CHAPTER 2

Getting Started

Chapter Objectives

This chapter introduces you to some of the fundamentals of computing that apply to all programming languages, and specifically to the programming environment used for program development. The fundamentals of programming include:

- How to use abstraction to think in a general way about a collection of data and procedural steps
- How to describe the solution of a problem as an algorithm
- The three paradigms of computing and the position of MATLAB in that spectrum
- Three aspects of the apparently simple task of assigning a value to a variable

As you study the MATLAB user interface, you will understand:

- How to use the Command window to explore single commands interactively and how to recall earlier commands to be repeated or changed
- Where to examine the variables and files created in MATLAB
- How to view data created in MATLAB
- How MATLAB presents graphical data in separate windows
- How to create scripts to solve simple arithmetic problems

Introduction

The name MATLAB is a contraction of **Mat**rix **Lab**oratory. It was developed for engineers to create, manipulate, and visualize matrices—rectangular arrays of numerical values. At its most basic level, MATLAB can perform the same functions as your scientific calculator, but it has expanded far beyond its original capabilities and now provides an interactive system and programming language for many applications, including financial analysis as well as general scientific and technical computation.

The following are the fundamental components of MATLAB:

- A computing system that accepts one instruction at a time in text form and implements the logic of that instruction. Instructions must conform to a specific syntax and vocabulary, which will be the topic of Chapters 3–9.
- A large library of modules that provide high-level capabilities for processing data. These modules will be the major topic of Chapters 10–17.
- A graphical user interface (GUI) that lets users assemble and implement programs that solve specific problems. The rest of this chapter will describe the basic behavior of these windows.

MATLAB offers a number of advantages to users over conventional, compiled languages like C++, Java, or FORTRAN:

- Because MATLAB programs are interpreted rather than compiled, the process of producing a working solution can be much quicker than with compiled languages.
- MATLAB excels at numerical calculations, especially matrix calculations.
- MATLAB has built-in graphics capabilities that produce professional-looking images for reports.

However, the very attributes that make MATLAB convenient for a user to develop quick solutions to certain problems make it unsuitable for other kinds of projects. For example:

- MATLAB does not work well for large computing projects where a number of developers share coding responsibilities.
- Professional GUIs and windowing applications (like the MATLAB system itself) are best written in a compiled language.

2.1 Programming Language Background

Before learning about concepts in computing, you need to understand the background of programming languages. This section discusses the following aspects of programming languages: abstraction, algorithms, programming paradigms, and three fundamental concepts of programming—assigning values to variables, data typing, and the difference between classes and objects.

2.1.1 Abstraction

For the purpose of this text, we will define **abstraction** as "expressing a quality apart from a particular implementation." We use the concept of abstraction in everyday conversation without thinking about it:

> "To convert from degrees Celsius to Kelvin, you add 273 to *the temperature*."
>
> "He *drove home* from the office."

The first is an example of **data abstraction**. "The temperature" could mean a single reading from the thermometer hanging outside the window or a table of temperature readings for the month of August. The specifics are unimportant; the phrase captures all you need to know.

The second example is actually much more complex—an example of multiple levels of **procedural abstraction**. To a businessperson taking the same route home every night, "drive home" is all that is required to understand the idea. To a competent driver unfamiliar with the route, the next level of abstraction might be necessary—turn right out of the parking lot, left onto Main Street, and so on. For instructions to guide a future robotic commuter vehicle, an incredibly fine-grained level of abstraction will be required. Everything taken for granted in the higher level abstractions will need to be meticulously spelled out for the robotic vehicle—start the engine, accelerate the vehicle, look out for traffic, keep in the lane, find the turn, steer the vehicle, control the speed, observe and obey all signs, and so on.

2.1.2 Algorithms

Chapter 1 defined problem solving as the ability to isolate sub-problems that seem simple and appropriate to solve, and then assemble the solutions to these sub-problems. The solutions to each of these sub-problems would be expressed as an **algorithm**, which is a sequence of instructions for solving a sub-problem. The process of solving each sub-problem and assembling the solutions to form the solution to the whole problem would also be expressed as an algorithm at a higher level of abstraction.

