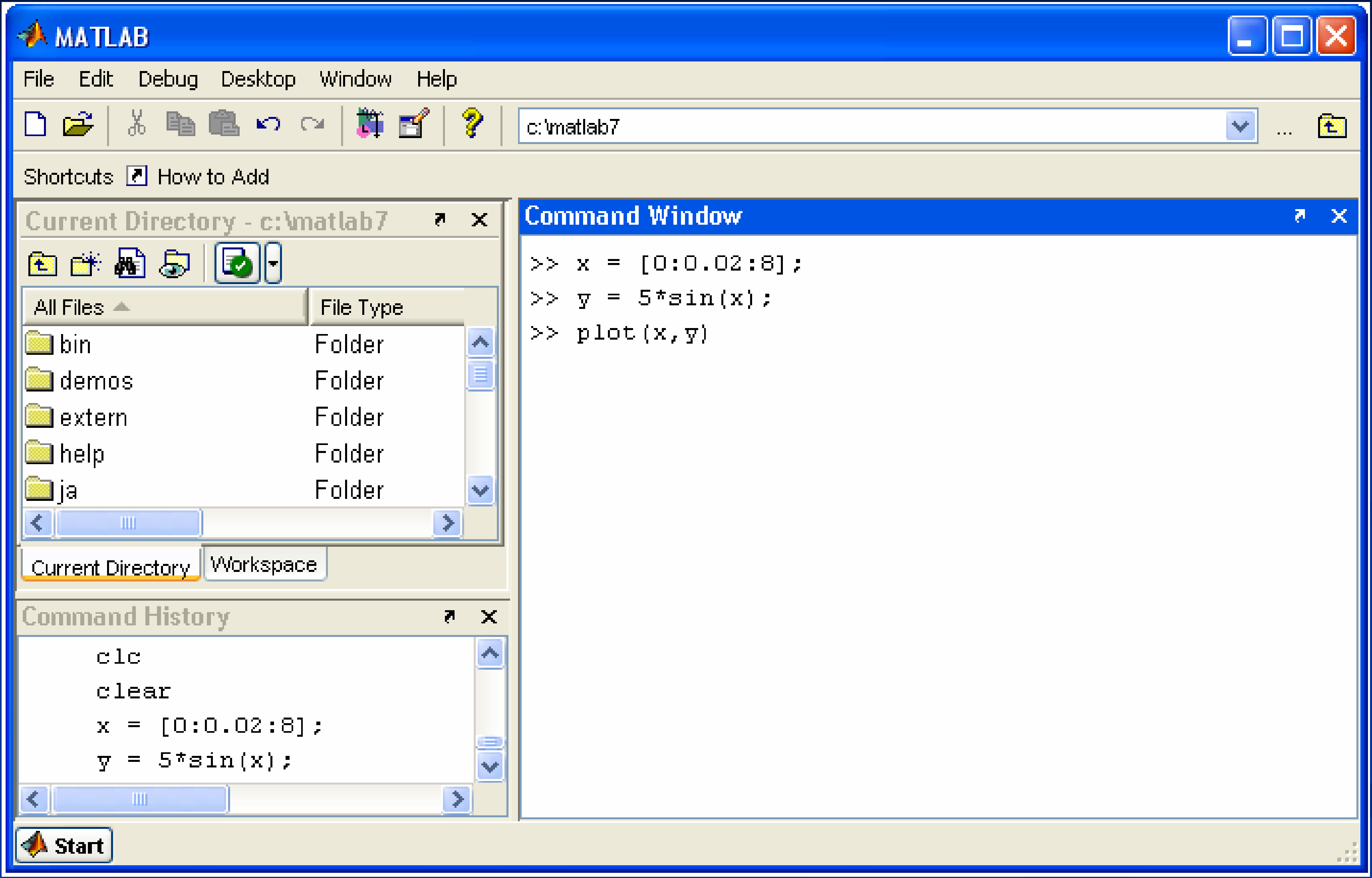
PowerPoint to accompany

Introduction to MATLAB 7 for Engineers

William J. Palm III

Chapter 1
An Overview Of MATLAB

The default MATLAB Desktop. Figure 1.1–1



All Files	File Type
bin	Folder
demos	Folder
extern	Folder
help	Folder
ja	Folder

Command History:

```
clc
clear
x = [0:0.02:8];
y = 5*sin(x);
```

Command Window:

```
>> x = [0:0.02:8];
>> y = 5*sin(x);
>> plot(x,y)
```

More? See pages 6-7.

Entering Commands and Expressions

- MATLAB retains your previous keystrokes.
- Use the up-arrow key to scroll back back through the commands.
- Press the key once to see the previous entry, and so on.
- Use the down-arrow key to scroll forward. Edit a line using the left- and right-arrow keys the **Backspace** key, and the **Delete** key.
- Press the **Enter** key to execute the command.

An Example Session

```
>> 8/10
ans =
     0.8000
>> 5*ans
ans =
     4
>> r=8/10
r =
   0.8000
>> r
r =
   0.8000
>> s=20*r
s =
   16
```

More? See pages 8-9.

Scalar Arithmetic Operations Table 1.1–1

Symbol	Operation	MATLAB form
^	exponentiation: a^b	`a^b`
*	multiplication: ab	`a*b`
/	right division: a/b	`a/b`
\	left division: b/a	`a\b`
+	addition: $a + b$	`a + b`
-	subtraction: $a - b$	`a - b`

Order of Precedence Table 1.1–2

Precedence	Operation
First	Parentheses, evaluated starting with the innermost pair.
Second	Exponentiation, evaluated from left to right.
Third	Multiplication and division with equal precedence, evaluated from left to right.
Fourth	Addition and subtraction with equal precedence, evaluated from left to right.

Examples of Precedence

```
>> 8 + 3*5
ans =
     23
>> 8 + (3*5)
ans =
     23
>>(8 + 3)*5
ans =
     55
>>4^2-12- 8/4*2
ans =
     0
>>4^2-12- 8/(4*2)
ans =
     3
```

(continued …)

Examples of Precedence (continued)

```
>> 3*4^2 + 5
ans =
     53
>>(3*4)^2 + 5
ans =
     149
>>27^(1/3) + 32^(0.2)
ans =
     5
>>27^(1/3) + 32^0.2
ans =
     5
>>27^1/3 + 32^0.2
ans =
     11
```

The Assignment Operator `=`

- Typing `x = 3` assigns the value 3 to the variable `x`.
- We can then type `x = x + 2`. This assigns the value 3 + 2 = 5 to `x`. But in algebra this implies that 0 = 2.
- In algebra we can write x + 2 = 20, but in MATLAB we cannot.
- In MATLAB the left side of the = operator must be a single variable.
- The right side must be a *computable* value.

