

1.1 The Advantages of MATLAB

MATLAB has many advantages compared to conventional computer languages for technical problem solving. Among them are:

1. Ease of Use

MATLAB is an interpreted language, like many versions of Basic. Like Basic, it is very easy to use. The program can be used as a scratch pad to evaluate expressions typed at the command line, or it can be used to execute large pre-written programs. Programs may be easily written and modified with the built-in integrated development environment and debugged with the MATLAB debugger. Because the language is so easy to use, it is ideal for the rapid prototyping of new programs.

Many program development tools are provided to make the program easy to use. They include an integrated editor/debugger, on-line documentation and manuals, a workspace browser, and extensive demos.

2. Platform Independence

MATLAB is supported on many different computer systems, providing a large measure of platform independence. At the time of this writing, the language is supported on Windows NT/2000/XP, Linux, several versions of Unix, and the Macintosh. Programs written on any platform will run on all of the other platforms, and data files written on any platform may be read transparently on any other platform. As a result, programs written in MATLAB can migrate to new platforms when the needs of the user change.

3. Predefined Functions

MATLAB comes complete with an extensive library of predefined functions that provide tested and pre-packaged solutions to many basic technical tasks. For example, suppose that you are writing a program that must calculate the statistics associated with an input data set. In most languages, you would need to write your own subroutines or functions to implement calculations such as the arithmetic mean, standard deviation, median, etc. These and hundreds of other functions are built right into the MATLAB language, making your job much easier.

In addition to the large library of functions built into the basic MATLAB language, there are many special-purpose toolboxes available to help solve complex problems in specific areas. For example, a user can buy standard toolboxes to solve problems in signal processing, control systems, communications, image processing, and neural networks, among many others. There is also an extensive collection of free user-contributed MATLAB programs that are shared through the MATLAB Web site.

4. Device-Independent Plotting

Unlike most other computer languages, MATLAB has many integral plotting and imaging commands. The plots and images can be displayed on any graphical output device supported by the computer on which MATLAB is

running. This capability makes MATLAB an outstanding tool for visualizing technical data.

5. Graphical User Interface

MATLAB includes tools that allow a programmer to interactively construct a graphical user interface (GUI) for his or her program. With this capability, the programmer can design sophisticated data-analysis programs that can be operated by relatively inexperienced users.

6. MATLAB Compiler

MATLAB's flexibility and platform independence is achieved by compiling MATLAB programs into a device-independent p-code, and then interpreting the p-code instructions at runtime. This approach is similar to that used by Microsoft's Visual Basic language. Unfortunately, the resulting programs can sometimes execute slowly because the MATLAB code is interpreted rather than compiled. We will point out features that tend to slow program execution when we encounter them.

A separate MATLAB compiler is available. This compiler can compile a MATLAB program into a true executable that runs faster than the interpreted code. It is a great way to convert a prototype MATLAB program into an executable suitable for sale and distribution to users.

1.2 Disadvantages of MATLAB

MATLAB has two principal disadvantages. The first is that it is an interpreted language and therefore can execute more slowly than compiled languages. This problem can be mitigated by properly structuring the MATLAB program, and by the use of the MATLAB compiler to compile the final MATLAB program before distribution and general use.

The second disadvantage is cost: a full copy of MATLAB is five to ten times more expensive than a conventional C or Fortran compiler. This relatively high cost is more than offset by the reduced time required for an engineer or scientist to create a working program, so MATLAB is cost-effective for businesses. However, it is too expensive for most individuals to consider purchasing. Fortunately, there is also an inexpensive Student Edition of MATLAB, which is a great tool for students wishing to learn the language. The Student Edition of MATLAB is essentially identical to the full edition.

1.3 The MATLAB Environment

The fundamental unit of data in any MATLAB program is the **array**. An array is a collection of data values organized into rows and columns and known by a single name. Individual data values within an array may be accessed by including the name of the array followed by subscripts in parentheses that identify the row and

column of the particular value. Even scalars are treated as arrays by MATLAB—they are simply arrays with only one row and one column. We will learn how to create and manipulate MATLAB arrays in Section 1.4.

When MATLAB executes, it can display several types of windows that accept commands or display information. The three most important types of windows are Command Windows, where commands may be entered; Figure Windows, which display plots and graphs; and Edit Windows, which permit a user to create and modify MATLAB programs. We will see examples of all three types of windows in this section.

In addition, MATLAB can display other windows that provide help and that allow the user to examine the values of variables defined in memory. We will examine some of these additional windows here. We will examine the others when we discuss how to debug MATLAB programs.

The MATLAB Desktop

When you start MATLAB Version 7, a special window called the MATLAB desktop appears. The desktop is a window that contains other windows showing MATLAB data, plus toolbars and a “Start” button similar to that used by Windows 2000 or XP. By default, most MATLAB tools are “docked” to the desktop, so that they appear inside the desktop window. However, the user can choose to “undock” any or all tools, making them appear in windows separate from the desktop.

The default configuration of the MATLAB desktop is shown in Figure 1.1. It integrates many tools for managing files, variables, and applications within the MATLAB environment.

The major tools within or accessible from the MATLAB desktop are:

- The Command Window
- The Command History Window
- The Start Button
- The Documents Window, including the Editor/Debugger and Array Editor
- The Figure Windows
- The Workspace Browser
- The Help Browser
- The Path Browser

We will discuss the functions of these tools in later sections of this chapter.

The Command Window

The right hand side of the default MATLAB desktop contains the **Command Window**. A user can enter interactive commands at the command prompt (») in the Command Window, and they will be executed on the spot.

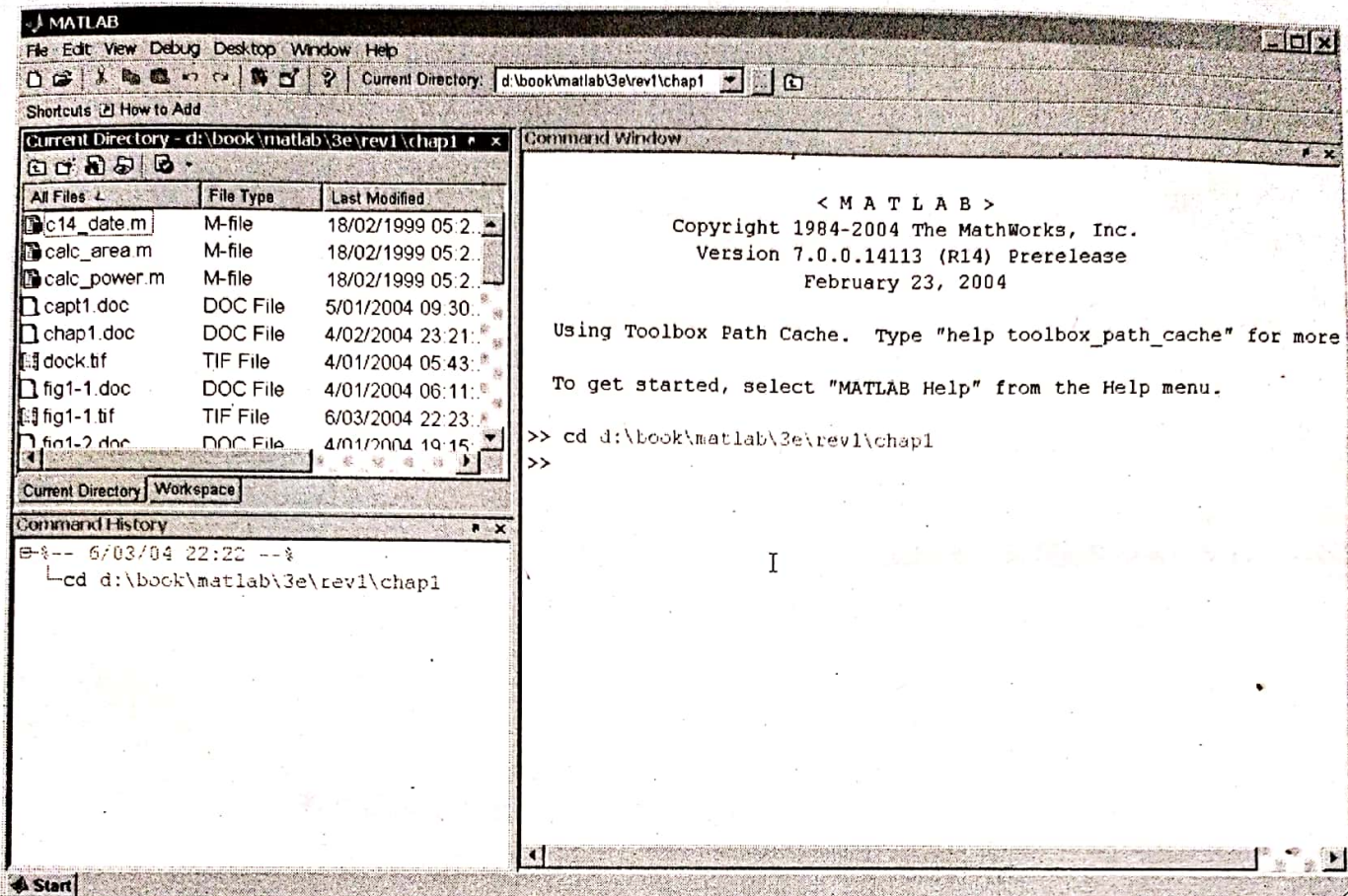


Figure 1.1 The default MATLAB desktop. The exact appearance of the desktop may differ slightly on different types of computers.