The level of abstraction needed to describe an algorithm varies greatly with the mechanism available. For example, describing the algorithm (recipe) for baking cookies might take the following forms:

- To your grandmother, who has been baking cookies for the last 50 years, it might be "Please bake some cookies."
- To others it might be "Buy a cookie mix and follow the directions."
- To a young person learning to cook from scratch, the algorithm might include an intricate series of instructions for measuring, sifting, and combining ingredients; setting the oven temperature and preheating the oven; forming the cookies and putting them on the cookie sheet; and so on.

In programming terms, algorithms are frequently expressed first conceptually at a high level of abstraction, as demonstrated in Section 1.5. The solutions to each sub-problem would then be expressed at lower and lower levels of abstraction until the description is sufficient to write programs that solve each sub-problem, thereby contributing the pieces that, when assembled, solve the whole problem.

2.1.3 Programming Paradigms

From the Greek word *paradeigma*—"to show alongside"—the *American Heritage Dictionary* defines a paradigm as "a set of assumptions, concepts, values, and practices that constitutes a way of viewing reality for the community that shares them, especially in an intellectual discipline." So a programming paradigm becomes a codified set of practices allowing the community of computing professionals to frame their ideas. This section considers three radically different paradigms: functional programming, procedural programming, and object-oriented programming.

Functional programming is typically associated with languages like Lisp and Forth, in which every programming operation is actually implemented as a function call with no side effects (changes of state of the program surroundings) permitted or implemented in the language. Without side effects, a programming solution can be mathematically proven to be correct—an enormous advantage. Except for the discussion of recursion, this paradigm will not be mentioned again.

Procedural programming is typical of languages like FORTRAN, C, and MATLAB, where the basic programs or sub-programs are sequences of operations on data items that are generally accessible to all programs. Although side effects from sub-programs—such as changing the values of variables outside that sub-program—are considered poor practice, they are not prohibited by the language.

Object-oriented programming (OOP), typical of languages like C++, Ada, and Java, is a relatively new addition to the world of programming paradigms. It is characterized by the concept of encapsulating, or packaging, data items together with the methods or functions that manipulate those data items. In this paradigm, side effects are explicitly managed by controlling access to the data and methods in a particular grouping. The major theme in true OOP is that "everything is an object." You will see MATLAB exhibiting many traits of OOP as you work through this book, but you will not need to use this programming paradigm.

2.2 Basic Data Manipulation

In order to use MATLAB to demonstrate basic data manipulation, we begin with an exercise in starting and stopping the MATLAB system.

2.2.1 Starting and Stopping MATLAB

Exercise 2.1 shows you how to start and stop the MATLAB user interface. We will soon see the details of all the program's windows. For the moment,

Exercise 2.1 Starting and stopping MATLAB

If you have not installed MATLAB on your computer yet, follow the directions that came with your license for performing and testing the installation.

To start MATLAB, double-click on its icon. In the Interactions window you should see the MATLAB prompt (>>), which tells you that the MATLAB system is waiting for you to enter a command.

To exit MATLAB, type `exit` at the MATLAB prompt, choose the menu option File > Exit, or click the close icon (x) in the upper-right corner of the screen.

however, we will interact with MATLAB by typing instructions in the large Command window that occupies the left side of your screen.

2.2.2 Assigning Values to Variables

The concept of assigning values to variables is the first challenge facing novice programmers. The difficulty arises because many programming languages (including MATLAB) present this simple concept in a syntax that is very similar to conventional algebra, but with significantly different meaning. Consider, for example, the following algebraic expression:

```
z = x + y
```

In normal algebra, this is a two-way relationship that is an identity for the duration of the problem. If you knew the values of z and x, you could derive the value of y with no further analysis. To a programmer, however, this statement has a different meaning. It means that you want to sum the values given to the variables x and y, and store the result in a variable called z. If either x or y is unknown at the time of executing this statement, an error ensues. In particular, this relationship is true *only for this statement*. The relationship can be revoked in the next instruction, which might be:

```
z = 4*x - y
```

In algebra, this pair of statements collectively constrains the values of x, y, and z. In programming, the only significance is that the programmer decided to calculate the current value of z differently. A few computer languages are sensitive to this dilemma and use a different symbol for assigning values to a variable. For example, in Pascal or Ada, an instruction to assign the value `z = x + y` would be written as follows:

```
z := x + y
```

The "`:=`" operator clearly indicates that this is an assignment statement, not an algebraic identity.