More? See pages 11-12.

Commands for managing the work session Table 1.1–3

Command	Description
`clc`	Clears the Command window.
`clear`	Removes all variables from memory.
`clear v1 v2`	Removes the variables `v1` and `v2` from memory.
`exist('var')`	Determines if a file or variable exists having the name '`var`'.
`quit`	Stops MATLAB.

(continued …)

Commands for managing the work session
Table 1.1–3 (continued)

`who`	Lists the variables currently in memory.
`whos`	Lists the current variables and sizes, and indicates if they have imaginary parts.
`:`	Colon; generates an array having regularly spaced elements.
`,`	Comma; separates elements of an array.
`;`	Semicolon; suppresses screen printing; also denotes a new row in an array.
`...`	Ellipsis; continues a line.

More? See pages 12-15.

Special Variables and Constants Table 1.1–4

Command	**Description**
`ans`	Temporary variable containing the most recent answer.
`eps`	Specifies the accuracy of floating point precision.
`i,j`	The imaginary unit $\sqrt{-1}$.
`Inf`	Infinity.
`NaN`	Indicates an undefined numerical result.
`pi`	The number π.

Complex Number Operations

- The number $c_1 = 1 - 2i$ is entered as follows:
  `c1 = 1-2i`.

- An asterisk is not needed between i or j and a number, although it is required with a variable, such as `c2 = 5 - i*c1`.

- Be careful. The expressions

```
y = 7/2*i
```

and

```
x = 7/2i
```

give two different results: $y = (7/2)i = 3.5i$ and $x = 7/(2i) = -3.5i$.

Numeric Display Formats Table 1.1–5

Command	Description and Example
`format short`	Four decimal digits (the default); 13.6745.
`format long`	16 digits; 17.27484029463547.
`format short e`	Five digits (four decimals) plus exponent; 6.3792e+03.
`format long e`	16 digits (15 decimals) plus exponent; 6.379243784781294e–04.

Arrays

- The numbers 0, 0.1, 0.2, …, 10 can be assigned to the variable u by typing `u = [0:0.1:10]`.
- To compute $w = 5 \sin u$ for $u = 0, 0.1, 0.2, \ldots, 10$, the session is;

```
>>u = [0:0.1:10];
 >>w = 5*sin(u);
```

- The single line, `w = 5*sin(u)`, computed the formula $w = 5 \sin u$ 101 times.

Array Index

```
>>u(7)
ans =
     0.6000
>>w(7)
ans =
     2.8232
```

- Use the `length` function to determine how many values are in an array.

```
>>m = length(w)
m =
   101
```

More? See pages 19-20.

Polynomial Roots

To find the roots of $x^3 - 7x^2 + 40x - 34 = 0$, the session is

```
>>a = [1,-7,40,-34];
>>roots(a)
ans =
     3.0000 + 5.000i
     3.0000 - 5.000i
     1.0000
```

The roots are $x = 1$ and $x = 3 \pm 5i$.

Some Commonly Used Mathematical Functions Table 1.3–1

Function	MATLAB syntax[1]
e^x	`exp(x)`
$\sqrt{x}$	`sqrt(x)`
$\ln x$	`log(x)`
$\log_{10} x$	`log10(x)`
$\cos x$	`cos(x)`
$\sin x$	`sin(x)`
$\tan x$	`tan(x)`
$\cos^{-1} x$	`acos(x)`
$\sin^{-1} x$	`asin(x)`
$\tan^{-1} x$	`atan(x)`

[1]The MATLAB trigonometric functions use radian measure.

When you type `problem1`,

1. MATLAB first checks to see if problem1 is a variable and if so, displays its value.
2. If not, MATLAB then checks to see if problem1 is one of its own commands, and executes it if it is.
3. If not, MATLAB then looks in the current directory for a file named `problem1.m` and executes `problem1` if it finds it.
4. If not, MATLAB then searches the directories in its search path, in order, for `problem1.m` and then executes it if found.

More? See pages 22-24.

When you type `problem1`,

1. MATLAB first checks to see if `problem1` is a variable and if so, displays its value.
2. If not, MATLAB then checks to see if `problem1` is one of its own commands, and executes it if it is.
3. If not, MATLAB then looks in the current directory for a file named `problem1.m` and executes `problem1` if it finds it.
4. If not, MATLAB then searches the directories in its search path, in order, for `problem1.m` and then executes it if found.

System, Directory, and File Commands Table 1.3–2

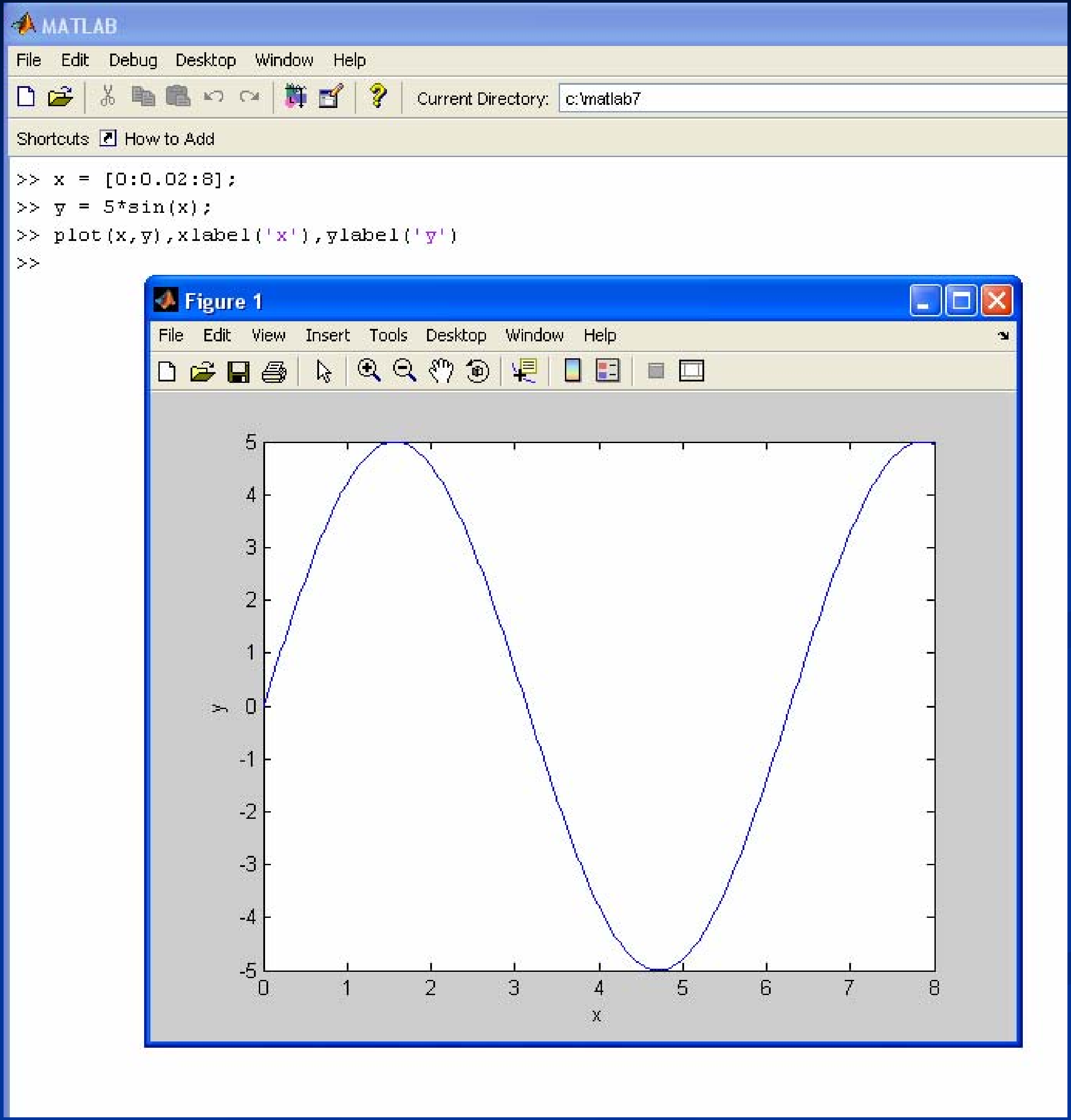
Command	Description
`addpath dirname`	Adds the directory `dirname` to the search path.
`cd dirname`	Changes the current directory to `dirname`.
`dir`	Lists all files in the current directory.
`dir dirname`	Lists all the files in the directory `dirname`.
`path`	Displays the MATLAB search path.
`pathtool`	Starts the Set Path tool.