As an example of a simple interactive calculation, suppose that you want to calculate the area of a circle with a radius of 2.5 m. This can be done in the MATLAB Command Window by typing:

```
>> area = pi * 2.5^2
area =
    19.6350
```

MATLAB calculates the answer as soon as the Enter key is pressed, and stores the answer in a variable (really a 1×1 array) called `area`. The contents of the variable are displayed in the Command Window as shown in Figure 1.2, and the variable can be used in further calculations. (Note that π is predefined in MATLAB, so we can just use `pi` without first declaring it to be 3.141592...).

If a statement is too long to type on a single line, it may be continued on successive lines by typing an **ellipsis** (`...`) at the end of the first line, and then continuing on the next line. For example, the following two statements are identical.

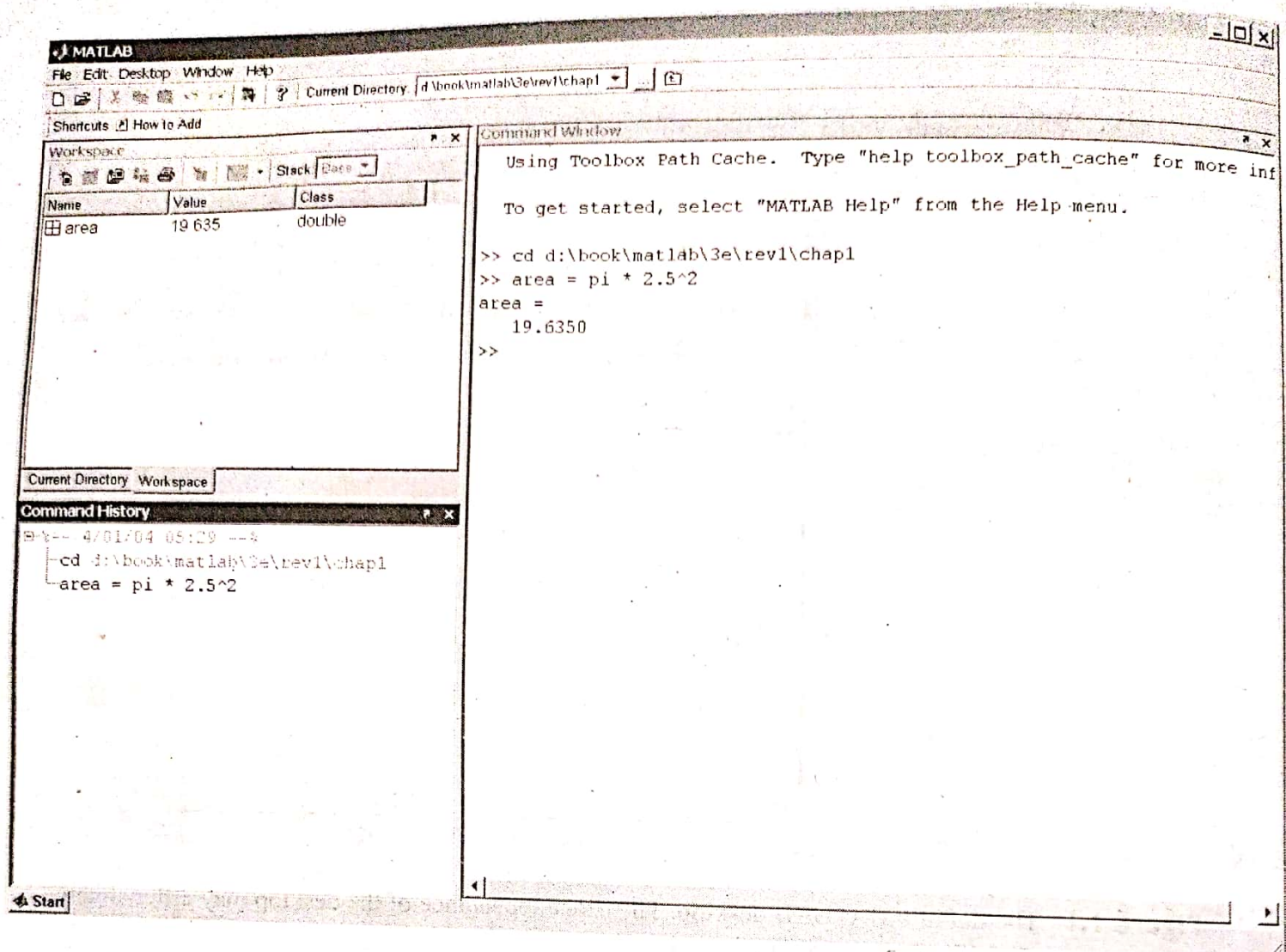


Figure 1.2 The Command Window appears on the right side of the desktop. Users enter commands and see responses here.

$x1 = 1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6;$

and

$x1 = 1 + 1/2 + 1/3 + 1/4 \dots$
 $+ 1/5 + 1/6;$

Instead of typing commands directly in the Command Window, a series of commands can be placed into a file, and the entire file can be executed by typing its name in the Command Window. Such files are called **script files**. Script files (and functions, which we will see later) are also known as **M-files**, because they have a file extension of ".m".

✓ The Command History Window

The Command History window displays a list of the commands that a user has entered in the Command Window. The list of previous commands can extend back to previous executions of the program. Commands remain in the list until they are deleted. To re-execute any command, simply double-click it with the left mouse

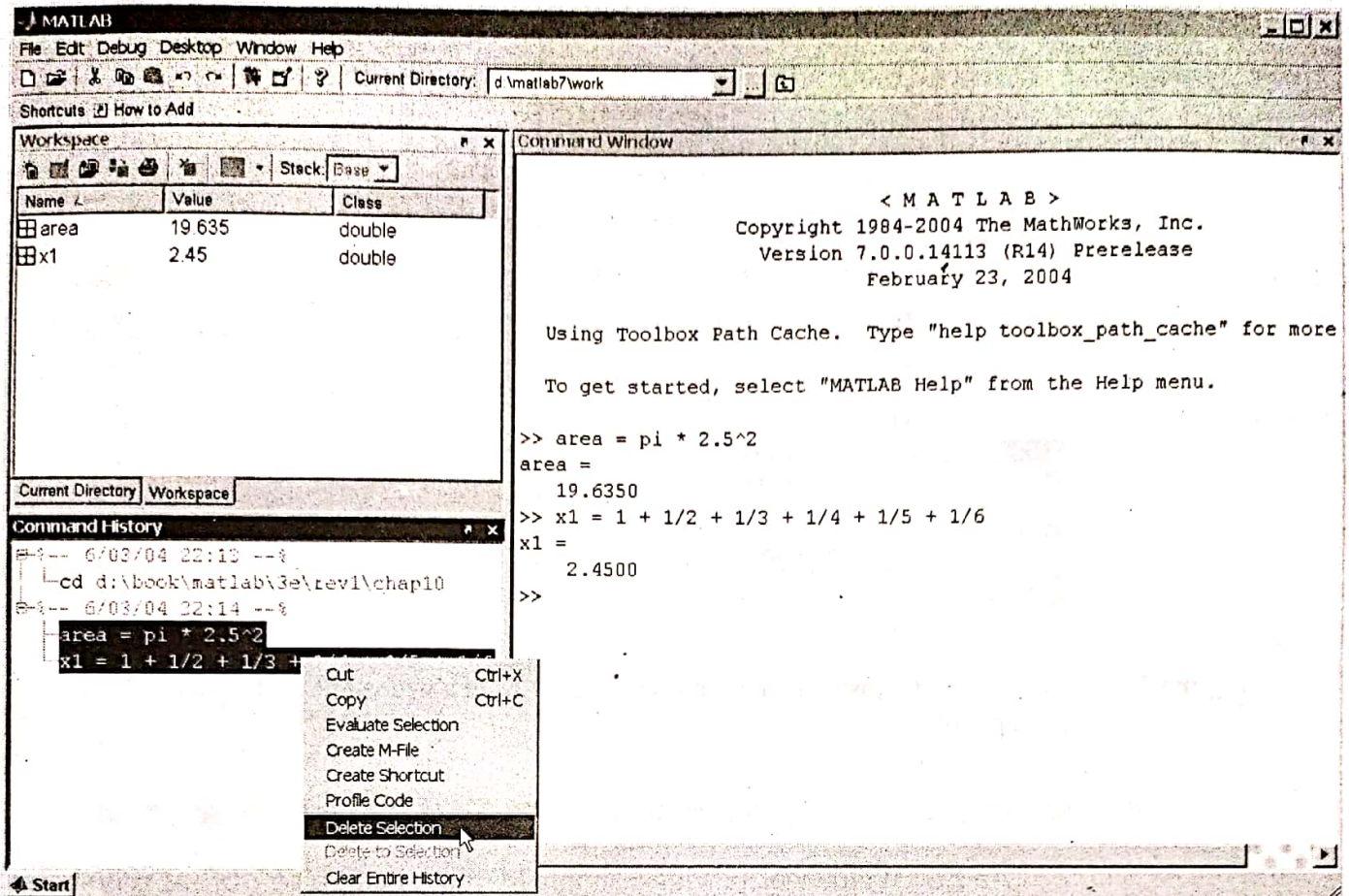




Figure 1.3 The Command History Window, showing two commands being deleted.

button. To delete one or more commands from the Command History window, select the commands and right-click them with the mouse. A popup menu will be displayed that allows the user to delete the items (see Figure 1.3).