Variable names: In general, variable names may contain any combination of uppercase and lowercase alphabetic letters, numbers, and the special

Exercise 2.2 Assigning variables

When you start MATLAB, you should see the prompt '>> ' in the Interactions window. This is your invitation to type something. Text that you should type will be shown like this throughout this book:

```
>> radius = 49
```

Note that all entries in the Interactions window terminate with the Enter key. The system response will be shown like this:

```
radius =
    49
>>
```

This response indicates that the value 49 has been stored in a variable named `radius`. To retrieve the value of `radius`, you just type its name and press Enter.

```
>> radius
ans =
    49
```

This response shows that the value 49 has been retrieved. Since you didn't specify where to put this result, it was stored in a default variable named `ans`.

characters _ (underscore) and $ (dollar). The underscore character is frequently used to represent a space in a variable name because spaces are not allowed. However, variable names may not begin with a numeric character, and even though the names may be hundreds of characters long, the first 64 characters must be unique. Exercise 2.2 demonstrates the assignment of values to variables.

Style Points 2.1

1. Some early versions of the FORTRAN and Basic languages severely restricted the number of characters you could use for variable names. It is no longer necessary to program as if you were still in the "bad old days." Choose names for variables that describe their content. For example, a variable used to store the velocity of an object should be named `velocity_in_feet_per_second` rather than `v`.

2. Since the space character is not permitted in variable names, there are two conventions for joining multiple words together to make a single variable name. One uses the underscore character to separate the words (`file_size`), and the other capitalizes the first letter of additional words (`fileSize`). You should choose one convention and be consistent with it. You cannot use a hyphen to concatenate words—MATLAB treats the name `file-size` as the arithmetic operation subtracting the value of the variable `size` from the value of the variable `file`.

2.2.3 Data Typing

It is important to understand how MATLAB treats the data stored in a variable. Different languages take varying approaches to this problem, and languages in general fall into two broad categories: untyped and typed. In general, interpreted languages like Lisp, Forth, Python, and MATLAB determine the type of data contained by a variable based on the type of data being stored there. Such languages are referred to as **untyped languages**. Each assignment

statement is presumed to be correct. If the variable already exists, both its type and value are reassigned; if it did not exist before, a new variable is created. Exercise 2.3 illustrates the effect of performing simple mathematical operations in MATLAB. By putting 49 into the variable `radius`, you established its type as numeric and enabled it to be used in normal arithmetic operations. Character strings are specified by including arbitrary characters between single quote marks. These have the type `char`, and must be handled differently, as discussed fully in Chapter 6. When you stored a character string in the variable `radius`, adding 1 to it did not cause an error in MATLAB as it would in some other languages, because addition is actually defined for character strings. It just did something radically different—it actually converted the individual characters to numbers and then added 1!

While this ability to assign data types dynamically is good for interpreted languages, it has two undesirable consequences that are really hard to unravel as the program runs:

- Typographical errors that misspell variable names in assignment statements cause new variables to be declared unintentionally and without the user noticing the error
- Logical errors that assign incompatible data to the same variable can cause obscure runtime errors

Typed languages require that programmers declare both the name and type of a variable before a value can be assigned to it. With this information, a compiler can then do a better job of ensuring that the programmer is not using a variable in an unintended way. Typed languages fall into two categories: weak typing and strong typing. If programmers decide to use only the normal data types, such as `double` and `char` as we saw above, this

Exercise 2.3 Performing basic mathematical operations

Make the following entries in the Interactions window. You should see the responses as shown here:

```
>> radius = 49
radius =
    49
>> radius + 1
ans =
    50
>> radius = 'radius of a circle'
radius =
   radius of a circle
>> radius + 1
ans =
  115 98 101 106 118 116 33 112 103 ...
```

is known as **weak typing** and is the usual approach to typing. In some extreme circumstances, programmers may choose to be more restrictive and define specific data types with a limited set of permitted interactions. This is called **strong typing**. For example, programmers might define the following data types, all of which are actually of type `double`: `meters`, `seconds`, and `meters_per_second`. The compiler would then be provided with a set of rules specifying the legal relationships between these types. For example, assignments can only be made to a variable of type `meters_per_second` from another variable of the same type, or by dividing a variable of type `meters` by a variable of type `seconds`.[1]