(continued …)

System, Directory, and File Commands Table 1.3–2 (continued)

Command	**Description**
`pwd`	Displays the current directory.
`rmpath dirname`	Removes the directory `dirname` from the search path.
`what`	Lists the MATLAB-specific files found in the current working directory. Most data files and other non-MATLAB files are not listed. Use dir to get a list of all files.
`what dirname`	Lists the MATLAB-specific files in directory `dirname`.

A graphics window showing a plot. Figure 1.3–1


MATLAB
File Edit Debug Desktop Window Help
Current Directory: c:\matlab7
Shortcuts How to Add
Figure 1
File Edit View Insert Tools Desktop Window Help
x
y
0 1 2 3 4 5 6 7 8
-5 -4 -3 -2 -1 0 1 2 3 4 5

```
>> x = [0:0.02:8];
>> y = 5*sin(x);
>> plot(x,y),xlabel('x'),ylabel('y')
>>
```

Some MATLAB plotting commands Table 1.3–3

Command	Description
`[x,y] = ginput(n)`	Enables the mouse to get *n* points from a plot, and returns the *x* and *y* coordinates in the vectors x and y, which have a length *n*.
`grid`	Puts grid lines on the plot.
`gtext('text')`	Enables placement of text with the mouse.

(continued …)

Some MATLAB plotting commands Table 1.3–3 (continued)

`plot(x,y)`	Generates a plot of the array y versus the array x on rectilinear axes.
`title('text')`	Puts text in a title at the top of the plot.
`xlabel('text')`	Adds a text label to the horizontal axis (the abscissa).
`ylabel('text')`	Adds a text label to the vertical axis (the ordinate).

More? See pages 25-27.

Solution of Linear Algebraic Equations

$6x + 12y + 4z = 70$
$7x - 2y + 3z = 5$
$2x + 8y - 9z = 64$

```
>>A = [6,12,4;7,-2,3;2,8,-9];
>>B = [70;5;64];
>>Solution = A\B
Solution =
     3
     5
    -2
```

The solution is $x = 3$, $y = 5$, and $z = -2$.

More? See page 28.

You can perform operations in MATLAB in two ways:

1. In the interactive mode, in which all commands are entered directly in the Command window, or
2. By running a MATLAB program stored in *script* file.

   This type of file contains MATLAB commands, so running it is equivalent to typing all the commands—one at a time—at the Command window prompt.

   You can run the file by typing its name at the Command window prompt.

More? See pages 29-32.