The Start Button

The Start Button (see Figure 1.4) allows a user to access MATLAB tools, desktop tools, help files, etc. It works just like the Start button on a Windows desktop. To start a particular tool, just click on the Start Button and select the tool from the appropriate sub-menu.

The Edit/Debug Window

An **Edit Window** is used to create new M-files or to modify existing ones. An Edit Window is created automatically when you create a new M-file or open an existing one. You can create a new M-file with the "File/New/M-file" selection from the desktop menu or by clicking the  toolbar icon. You can open an existing M-file with the "File/Open" selection from the desktop menu or by clicking the  toolbar icon.

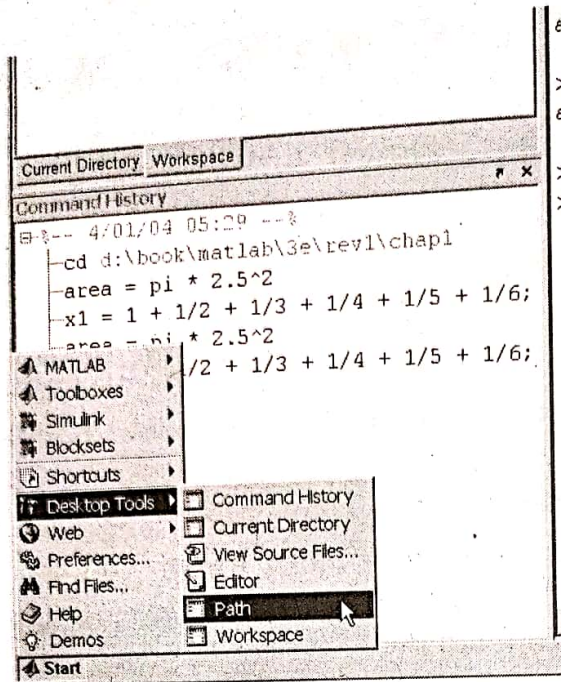


Figure 1.4 The Start Button, which allows a user to select from a wide variety of MATLAB and desktop tools.

An Edit Window displaying a simple M-file called `calc_area.m` is shown in Figure 1.5. This file calculates the area of a circle given its radius and displays the result. By default, the Edit Window is an independent window not docked to the desktop, as shown in Figure 1.5(a). The Edit Window can also be docked to the MATLAB desktop. In that case, it appears within a container called the Documents Window, as shown in Figure 1.5(b). We will learn how to dock and undock a window later in this chapter.

The Edit Window is essentially a programming text editor with the MATLAB languages features highlighted in different colors. Comments in an M-file appear in green, variables and numbers appear in black, complete character strings appear in magenta, incomplete character strings appear in red, and language keywords appear in blue.

After an M-file is saved, it may be executed by typing its name in the Command Window. For the M-file in Figure 1.5, the results are:

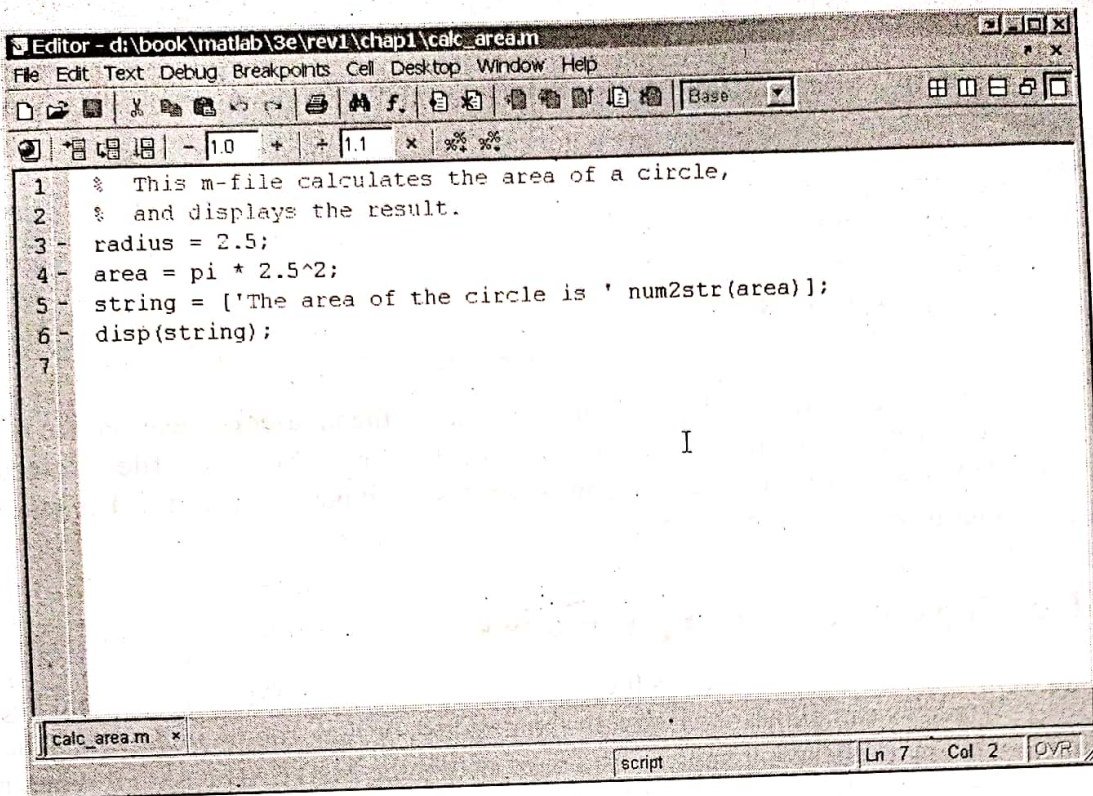
```
>> calc_area
```

```
The area of the circle is 19.635
```

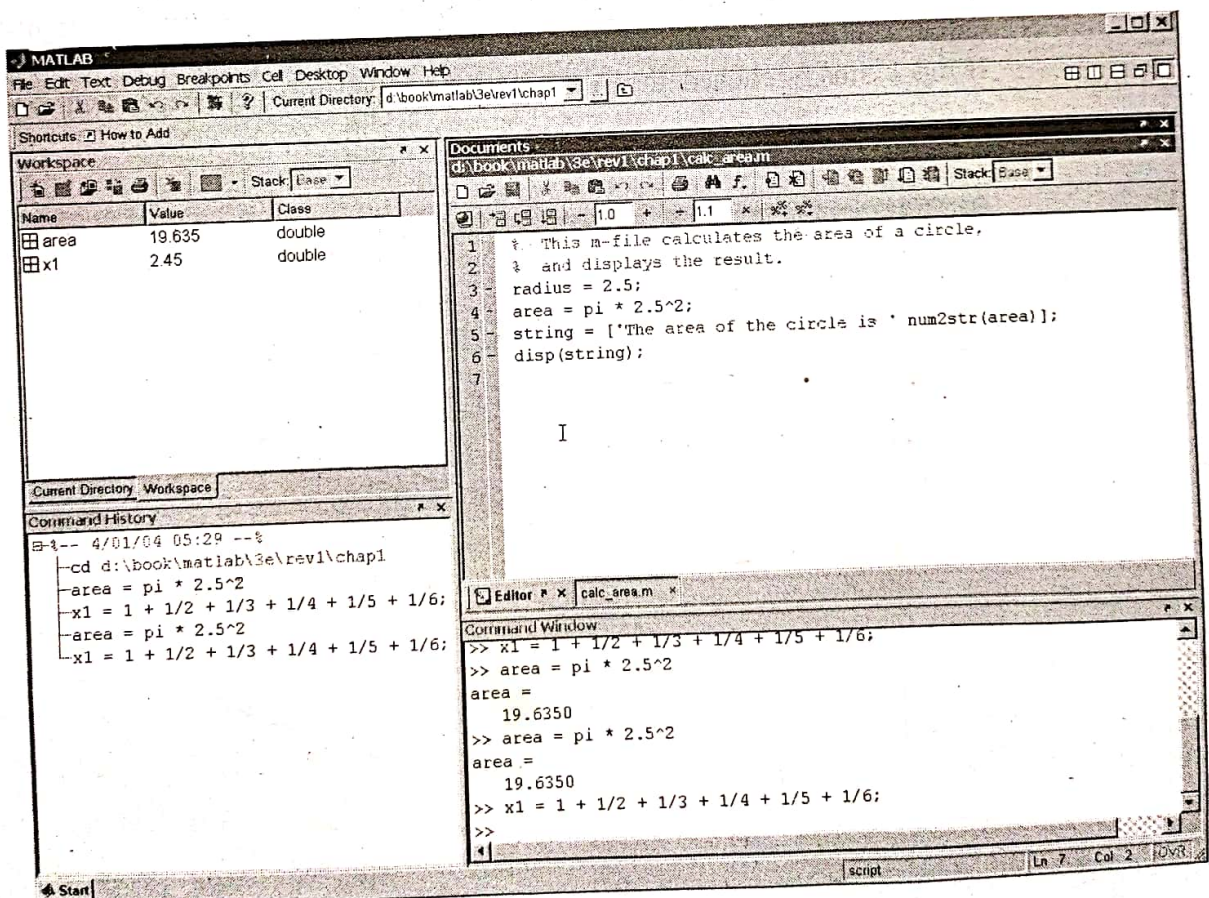
The Edit Window also doubles as a debugger, as we shall see in Chapter 2.