2.2.4 Classes and Objects

This section discusses two different attributes of a variable: its type and its value. In Section 2.2.2 you saw that a variable is a container for data, whose **value** is determined by what is assigned to the variable. In Section 2.2.3 you saw that by making that assignment to a variable, MATLAB also infers the **type** of data stored in that variable. You will see that while MATLAB is an untyped language, the programs you write will behave differently if applied to data of different types. For example, the type `double` specifies the form and expected behavior of a number. Adding 1 to a variable of class `double` containing 4 will, as expected, produce the result 5. Similarly, the type `char` is intended to hold a single character. Adding 1 to a `char` variable containing the value `'d'` will produce the numerical equivalent of the character `'e'`. MATLAB refers to the type of data in a variable as its **class**, and the value contained in the variable at any time as an **object**, an instance of that class. So in the operation:

```
this_number = 42.0
```

the variable `this_number` would be defined (if it didn't already exist); its class would be set to `double`, the inherent type of a floating point number; and its value to `42.0`. So the word `double` corresponds to a type definition or `class`, while the variable `this_number` is a variable of that type, which is an instance of that class or, in programming terms, an `object`.

2.3 MATLAB User Interface

MATLAB uses several display windows (see Figure 2.1). The default view includes a large Command window on the left, and stacked on the right are the Files, Workspace, and Command History windows. The tabs near the

[1] Before rushing to judge on the pickiness of this approach, note that this would have avoided the loss in 1999 of the Mars Climate Orbiter, which crashed into Mars because one group of programmers used English units while another used metric.

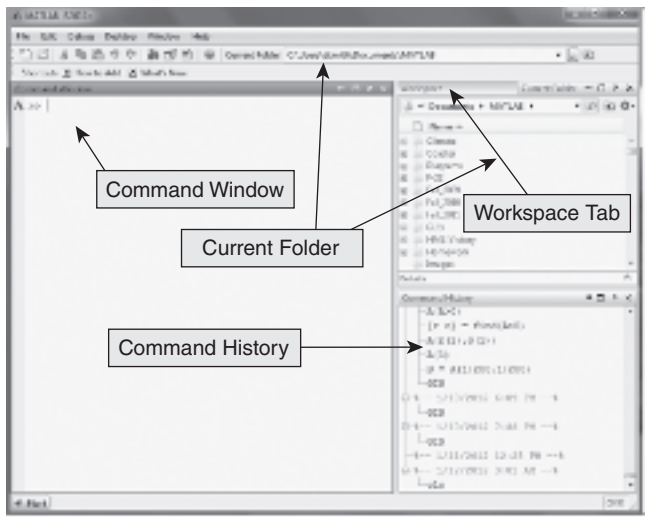

Figure 2.1 *The MATLAB default window configuration*

Hint 2.1

If you are using MATLAB, you can customize how your initial window will display. If you make a mistake and close an essential window, you can always restore the default configuration by choosing Desktop > Desktop Layout > Default.

Hint 2.2

When you make a mistake, you cannot easily correct it as you would in a word processor. The Interactions window really is functioning like a calculator, performing one instruction at a time exactly as you specify them. When you enter the command, it is immediately executed, regardless of whether that is what you intended. MATLAB offers several ways to correct erroneous commands. One way is to use the arrow keys on your keyboard. The up and down arrows let you move through the list of commands you have executed. Once you find the appropriate command, you can edit it and then press Enter to execute your new version.

middle of the windows on the right indicate which views are layered in that particular window. Selecting a tab will bring that view to the top. Other windows, such as an Editor window or a Figure window, will automatically open when needed.