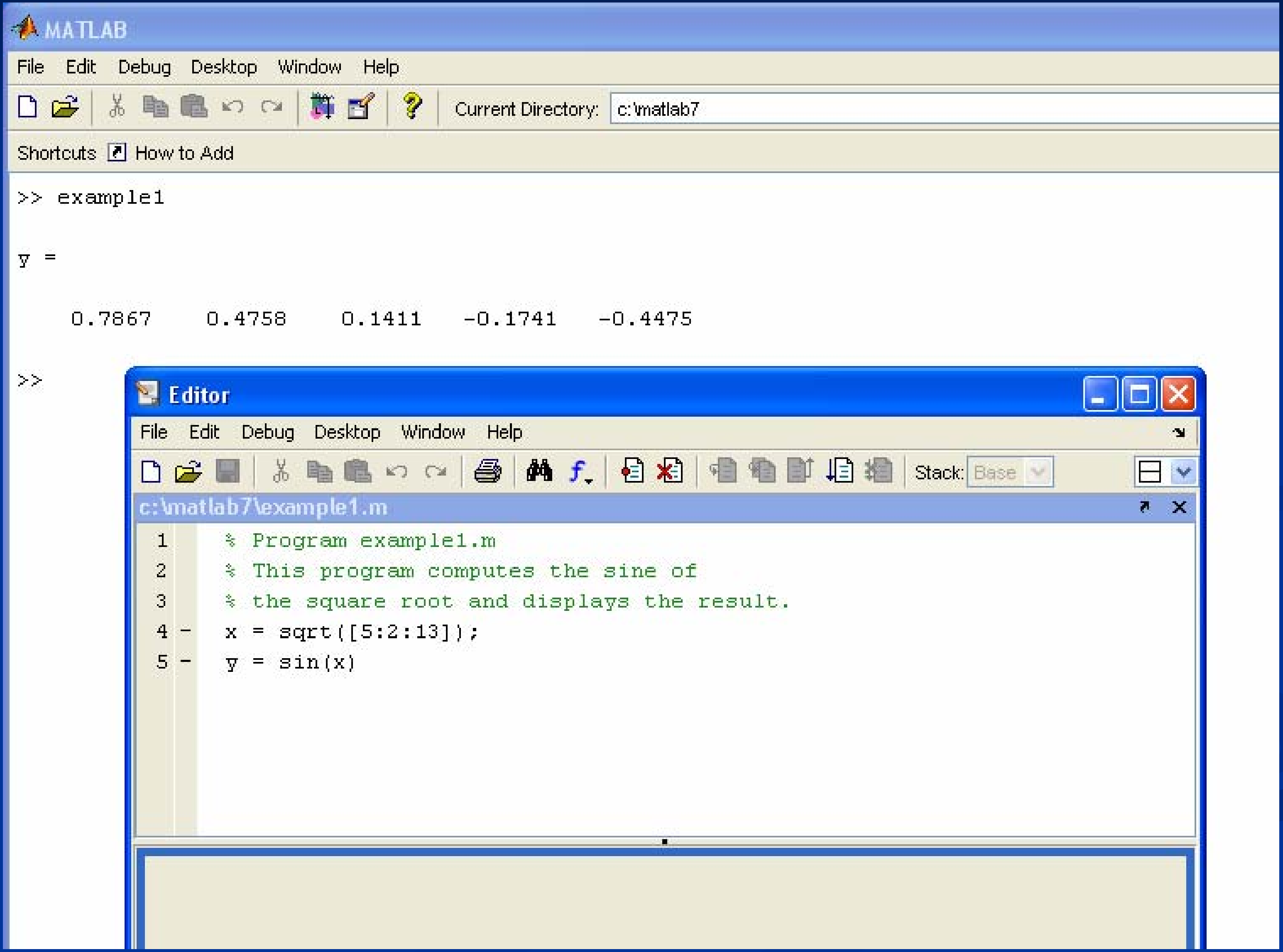
COMMENTS

The comment symbol may be put anywhere in the line. MATLAB ignores everything to the right of the % symbol. For example,

```
>>% This is a comment.
>>x = 2+3 % So is this.
x =
   5
```

Note that the portion of the line before the % sign is executed to compute x.

The MATLAB Command window with the Editor/Debugger open. Figure 1.4–1


MATLAB
File Edit Debug Desktop Window Help
Current Directory: c:\matlab7
Shortcuts How to Add
Editor
File Edit Debug Desktop Window Help
Stack: Base
c:\matlab7\example1.m

```
>> example1

y =

    0.7867    0.4758    0.1411   -0.1741   -0.4475

>>
```

```
% Program example1.m
% This program computes the sine of
% the square root and displays the result.
x = sqrt([5:2:13]);
y = sin(x)
```

Keep in mind when using script files:

1. The name of a script file must begin with a letter, and may include digits and the underscore character, up to 31 characters.

2. Do not give a script file the same name as a variable.

3. Do not give a script file the same name as a MATLAB command or function. You can check to see if a command, function or file name already exists by using the exist command.

Debugging Script Files

Program errors usually fall into one of the following categories.

1. Syntax errors such as omitting a parenthesis or comma, or spelling a command name incorrectly. MATLAB usually detects the more obvious errors and displays a message describing the error and its location.
2. Errors due to an incorrect mathematical procedure, called *runtime errors*. Their occurrence often depends on the particular input data. A common example is division by zero.

To locate program errors, try the following:

1. Test your program with a simple version of the problem which can be checked by hand.
2. Display any intermediate calculations by removing semicolons at the end of statements.
3. Use the debugging features of the Editor/Debugger.

More? See pages 32-33.

Programming Style

1. *Comments section*

a. The name of the program and any key words in the first line.

b. The date created, and the creators' names in the second line.

c. The definitions of the variable names for every input and output variable. Include definitions of variables used in the calculations and *units of measurement for all input and all output variables!*

d. The name of every user-defined function called by the program.

(continued …)

Programming Style (continued)

2. *Input section* Include input data and/or the input functions and comments for documentation.

3. *Calculation section*

4. *Output section* This section might contain functions for displaying the output on the screen.

More? See pages 34-37.

Input/output commands Table 1.4–2

Command	Description
`disp(A)`	Displays the contents, but not the name, of the array `A`.
`disp('text')`	Displays the text string enclosed within quotes.
`x = input('text')`	Displays the text in quotes, waits for user input from the keyboard, and stores the value in `x`.
`x = input('text','s')`	Displays the text in quotes, waits for user input from the keyboard, and stores the input as a string in `x`.

Example of a Script File

Problem:

The speed v of a falling object dropped with no initial velocity is given as a function of time t by $v = gt$.

Plot v as a function of t for $0 \leq t \leq t_f$, where t_f is the final time entered by the user.

(continued …)

Example of a Script File (continued)

```
% Program falling_speed.m:
% Plots speed of a falling object.
% Created on March 1, 2004 by W. Palm
%
% Input Variable:
% tf = final time (in seconds)
%
% Output Variables:
% t = array of times at which speed is
% computed (in seconds)
% v = array of speeds (meters/second)
%
```

(continued …)

Example of a Script File (continued)

```
% Parameter Value:
g = 9.81; % Acceleration in SI units
%
% Input section:
tf = input('Enter final time in seconds:');
%
```

(continued …)

Example of a Script File (continued)

```
% Calculation section:
dt = tf/500;
% Create an array of 501 time values.
t = [0:dt:tf];
% Compute speed values.
v = g*t;
%
% Output section:
Plot(t,v),xlabel('t (s)'),ylabel('v m/s)')
```

More? See pages 37-38.