Figure Windows

A **Figure Window** is used to display MATLAB graphics. A figure can be a two- or three-dimensional plot of data, an image, or a graphical user interface



(a)



(b)


Figure 1.5 (a) The MATLAB Editor, displayed as an independent window. (b) The MATLAB Editor, docked to the MATLAB desktop.

(GUI). A simple script file that calculates and plots the function $\sin x$ is shown below:

```
% sin_x.m: This M-file calculates and plots the  
% function sin(x) for 0 <= x <= 6.  
x = 0:0.1:6;  
y = sin(x);  
plot(x,y);
```

If this file is saved under the name `sin_x.m`, then a user can execute the file by typing “`sin_x`” in the Command Window. When this script file is executed, MATLAB opens a figure window and plots the function $\sin x$ in it. The resulting plot is shown in Figure 1.6.

Docking and Undocking Windows

MATLAB windows such as the Command Window, the Edit Window, and Figure Windows can either be *docked* to the desktop, or they can be *undocked*. When a window is docked, it appears as a pane within the MATLAB desktop. When it is undocked it appears as an independent window on the computer screen separate from the desktop. When a window is docked to the desktop, the upper right-hand corner contains a small button with an arrow pointing up and to the right (). If this button is clicked, then the window will become an independent window.

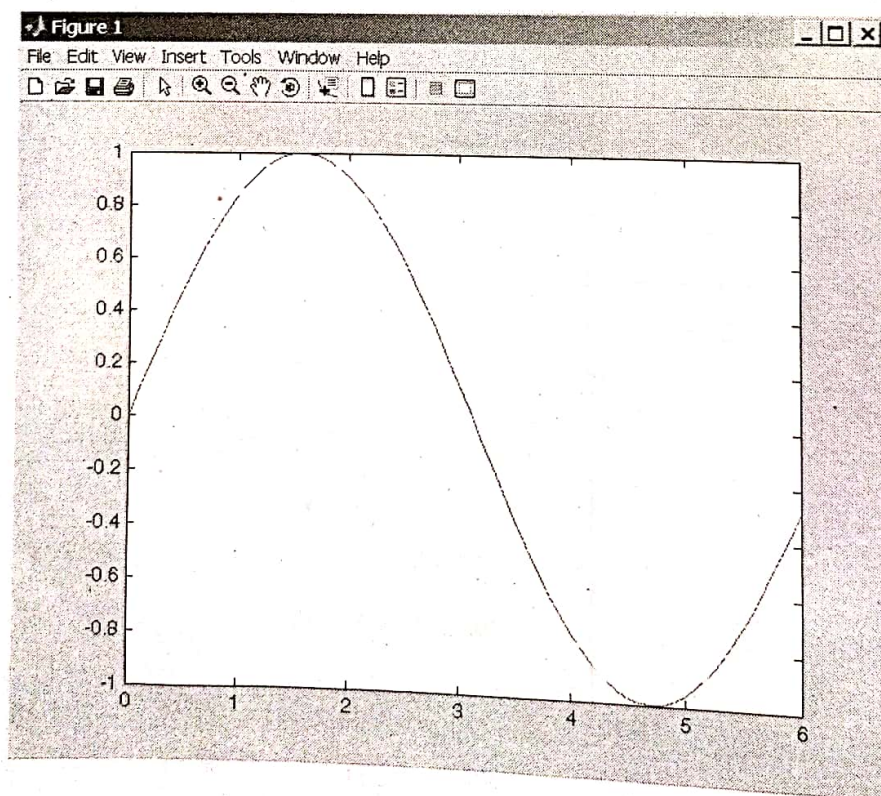



Figure 1.6 MATLAB plot of $\sin x$ versus x .

When the window is an independent window, the upper right-hand corner contains a small button with an arrow pointing down and to the right (). If this button is clicked, then the window will be redocked with the desktop. Figure 1.5 shows the Edit Window in both its docked and undocked state. Note the undock and dock arrows in the upper right hand corner.

4 The MATLAB Workspace

A statement such as

```
z = 10;
```

creates a variable named *z*, stores the value 10 in it, and saves it in a part of computer memory known as the **workspace**. A workspace is the collection of all the variables and arrays that can be used by MATLAB when a particular command, M-file, or function is executing. All commands executed in the Command Window (and all script files executed from the Command Window) share a common workspace, so they can all share variables. As we will see later, MATLAB functions differ from script files in that each function has its own separate workspace.

A list of the variables and arrays in the current workspace can be generated with the `whos` command. For example, after M-files `calc_area` and `sin_x` are executed, the workspace contains the following variables:

```
>> whos
```

Name	Size	Bytes	Class
area	1x1	8	double array
radius	1x1	8	double array
string	1x32	64	char array
x	1x61	488	double array
y	1x61	488	double array

```
Grand total is 156 elements using 1056 bytes
```

Script file `calc_area` created variables `area`, `radius`, and `string`, while script file `sin_x` created variables `x` and `y`. Note that all of the variables are in the same workspace, so if two script files are executed in succession, the second script file can use variables created by the first script file.

The contents of any variable or array may be determined by typing the appropriate name in the Command Window. For example, the contents of `string` can be found as follows:

```
>> string
```

```
string =
```

```
The area of the circle is 19.635
```

A variable can be deleted from the workspace with the `clear` command. The `clear` command takes the form

```
clear var1 var2 ...
```


where `var1` and `var2` are the names of the variables to be deleted. The command `clear variables` or simply `clear` deletes all variables from the current workspace.

✓ The Workspace Browser

The contents of the current workspace can also be examined with a GUI-based Workspace Browser. The Workspace Browser appears by default in the upper left-hand corner of the desktop. It provides a graphic display of the same information as the `whos` command, and it also shows the actual contents of each array if the information is short enough to fit within the display area. The Workspace Browser is dynamically updated whenever the contents of the workspace change.

A typical Workspace Browser window is shown in Figure 1.7. As you can see, it displays the same information as the `whos` command. Double-clicking on

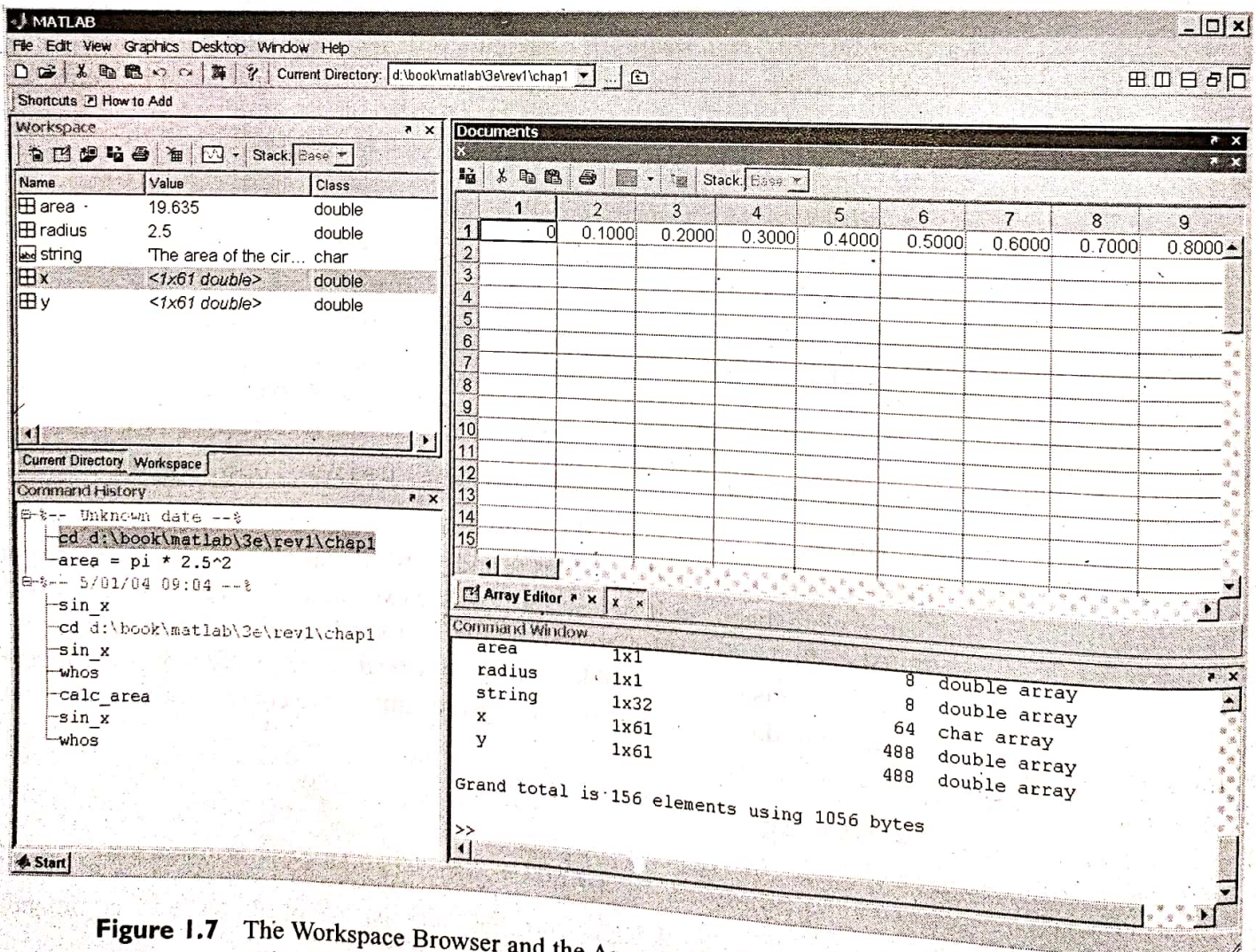



Figure 1.7 The Workspace Browser and the Array Editor. The Array Editor is invoked by double-clicking a variable in the Workspace Browser. It allows a user to change the values contained in a variable or array.

any variable in the window will bring up the Array Editor, which allows the user to modify the information stored in the variable.