2.3.1 Command Window

You can use MATLAB in two modes: Command mode, which is useful if you need instant responses to specific MATLAB commands, and Edit mode, in which practical solutions are developed. When working in Command mode, we use the Command window, which offers an environment similar to a scientific calculator. This window lets you save any values you calculate, but you cannot permanently save the commands used to generate those values. You will see in the next section how to use the Editor window to create and execute a text file of commands as the first step to unleashing the full programming capability of the language. The Command window is useful for performing quick experiments to discover the effects of different commands in MATLAB before embedding them in a larger program. You can perform calculations in the Command window much like doing calculations on a scientific calculator. Most of the

Exercise 2.4 Using the Interactions window

To compute the value of 5^2, type this command:

```
>> 5^2
```

The following output will be displayed:

```
ans =
     25
```

To find the cosine of π, type:

```
>> cos(pi)
```

which results in the following output:

```
ans =
-1
```

syntax is even the same. Exercise 2.4 shows how you might use the Command window to test two simple calculations.

Notice that in both of the examples in Exercise 2.4, MATLAB echoes the result as if it were saved in a variable called `ans`. This is the default variable used to save the result of any calculation you perform in the Command window that is not specifically assigned to another variable. Notice also the use of one of MATLAB's many built-in functions, `cos(...)`, that compute the cosine of an angle in radians, and of the built-in constant `pi`.

2.3.2 Command History

The Command History window records the commands you issued in the Command window in chronological sequence. When you exit MATLAB or when you issue the `clc` (Clear Commands) instruction, the commands listed in the Command window are cleared. However, the Command History window retains a list of all the commands you issued. You can clear the Command History using the Edit menu if you need to by selecting Edit and then Clear Command History. If you entered the sample commands in Exercise 2.4, notice that they are repeated in the Command History window. This window lets you review previous MATLAB sessions, and you can transfer the commands to the Command window by copying and pasting. Exercise 2.5 demonstrates the use of the Command History window. You will find the Command History window useful as you perform more and more complicated calculations in the Command window.

Hint 2.3

As a security precaution, if you use MATLAB on a public computer, you can set its defaults to clear the Command History window when you exit MATLAB or when you log off the computer.

Exercise 2.5 Using the Command History window

In the Interactions window, type:

```
>> clc
```

This should clear the Interactions window but leave the data in the Command History window intact. You can transfer any command from the Command History window to the Interactions window by double-clicking it (which also executes the command) or by clicking and dragging the line of code into the Interactions window. Try double-clicking:

```
cos(pi)
```

This should result in the following display in the Interactions window:

```
ans =
     -1
```

Now click and drag `5^2` from the Command History window into the Interactions window. The command won't execute until you press the Enter key, and then you'll get the following result:

```
ans =
     25
```

2.3.3 Workspace Window

The Workspace window keeps track of the variables you have defined as you execute commands in the Command window. As you have seen in the exercises so far, because you have not created other variables yet, the Workspace window should just show one variable, `ans`. The columns in the window display the name of the variable, its current value, and an entry in the class column (see Figure 2.2). In this case, the variable `ans` has a value of `25` and is a `double` array. Actually, even a single number you would usually consider a scalar is a 1×1 array to MATLAB. Exercise 2.6 shows how to obtain more information about a particular variable. Figure 2.2 shows the normal Variable window display for the variable `ans`. Figure 2.3 shows that the variable `ans` is a 1×1 array, uses 8 bytes of memory, and is an object of class `double`.

In Exercise 2.7, variable A has been added to the Workspace window, which lists variables in alphabetical order. Variables beginning with capital

Exercise 2.6 Showing more details in the Workspace window

Set the Variables window to show more about the variable `ans` by right-clicking on the bar with the column labels. On the drop-down menu, check the boxes next to Size and Bytes, so that these will display in addition to Name,Value, and Class. Your Variables window should now look like Figure 2.3.

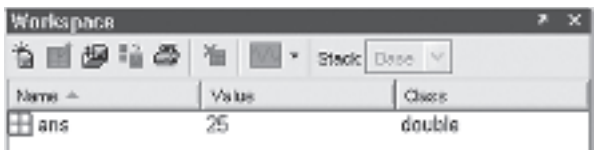

Figure 2.2 *The Workspace window*

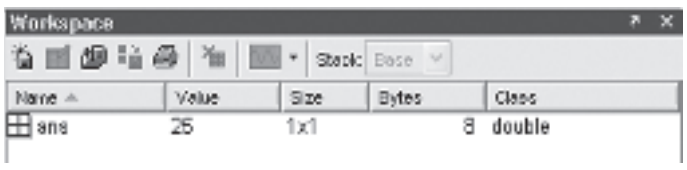

Figure 2.3 *Additional information in the Workspace window*

letters are listed first, followed by variables starting with lowercase letters, as shown in Figure 2.4.