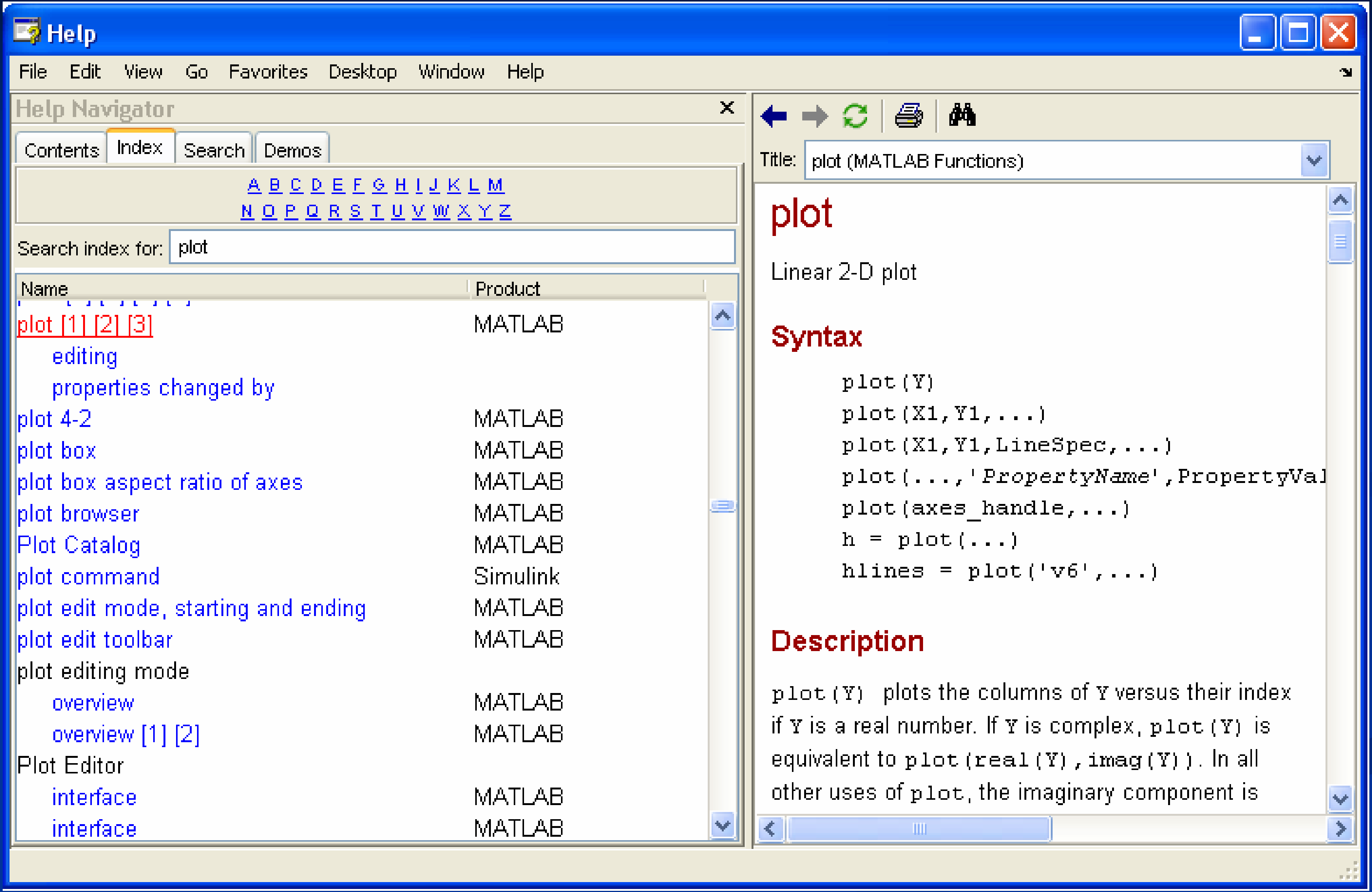
Getting Help

- Throughout each chapter margin notes identify where key terms are introduced.
- Each chapter contains tables summarizing the MATLAB commands introduced in that chapter.
- At the end of each chapter is a summary guide to the commands covered in that chapter.
- Appendix A contains tables of MATLAB commands, grouped by category, with the appropriate page references.
- There are three indexes. The first lists MATLAB commands and symbols, the second lists Simulink blocks, and the third lists topics.

The Help Navigator contains four tabs:

- *Contents:* a contents listing tab,
- *Index:* a global index tab,
- *Search:* a search tab having a find function and full text search features, and
- *Demos:* a bookmarking tab to start built-in demonstrations.

The MATLAB Help Browser. Figure 1.5–1


Help
File Edit View Go Favorites Desktop Window Help
Help Navigator
Contents Index Search Demos
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
Search index for: plot
Title: plot (MATLAB Functions)

Name	Product
plot [1] [2] [3]	MATLAB
editing	
properties changed by	
plot 4-2	MATLAB
plot box	MATLAB
plot box aspect ratio of axes	MATLAB
plot browser	MATLAB
Plot Catalog	MATLAB
plot command	Simulink
plot edit mode, starting and ending	MATLAB
plot edit toolbar	MATLAB
plot editing mode	
overview	MATLAB
overview [1] [2]	MATLAB
Plot Editor	
interface	MATLAB
interface	MATLAB

plot

Linear 2-D plot

Syntax

```
plot(Y)
plot(X1,Y1,...)
plot(X1,Y1,LineSpec,...)
plot(...,'PropertyName',PropertyVal
plot(axes_handle,...)
h = plot(...)
hlines = plot('v6',...)
```

Description

`plot(Y)` plots the columns of `Y` versus their index if `Y` is a real number. If `Y` is complex, `plot(Y)` is equivalent to `plot(real(Y),imag(Y))`. In all other uses of `plot`, the imaginary component is

Help Functions

- `help funcname`: Displays in the Command window a description of the specified function `funcname`.
- `lookfor topic`: Displays in the Command window a brief description for all functions whose description includes the specified key word `topic`.
- `doc funcname`: Opens the Help Browser to the reference page for the specified function `funcname`, providing a description, additional remarks, and examples.

More? See pages 38-43.

Relational operators Table 1.6–1

Relational operator	Meaning
<	Less than.
<=	Less than or equal to.
>	Greater than.
>=	Greater than or equal to.
==	Equal to.
~=	Not equal to.

Examples of Relational Operators

```
>> x = [6,3,9]; y = [14,2,9];
>> z = (x < y)
z =
  1 0 0
>>z = ( x > y)
z =
  0 1 0
>>z = (x ~= y)
z =
  1 1 0
>>z = ( x == y)
z =
  0 0 1
>>z = (x > 8)
z =
  0 0 1
```

More? See pages 44-45.

The `find` Function

`find(x)` computes an array containing the indices of the *nonzero* elements of the numeric array `x`. For example

```
>>x = [-2, 0, 4];
>>y = find(x)
Y =
  1    3
```

The resulting array `y = [1, 3]` indicates that the first and third elements of x are nonzero.

Note the difference between the result obtained by `x(x<y)` and the result obtained by `find(x<y)`.

```
>>x = [6,3,9,11];y = [14,2,9,13];
>>values = x(x<y)
values =
     6    11
>>how_many = length(values)
how_many =
    2
>>indices = find(x<y)
indices =
    1  4
```

More? See pages 45-46.

The `if` Statement

The general form of the `if` statement is

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

The `else` and `elseif` statements may be omitted if not required.

Suppose that we want to compute *y* such that

$$y = \begin{cases} 15\sqrt{4x} + 10 & \text{if } x \geq 9 \\ 10x + 10 & \text{if } 0 \leq x < 9 \\ 10 & \text{if } x < 0 \end{cases}$$

The following statements will compute *y*, assuming that the variable `x` already has a scalar value.

```
if x >= 9
   y = 15*sqrt(4x) + 10
elseif x >= 0
   y = 10*x + 10
else
   y = 10
end
```

Note that the `elseif` statement does not require a separate `end` statement.