One or more variables may be deleted from the workspace by selecting them in the Workspace Browser with the mouse and pressing the delete key, or by right-clicking with the mouse and selecting the delete option.

Getting Help

There are three ways to get help in MATLAB. The preferred method is to use the Help Browser. The Help Browser can be started by selecting the  icon from the desktop toolbar, or by typing `helpdesk` or `helpwin` in the Command Window. A user can get help by browsing the MATLAB documentation, or he or she can search for the details of a particular command. The Help Browser is shown in Figure 1.8.

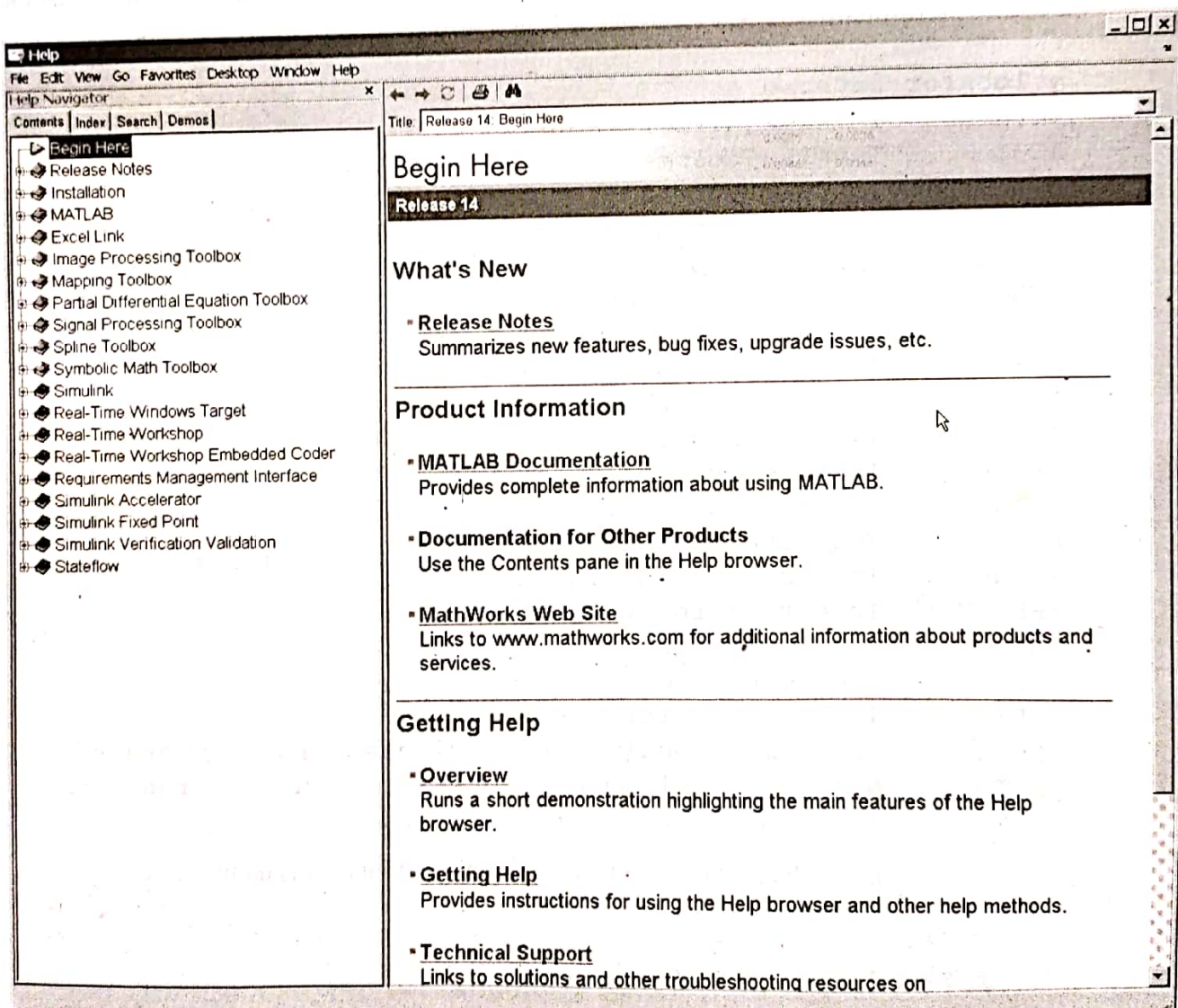


Figure 1.8 The Help Browser.

There are also two command-line oriented ways to get help. The first way is to type `help` or `help` followed by a function name in the Command Window. If you just type `help`, MATLAB will display a list of possible help topics in the Command Window. If a specific function or a toolbox name is included, help will be provided for that particular function or toolbox.

The second way to get help is the `lookfor` command. The `lookfor` command differs from the `help` command in that the `help` command searches for an exact function name match, while the `lookfor` command searches the quick summary information in each function for a match. This makes `lookfor` slower than `help`, but it improves the chances of getting back useful information. For example, suppose that you were looking for a function to take the inverse of a matrix. Since MATLAB does not have a function named `inverse`, the command "`help inverse`" will produce nothing. On the other hand, the command "`lookfor inverse`" will produce the following results:

```
> lookfor inverse
INVHILB   Inverse Hilbert matrix.
ACOS      Inverse cosine.
ACOSH     Inverse hyperbolic cosine.
ACOT      Inverse cotangent.
ACOTH     Inverse hyperbolic cotangent.
ACSC      Inverse cosecant.
ACSCH     Inverse hyperbolic cosecant.
ASEC      Inverse secant.
ASECH     Inverse hyperbolic secant.
ASIN      Inverse sine.
ASINH     Inverse hyperbolic sine.
ATAN      Inverse tangent.
ATAN2     Four quadrant inverse tangent.
ATANH     Inverse hyperbolic tangent.
ERFINV    Inverse error function.
INV       Matrix inverse.
PINV      Pseudoinverse.
IFFT      Inverse discrete Fourier transform.
IFFT2     Two-dimensional inverse discrete Fourier transform.
IFFTN     N-dimensional inverse discrete Fourier transform.
IPERMUTE  Inverse permute array dimensions.
```

From this list, we can see that the function of interest is named `inv`.

A Few Important Commands

If you are new to MATLAB, a few demonstrations may help to give you a feel for its capabilities. To run MATLAB's built-in demonstrations, type `demo` in the Command Window or select "demos" from the Start Button.

The contents of the Command Window can be cleared at any time using the `clc` command, and the contents of the current Figure Window can be cleared at any time using the `clf` command. The variables in the workspace can be cleared with the `clear` command. As we have seen, the contents of the workspace persist between the executions of separate commands and M-files, so it is possible for the results of one problem to have an effect on the next one that you may attempt to solve. To avoid this possibility, it is a good idea to issue the `clear` command at the start of each new independent calculation.

Another important command is the **abort** command. If an M-file appears to be running for too long, it may contain an infinite loop, and it will never terminate. In this case, the user can regain control by typing control-c (abbreviated `^c`) in the Command Window. This command is entered by holding down the control key while typing a "c." When MATLAB detects a `^c`, it interrupts the running program and returns a command prompt.

The exclamation point (!) is another important special character. Its special purpose is to send a command to the computer's operating system. Any characters after the exclamation point will be sent to the operating system and executed as though they had been typed at the operating system's command prompt. This feature lets you embed operating system commands directly into MATLAB programs.

Finally, it is possible to keep track of everything done during a MATLAB session with the **diary** command. The form of this command is

```
diary filename
```

After this command is typed, a copy of all input and most output typed in the Command Window is echoed in the diary file. This is a great tool for recreating events when something goes wrong during a MATLAB session. The command "diary off" suspends input into the diary file, and the command "diary on" resumes input again.

The MATLAB Search Path

MATLAB has a search path that it uses to find M-files. MATLAB's M-files are organized in directories on your file system. Many of these directories of M-files are provided along with MATLAB, and users may add others. If a user enters a name at the MATLAB prompt, the MATLAB interpreter attempts to find the name as follows:

1. It looks for the name as a variable. If it is a variable, MATLAB displays the current contents of the variable.
2. It checks to see if the name is an M-file in the current directory. If it is, MATLAB executes that function or command.
3. It checks to see if the name is an M-file in any directory in the search path. If it is, MATLAB executes that function or command.

Note that MATLAB checks for variable names first, so *if you define a variable with the same name as a MATLAB function or command, that function or command becomes inaccessible*. This is a common mistake made by novice users.