More? See pages 47-48.

Loops

There are two types of explicit loops in MATLAB;

• the `for` loop, used when the number of passes is known ahead of time, and

• the `while` loop, used when the looping process must terminate when a specified condition is satisfied, and thus the number of passes is not known in advance.

A simple example of a `for` loop is

```
m = 0;
x(1) = 10;
for k = 2:3:11
   m = m+1;
   x(m+1) = x(m) + k^2;
end
```

k takes on the values 2, 5, 8, 11. The variable `m` indicates the index of the array `x`. When the loop is finished the array `x` will have the values `x(1)=14,x(2)=39,x(3)=103,x(4)=224.`

A simple example of a `while` loop is

```
x = 5;k = 0;
while  x < 25
   k = k + 1;
   y(k) = 3*x;
   x = 2*x-1;
end
```

The loop variable `x` is initially assigned the value 5, and it keeps this value until the statement `x = 2*x - 1` is encountered the first time. Its value then changes to 9. Before each pass through the loop, `x` is checked to see if its value is less than 25. If so, the pass is made. If not, the loop is skipped.

Example of a `for` Loop

Write a script file to compute the sum of the first 15 terms in the series $5k^2 - 2k$, $k = 1, 2, 3, \ldots, 15$.

```
total = 0;
for k = 1:15
    total = 5*k^2 - 2*k + total;
end
disp('The sum for 15 terms is:')
disp(total)
```

The answer is 5960.

Example of a `for` Loop

Write a script file to determine how many terms are required for the sum of the series $5k^2 - 2k$, $k = 1, 2, 3, \ldots$ to exceed 10,000. What is the sum for this many terms?

```
total = 0;k = 0;
while total < 1e+4
   k = k + 1;
   total = 5*k^2 - 2*k + total;
end
disp('The number of terms is:')
disp(k)
disp('The sum is:')
disp(total)
```

The sum is 10,203 after 18 terms.

Example of a `while` Loop

Determine how long it will take to accumulate at least $10,000 in a bank account if you deposit $500 initially and $500 at the end of each year, if the account pays 5 percent annual interest.

```
amount = 500; k=0;
while amount < 10000
   k = k+1;
   amount = amount*1.05 + 500;
end
amount
k
```

The final results are amount = 1.0789e+004, or $10,789, and k = 14, or 14 years.

More? See pages 48-51.

Steps in engineering problem solving Table 1.7–1

1. Understand the purpose of the problem.
2. Collect the known information. Realize that some of it might later be found unnecessary.
3. Determine what information you must find.
4. Simplify the problem only enough to obtain the required information. State any assumptions you make.
5. Draw a sketch and label any necessary variables.
6. Determine which fundamental principles are applicable.
7. Think generally about your proposed solution approach and consider other approaches before proceeding with the details.

(continued …)

Steps in engineering problem solving Table 1.7–1 **(continued)**

8. Label each step in the solution process. **U**nderstand the purpose of the problem

9. If you solve the problem with a program, hand check the results using a simple version of the problem.

Checking the dimensions and units and printing the results of intermediate steps in the calculation sequence can uncover mistakes.

(continued …)

Steps in engineering problem solving Table 1.7–1 **(continued)**

10. Perform a “reality check” on your answer. Does it make sense? Estimate the range of the expected result and compare it with your answer. Do not state the answer with greater precision than is justified by any of the following:

(a) The precision of the given information.

(b) The simplifying assumptions.

(c) The requirements of the problem.

Interpret the mathematics. If the mathematics produces multiple answers, do not discard some of them without considering what they mean. The mathematics might be trying to tell you something, and you might miss an opportunity to discover more about the problem.

More? See pages 52-56.

Steps for developing a computer solution Table 1.7–2

1. State the problem concisely.
2. Specify the data to be used by the program. This is the “input.”
3. Specify the information to be generated by the program. This is the “output.”
4. Work through the solution steps by hand or with a calculator; use a simpler set of data if necessary.
5. Write and run the program.
6. Check the output of the program with your hand solution.
7. Run the program with your input data and perform a reality check on the output.
8. If you will use the program as a general tool in the future, test it by running it for a range of reasonable data values; perform a reality check on the results.

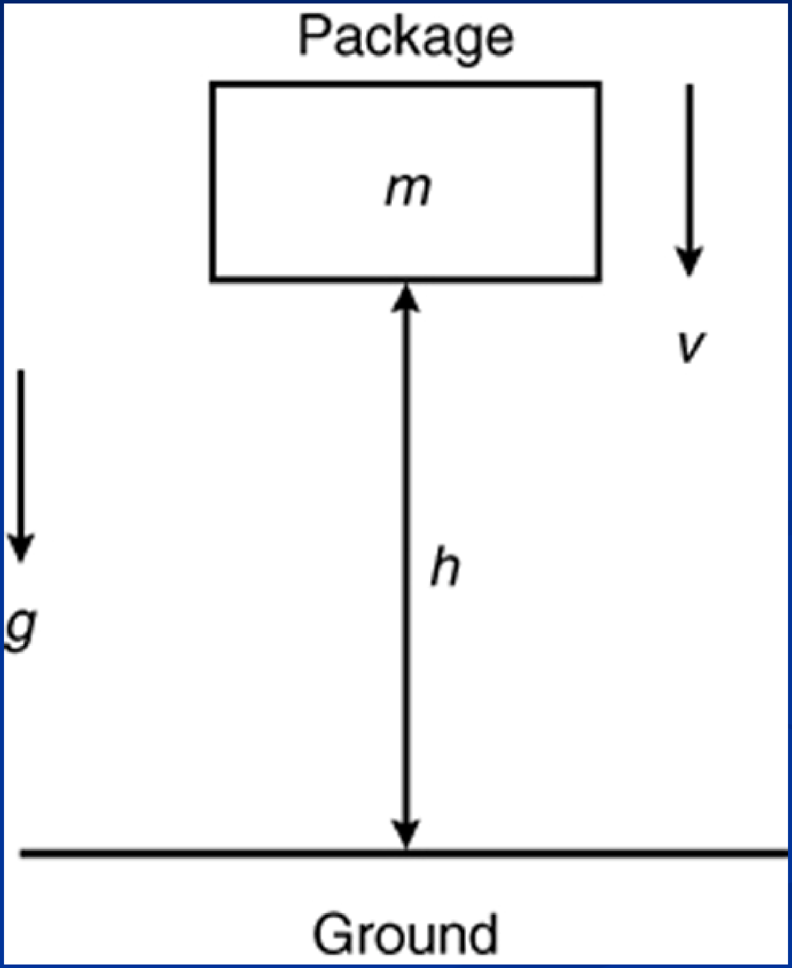
More? See pages 56-60.

Key Terms with Page References

The following slides contain figures from the chapter and its homework problems.

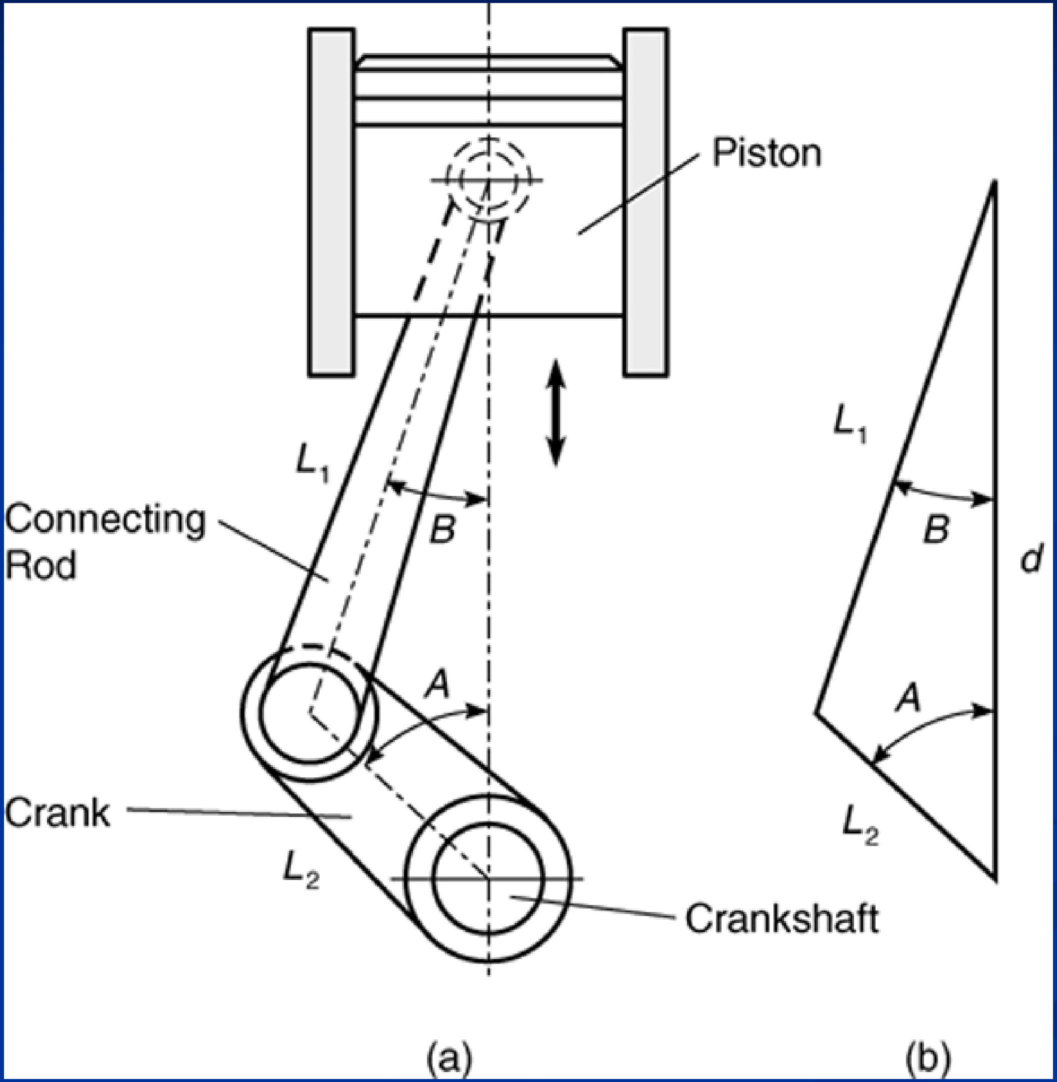
Sketch of the dropped-package problem.

Figure 1.7–1



A piston, connecting rod, and crank for an internal combustion engine.

Figure 1.7–2



Plot of the piston motion versus crank angle.