Programming Pitfalls

Never use a variable with the same name as a MATLAB function or command. If you do so, that function or command will become inaccessible.

Also, if there is more than one function or command with the same name, the *first* one found on the search path will be executed, and all of the others will be inaccessible. This is a common problem for novice users, since they sometimes create M-files with the same names as standard MATLAB functions, making them inaccessible.

Programming Pitfalls

Never create an M-file with the same name as a MATLAB function or command.

MATLAB includes a special command (`which`) to help you find out just which version of a file is being executed and where it is located. This can be

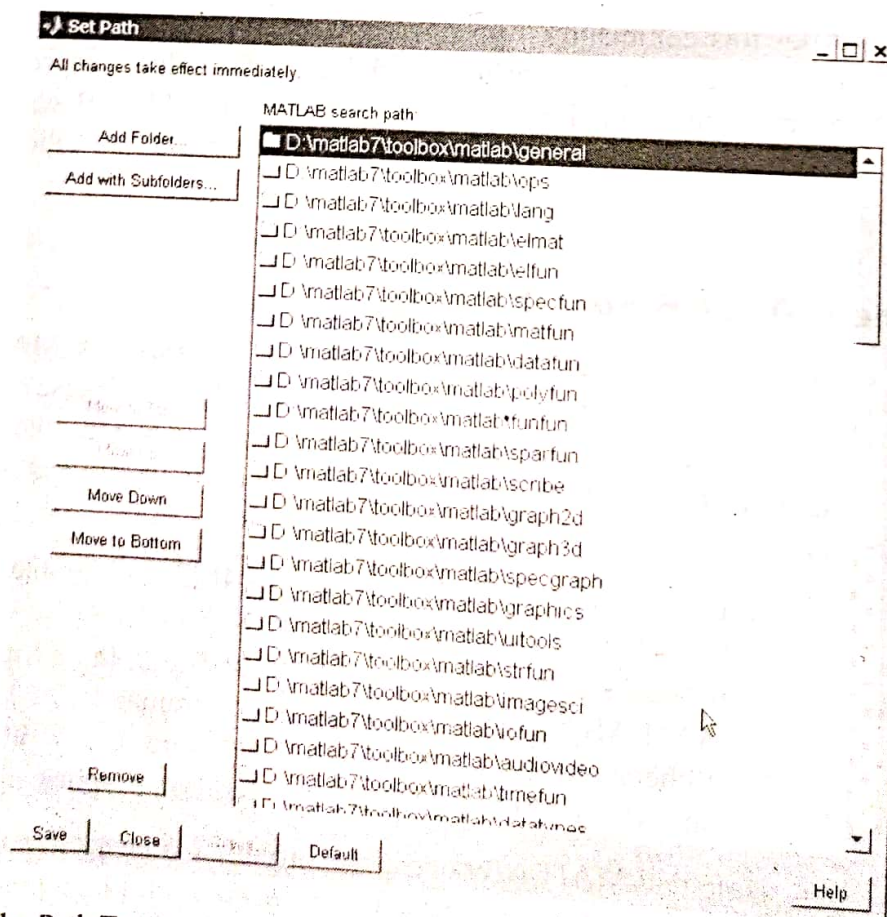


Figure 1.9 The Path Tool.

useful in finding filename conflicts. The format of this command is `which functionname`, where *functionname* is the name of the function that you are trying to locate. For example, the cross-product function `cross.m` can be located as follows:

```
» which cross
```

```
C:\Matlab7\toolbox\matlab\specfun\cross.m
```

The MATLAB search path can be examined and modified at any time by selecting “Desktop Tools/Path” from the Start Button, or by typing `editpath` in the Command Window. The Path Tool is shown in Figure 1.9. It allows a user to add, delete, or change the order of directories in the path.

Other path-related functions include:

- `addpath`—Add directory to MATLAB search path.
- `path`—Display MATLAB search path.
- `path2rc`—Adds current directory to MATLAB search path.
- `rmpath`—Remove directory from MATLAB search path.

1.4 Using MATLAB as a Scratch Pad

In its simplest form, MATLAB can be used as a scratch pad to perform mathematical calculations. The calculations to be performed are typed directly into the Command Window, using the symbols `+`, `-`, `*`, `/`, and `^` for addition, subtraction, multiplication, division, and exponentiation, respectively. After an expression is typed, the results of the expression will be automatically calculated and displayed. For example, suppose we would like to calculate the volume of a cylinder of radius r and length l . The area of the circle at the base of the cylinder is given by the equation

$$A = \pi r^2 \quad (1-1)$$

and the total volume of the cylinder will be

$$V = Al \quad (1-2)$$

If the radius of the cylinder is 0.1 m and the length is 0.5 m, then the volume of the cylinder can be found using the MATLAB statements (user inputs are shown in bold face):

```
» A = pi * 0.1^2
A =
    0.0314
» V = A * 0.5
V =
    0.0157
```

Note that `pi` is predefined to be the value 3.141592 . . . Also, note that the value stored in `A` was saved by MATLAB and reused when we calculated `V`.



Quiz 1.1

This quiz provides a quick check to see if you have understood the concepts introduced in Chapter 1. If you have trouble with the quiz, reread the sections, ask your instructor, or discuss the material with a fellow student. The answers to this quiz are found in the back of the book.

1. What is the purpose of the MATLAB Command Window? The Edit Window? The Figure Window?
2. List the different ways that you can get help in MATLAB.
3. What is a workspace? How can you determine what is stored in a MATLAB workspace?
4. How can you clear the contents of a workspace?
5. The distance traveled by a ball falling in the air is given by the equation

$$x = x_0 + v_0 t + \frac{1}{2} a t^2$$

Use MATLAB to calculate the position of the ball at time $t = 5$ s if $x_0 = 10$ m, $v_0 = 15$ m/s, and $a = -9.81$ m/sec².

6. Suppose that $x = 3$ and $y = 4$. Use MATLAB to evaluate the following expression:

$$\frac{x^2 y^3}{(x - y)^2}$$

The following questions are intended to help you become familiar with MATLAB tools.

7. Execute the M-files `calc_area.m` and `sin_x.m` in the Command Window (these M-files are available from the book's Web site). Then use the Workspace Browser to determine what variables are defined in the current workspace.
8. Use the Array Editor to examine and modify the contents of variable `x` in the workspace. Then type the command `plot(x, y)` in the Command Window. What happens to the data displayed in the Figure Window?

2.1 Variables and Arrays

The fundamental unit of data in any MATLAB program is the **array**. An array is a collection of data values organized into rows and columns and known by a single name. Individual data values within an array are accessed by including the name of the array followed by subscripts in parentheses that identify the row and column of the particular value. Even scalars are treated as arrays by MATLAB—they are simply arrays with only one row and one column.

Arrays can be classified as either **vectors** or **matrices**. The term “vector” is usually used to describe an array with only one dimension, while the term “matrix” is usually used to describe an array with two or more dimensions. In this text, we will use the term “vector” when discussing one-dimensional arrays and the term “matrix” when discussing arrays with two or more dimensions. If a particular discussion applies to both types of arrays, we will use the generic term “array.” (See Figure 2.1)

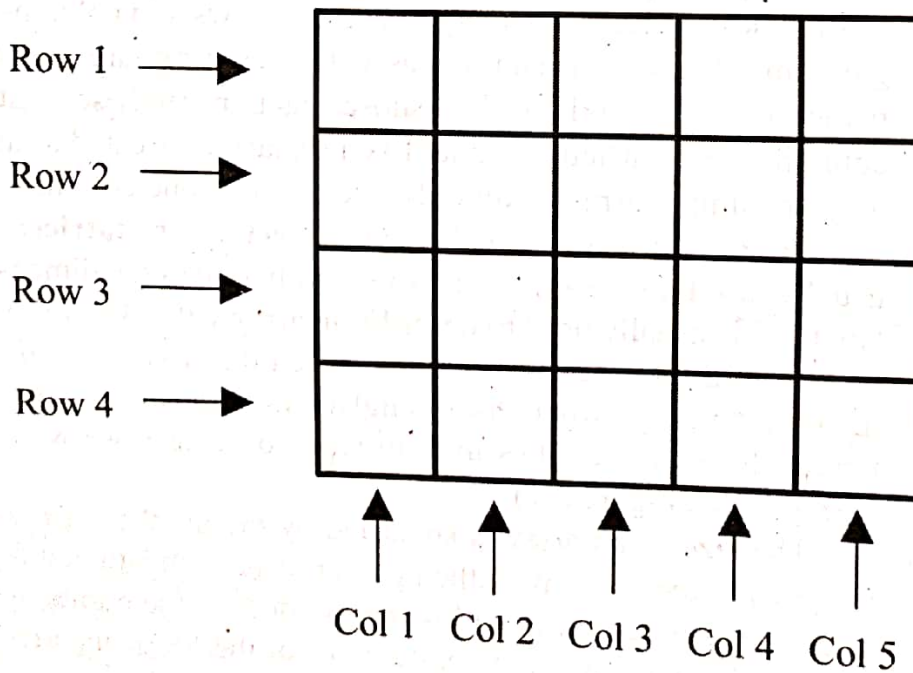
The **size** of an array is specified by the number of rows and the number of columns in the array, with the number of rows mentioned first. The total number of elements in the array will be the product of the number of rows and the number of columns. For example, the sizes of the following arrays are

Array	Size
$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$	This is 3×2 matrix, containing 6 elements.
$b = [1 \ 2 \ 3 \ 4]$	This is a 1×4 array containing 4 elements, known as a row vector .
$c = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	This is a 3×1 array containing 3 elements, known as a column vector .

Individual elements in an array are addressed by the array name followed by the row and column of the particular element. If the array is a row or column vector, then only one subscript is required. For example, in the above arrays $a(2, 1)$ is 3 and $c(2) = 2$.

A MATLAB **variable** is a region of memory containing an array that is known by a user-specified name. The contents of the array may be used or modified at any time by including its name in an appropriate MATLAB command.

MATLAB variable names must begin with a letter, followed by any combination of letters, numbers, and the underscore (`_`) character. Only the first 63



array arr

Figure 2.1 An array is a collection of data values organized into rows and columns.

characters are significant; if more than 63 are used, the remaining characters will be ignored. If two variables are declared with names that only differ in the 64th character, MATLAB will treat them as the same variable. MATLAB will issue a warning if it has to truncate a long variable name to 63 characters.¹

Programming Pitfalls

Make sure that your variable names are unique in the first 63 characters. Otherwise, MATLAB will not be able to tell the difference between them.

When writing a program, it is important to pick meaningful names for the variables. Meaningful names make a program *much* easier to read and to maintain. Names such as *day*, *month*, and *year* are quite clear even to a person seeing a program for the first time. Since spaces cannot be used in MATLAB variable names, underscore characters can be substituted to create meaningful names. For example, *exchange rate* might become `exchange_rate`.

Good Programming Practice

Always give your variables descriptive and easy-to-remember names. For example, a currency exchange rate could be given the name `exchange_rate`. This practice will make your programs clearer and easier to understand.

It is also important to include a **data dictionary** in the header of any program that you write. A data dictionary lists the definition of each variable used in a program. The definition should include both a description of the contents of the item and the units in which it is measured. A data dictionary may seem unnecessary while the program is being written, but it is invaluable when you or another person have to go back and modify the program at a later time.

Good Programming Practice

Create a data dictionary for each program to make program maintenance easier.

¹Until MATLAB 6.5, the maximum number of significant characters in a variable name was 31. If you are writing a program that might be run on MATLAB 6.1 or earlier, be sure to limit your variable names to 31 characters or less. Otherwise, your program might work properly on MATLAB 7 but fail when it is executed on an earlier version.

The MATLAB language is case-sensitive, which means that uppercase and lowercase letters are not the same. Thus the variables `name`, `NAME`, and `Name` are all different in MATLAB. You must be careful to use the same capitalization every time that variable name is used. While it is not required, it is customary to use all lowercase letters for ordinary variable names.

* Good Programming Practice

Be sure to capitalize a variable exactly the same way each time that it is used. It is good practice to use only lower-case letters in variable names.

The most common types of MATLAB variables are `double` and `char`. Variables of type `double` consist of scalars or arrays of 64-bit double-precision floating-point numbers. They can hold real, imaginary, or complex values. The real and imaginary components of each variable can be positive or negative numbers in the range 10^{-308} to 10^{308} , with 15 to 16 significant decimal digits of accuracy. They are the principal numerical data type in MATLAB.

A variable of type `double` is automatically created whenever a numerical value is assigned to a variable name. The numerical values assigned to `double` variables can be real, imaginary, or complex. A real value is just a number. For example, the following statement assigns the real value 10.5 to the `double` variable `var`:

```
var = 10.5;
```

An imaginary value is defined by appending the letter `i` or `j` to a number. For example, `10i` and `-4j` are both imaginary values. The following statement assigns the imaginary value `4i` to the `double` variable `var`:

```
var = 4i;
```

A complex value has both a real and an imaginary component. It is created by adding a real and an imaginary number together. For example, the following statement assigns the complex value `10 + 10i` to variable `var`:

```
var = 10 + 10i;
```

Variables of type `char` consist of scalars or arrays of 16-bit values, each representing a single character. Arrays of this type are used to hold character strings. They are automatically created whenever a single character or a character string is assigned to a variable name. For example, the following statement creates a variable of type `char` whose name is `comment` and stores the specified string in it. After the statement is executed, `comment` will be a 1×26 character array.

```
comment = 'This is a character string';
```


2.2 Initializing Variables in MATLAB | 25

In a language such as C, the type of every variable must be explicitly declared in a program before it is used. These languages are said to be **strongly typed**. In contrast, MATLAB is a **weakly typed** language. Variables may be created at any time by simply assigning values to them, and the type of data assigned to the variable determines the type of variable that is created.

2.3 Multidimensional Arrays

As we have seen, MATLAB arrays can have one or more dimensions. One-dimensional arrays can be visualized as a series of values laid out in a column, with a single subscript used to select the individual array elements (Figure 2.2a). Such arrays are useful to describe data that is a function of one independent variable, such as a series of temperature measurements made at fixed intervals of time.

Some types of data are functions of more than one independent variable. For example, we might wish to measure the temperature at five different locations at four different times. In this case, our 20 measurements could logically be grouped into five different columns of four measurements each, with a separate column for each location (Figure 2.2b). We will use two subscripts to access a given element in this array: the first one to select the row and the second one to select the column. Such arrays are called **two-dimensional arrays**. The number of elements in a two-dimensional array will be the product of the number of rows and the number of columns in the array.

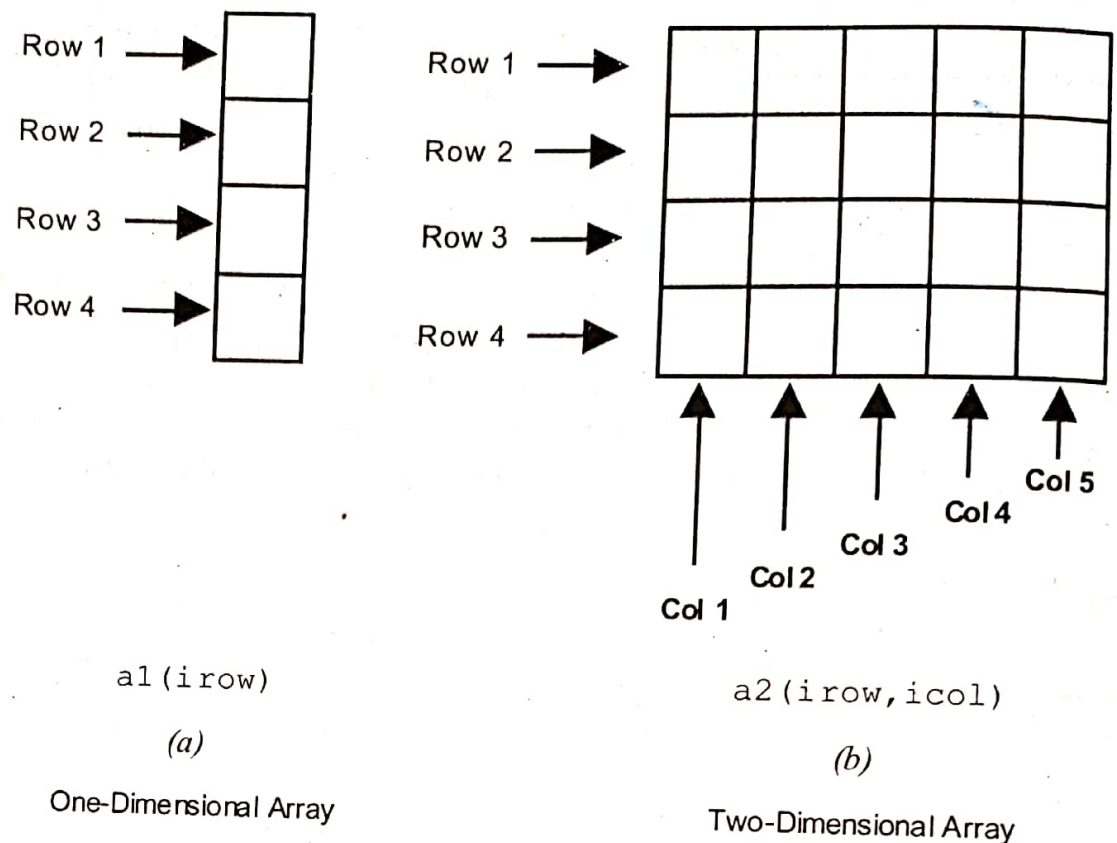


Figure 2.2 Representations of one- and two-dimensional arrays.

MATLAB allows us to create arrays with as many dimensions as necessary for any given problem. These arrays have one subscript for each dimension, and an individual element is selected by specifying a value for each subscript. The total number of elements in the array will be the product of the maximum value of each subscript. For example, the following two statements create a $2 \times 3 \times 2$ array *c*:

```
>> c(:,:,1)=[1 2 3; 4 5 6];
>> c(:,:,2)=[7 8 9; 10 11 12];
>> whos c
```

Name	Size	Bytes	Class
c	2x3x2	96	double array

This array contains 12 elements ($2 \times 3 \times 2$). Its contents can be displayed just like any other array.

```
>> c
c(:,:,1) =
     1     2     3
     4     5     6
c(:,:,2) =
     7     8     9
    10    11    12
```


Storing Multidimensional Arrays in Memory

A two-dimensional array with m rows and n columns will contain $m \times n$ elements, and these elements will occupy $m \times n$ successive locations in the computer's memory. How are the elements of the array arranged in the computer's memory? MATLAB always allocates array elements in **column major order**. That is, MATLAB allocates the first column in memory, then the second, then the third, etc., until all of the columns have been allocated. Figure 2.3 illustrates this memory allocation scheme for a 4×3 array a . As we can see, element $a(1, 2)$ is really the fifth element allocated in memory. The order according to which elements are allocated in memory will become important when we discuss single-subscript addressing in the next section, and low-level I/O functions in Chapter 8.