Figure 1.7–3

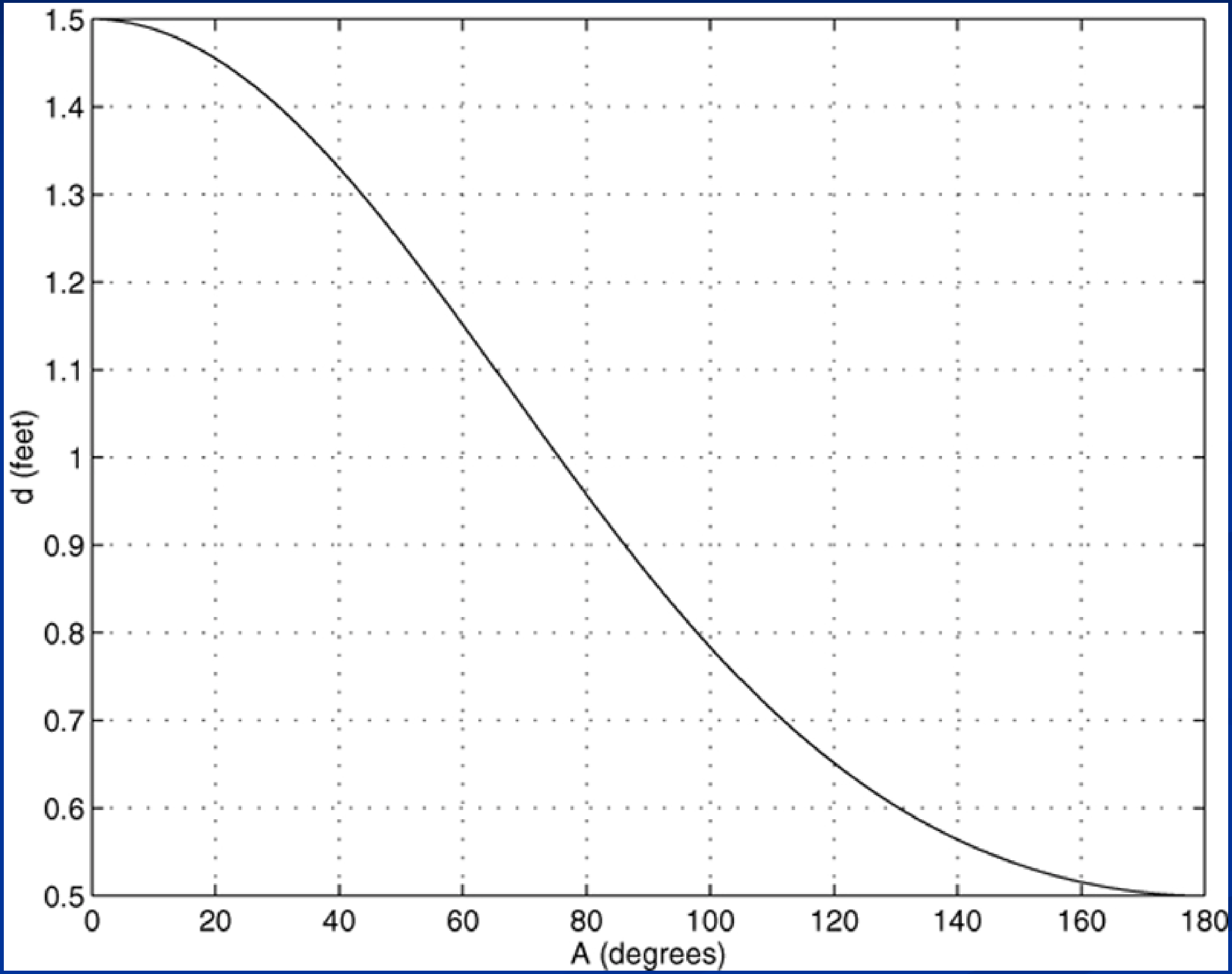


Figure P28

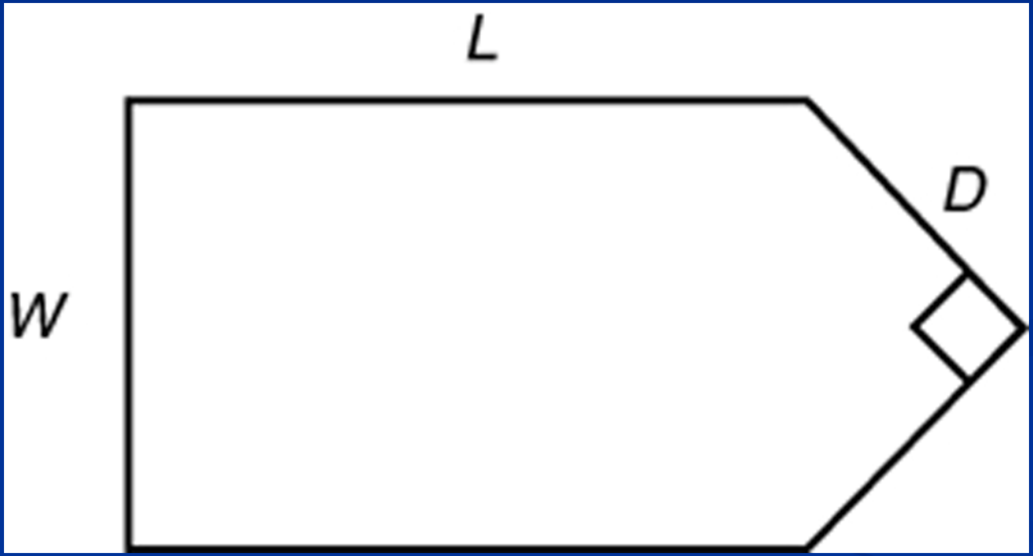


Figure P29

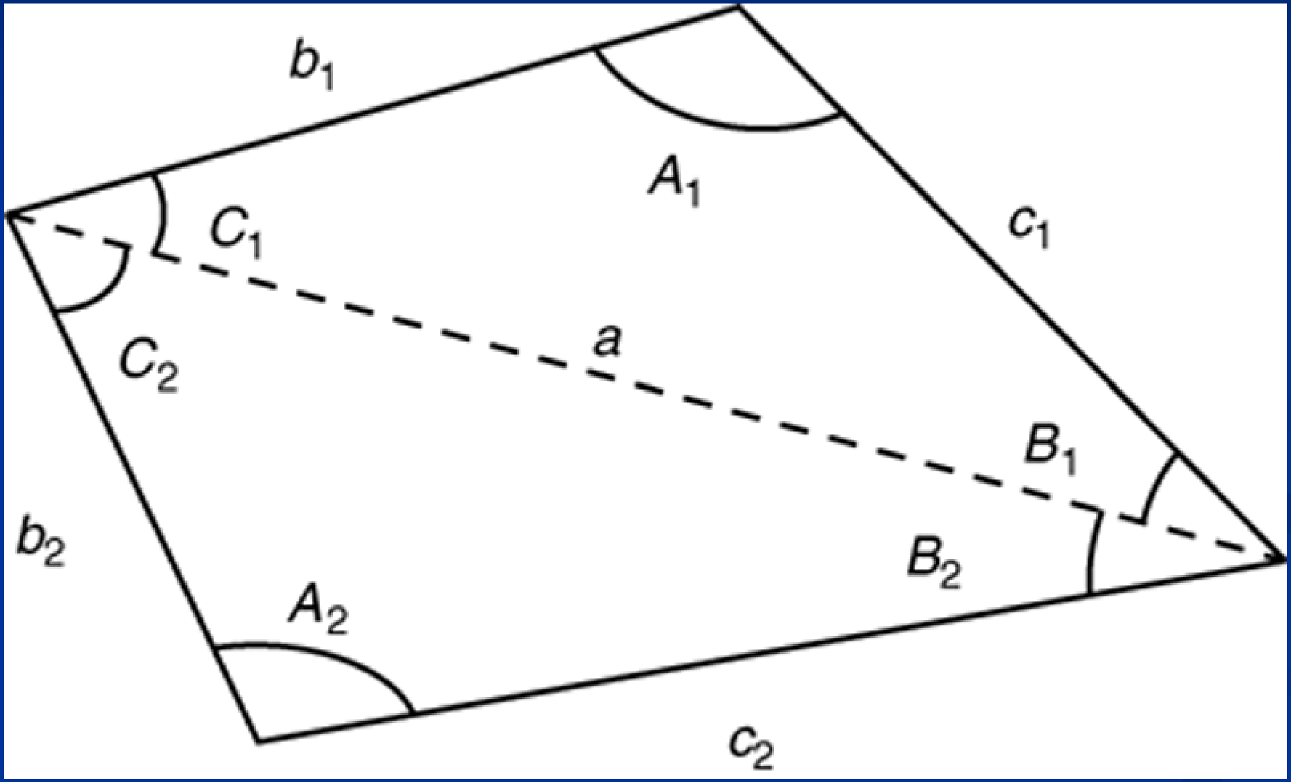


Figure P30