This same allocation scheme applies to arrays with more than two dimensions. The first array subscript is incremented most rapidly, the second subscript is incremented less rapidly, etc., and the last subscript is incremented most slowly. For example, in a $2 \times 2 \times 2$ array, the elements would be allocated in the following order: (1,1,1), (2,1,1), (1,2,1), (2,2,1), (1,1,2), (2,1,2), (1,2,2), (2,2,2).

Accessing Multidimensional Arrays with One Dimension

One of MATLAB's peculiarities is that it will permit a user or programmer to treat a multidimensional array as though it were a one-dimensional array whose length is equal to the number of elements in the multidimensional array. If a multidimensional array is addressed with a single dimension, then the elements will be accessed in the order in which they were allocated in memory.

For example, suppose that we declare the 4×3 element array a as follows:

```
>> a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
```

```
a =  
    1     2     3  
    4     5     6  
    7     8     9  
   10    11    12
```

Then the value of $a(5)$ will be 2, which is the value of element $a(1, 2)$, because $a(1, 2)$ was allocated fifth in memory.

Under normal circumstances, you should never use this feature of MATLAB. Addressing multidimensional arrays with a single subscript is a recipe for confusion.

* Good Programming Practice

Always use the proper number of dimensions when addressing a multidimensional array.

1	2	3
4	5	6
7	8	9
10	11	12

a

(a)

**Arrangement
in Computer
Memory**

1	a(1,1)
4	a(2,1)
7	a(3,1)
10	a(4,1)
2	a(1,2)
5	a(2,2)
8	a(3,2)
11	a(4,2)
3	a(1,3)
6	a(2,3)
9	a(3,3)
12	a(4,3)
.	
.	
.	

(b)

Figure 2.3 (a) Data values for array a. (b) Layout of values in memory for array a.

2.8 Scalar and Array Operations

Calculations are specified in MATLAB with an assignment statement, whose general form is

```
variable_name = expression;
```

The assignment statement calculates the value of the expression to the right of the equal sign, and *assigns* that value to the variable named on the left of the equal sign.

Table 2.5 Arithmetic Operations Between Two Scalars

Operation	Algebraic Form	MATLAB Form
Addition	$a + b$	$a + b$
Subtraction	$a - b$	$a - b$
Multiplication	$a \times b$	$a * b$
Division	$\frac{a}{b}$	a / b
Exponentiation	a^b	$a ^ b$

Note that the equal sign does not mean equality in the usual sense of the word. Instead, it means: *store the value of expression into location variable_name*. For this reason, the equal sign is called the **assignment operator**. A statement like

```
ii = ii + 1;
```

is complete nonsense in ordinary algebra, but makes perfect sense in MATLAB. It means: take the current value stored in variable `ii`, add one to it, and store the result back into variable `ii`.

Scalar Operations

The expression to the right of the assignment operator can be any valid combination of scalars, arrays, parentheses, and arithmetic operators. The standard arithmetic operations between two scalars are given in Table 2.5.

Parentheses may be used to group terms whenever desired. When parentheses are used, the expressions inside the parentheses are evaluated before the expressions outside the parentheses. For example, the expression $2 ^ ((8+2)/5)$ is evaluated as shown below

$$\begin{aligned} 2 ^ ((8+2)/5) &= 2 ^ (10/5) \\ &= 2 ^ 2 \\ &= 4 \end{aligned}$$

Array and Matrix Operations

MATLAB supports two types of operations between arrays, known as *array operations* and *matrix operations*. **Array operations** are operations performed between arrays on an **element-by-element basis**. That is, the operation is performed on corresponding elements in the two arrays. For example, if $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $b = \begin{bmatrix} -1 & 3 \\ -2 & 3 \end{bmatrix}$, then $a + b = \begin{bmatrix} 0 & 5 \\ 1 & 5 \end{bmatrix}$. Note that for these operations to work, *the number of rows and columns in both arrays must be the same*. If not, MATLAB will generate an error message.

Array operations may also occur between an array and a scalar. If the operation is performed between an array and a scalar, the value of the scalar is applied to every element of the array. For example, if $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, then $a + 4 = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$.

In contrast, **matrix operations** follow the normal rules of linear algebra, such as matrix multiplication. In linear algebra, the product $c = a \times b$ is defined by the equation

$$c(i, j) = \sum_{k=1}^n a(i, k)b(k, j)$$

For example, if $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $b = \begin{bmatrix} -1 & 3 \\ -2 & 1 \end{bmatrix}$, then $a \times b = \begin{bmatrix} -5 & 5 \\ -11 & 13 \end{bmatrix}$

Note that for matrix multiplication to work, *the number of columns in matrix a must be equal to the number of rows in matrix b*.

MATLAB uses a special symbol to distinguish array operations from matrix operations. In the cases where array operations and matrix operations have a different definition, MATLAB uses a period before the symbol to indicate an array operation (for example, `.*`). A list of common array and matrix operations is given in Table 2.6 on page 49.

New users often confuse array operations and matrix operations. In some cases, substituting one for the other will produce an illegal operation, and MATLAB will report an error. In other cases, both operations are legal, and MATLAB will perform the wrong operation and come up with a wrong answer. The most common problem happens when working with square matrices. Both array multiplication and matrix multiplication are legal for two square matrices of the same size, but the resulting answers are totally different. Be careful to specify exactly what you want!

Programming Pitfalls

Be careful to distinguish between array operations and matrix operations in your MATLAB code. It is especially common to confuse array multiplication with matrix multiplication.

Example 2.1

Assume that a , b , c , and d are defined as follows:

$$a = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} -1 & 2 \\ 0 & 1 \end{bmatrix}$$

$$c = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$d = 5$$

What is the result of each of the following expressions?

- | | |
|--------------|--------------|
| (a) $a + b$ | (e) $a + c$ |
| (b) $a .* b$ | (f) $a + d$ |
| (c) $a * b$ | (g) $a .* d$ |
| (d) $a * c$ | (h) $a * d$ |

SOLUTION

(a) This is array or matrix addition: $a + b = \begin{bmatrix} 0 & 2 \\ 2 & 2 \end{bmatrix}$

(b) This is element-by-element array multiplication: $a .* b = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

(c) This is matrix multiplication: $a * b = \begin{bmatrix} -1 & 2 \\ -2 & 5 \end{bmatrix}$

(d) This is matrix multiplication: $a * c = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$

(e) This operation is illegal, since a and c have different numbers of columns.

(f) This is addition of an array to a scalar: $a + d = \begin{bmatrix} 6 & 5 \\ 7 & 6 \end{bmatrix}$

(g) This is array multiplication: $a .* d = \begin{bmatrix} 5 & 0 \\ 10 & 5 \end{bmatrix}$

(h) This is matrix multiplication: $a * b = \begin{bmatrix} 5 & 0 \\ 10 & 5 \end{bmatrix}$

The matrix left division operation has a special significance that we must understand. A 3×3 set of simultaneous linear equations takes the form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \quad (2-1)$$

which can be expressed as

$$Ax = B \quad (2-2)$$

where $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$, $B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$, and $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$.

Equation (2-2) can be solved for x using linear algebra. The result is

$$x = A^{-1}B \quad (2-3)$$

Since the left division operator $A \setminus B$ is defined to be $\text{inv}(A) \times B$, the left division operator solves a system of simultaneous equations in a single statement!

★ Good Programming Practice

Use the left division operator to solve systems of simultaneous equations.

Table 2.6 Common Array and Matrix Operations

Operation	MATLAB Form	Comments
Array Addition	$a + b$	Array addition and matrix addition are identical.
Array Subtraction	$a - b$	Array subtraction and matrix subtraction are identical.
Array Multiplication	$a .* b$	Element-by-element multiplication of a and b . Both arrays must be the same shape, or one of them must be a scalar.
Matrix Multiplication	$a * b$	Matrix multiplication of a and b . The number of columns in a must equal the number of rows in b .
Array Right Division	$a ./ b$	Element-by-element division of a and b : $a(i,j) / b(i,j)$. Both arrays must be the same shape, or one of them must be a scalar.
Array Left Division	$a .\setminus b$	Element-by-element division of a and b , but with b in the numerator: $b(i,j) / a(i,j)$. Both arrays must be the same shape, or one of them must be a scalar.
Matrix Right Division	a / b	Matrix division defined by $a * \text{inv}(b)$, where $\text{inv}(b)$ is the inverse of matrix b .
Matrix Left Division	$a \setminus b$	Matrix division defined by $\text{inv}(a) * b$, where $\text{inv}(a)$ is the inverse of matrix a .
Array Exponentiation	$a .^ b$	Element-by-element exponentiation of a and b : $a(i,j)^{b(i,j)}$. Both arrays must be the same shape, or one of them must be a scalar.