Exercise 2.8 added the variable B to the Workspace window, and in Figure 2.5 you can see that its size is a 1×4 array.

You define two-dimensional arrays in a similar fashion. Semicolons are used to separate rows, as illustrated in Exercise 2.9. As you can see in

Exercise 2.7 Defining other variables

You can define additional variables in the Interactions window, and they will be listed in the Variables window. For example, type:

```
>> A = 5
```

This returns:

```
A =
     5
```

Exercise 2.8 Creating a vector

Entering matrices is not discussed in detail in this section. However, you can enter a simple one-dimensional matrix by typing:

```
>> B = [1, 2, 3, 4]
```

This returns:

```
B =
  1 2 3 4
```

The commas are optional. You would see the same result from:

```
>> B = [1 2 3 4]
```

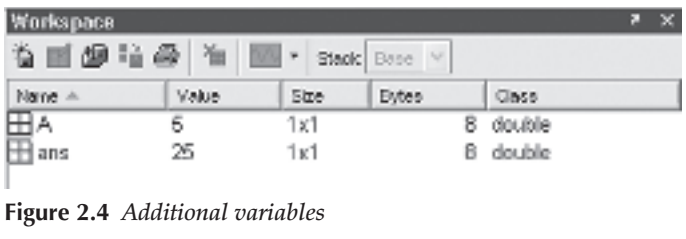

Figure 2.4 *Additional variables*

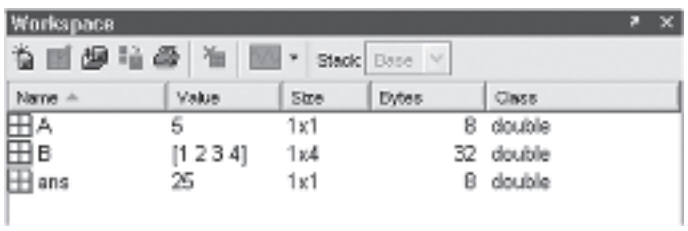

Figure 2.5 *Vector added in the Workspace window*

Figure 2.6, variable `c` appears in the Workspace window as a 3 × 4 array. Vectors and arrays are discussed fully in Chapter 3.

Note:

MATLAB presents numerical results in the following default format: if the value is an integer, there are no decimal places presented; but if there is a fractional part, four decimal places appear. You can change this by using the format command. See MATLAB help for details.

You can recall the values for any variable by just typing in the variable name, as shown in Exercise 2.10.

If you prefer to have a less cluttered desktop, you can close any of the windows (except the Command window) by clicking the x in the upper-right corner of each window.

Exercise 2.9 Creating a 3 × 4 matrix

```
>> C = [ 1 2 3 4; 10 20 30 40; 5 10 15 20]
```

returns:

```
C =
     1    2    3    4
    10   20   30   40
     5   10   15   20
```

Now, enter

```
>> C = [1 2 3 4; 10 20 30 40];
```

You will see the value of C change in the Variables window, but not echoed in the Interactions window. The semicolon on the end of the line suppresses presentation of the result.

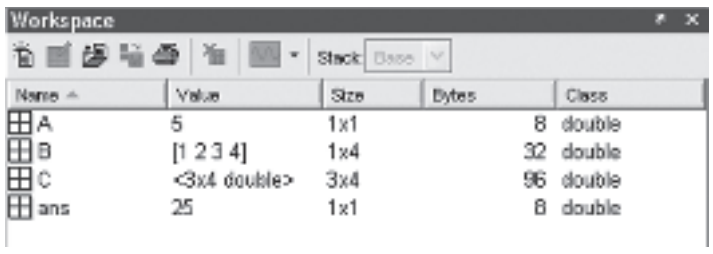

Figure 2.6 *Array added in the Workspace window*

Exercise 2.10 Recalling values for variables

```
>> A
```

returns:

```
ans =
5
```

You can also personalize which windows you prefer to keep open by selecting View from the menu bar and checking the appropriate windows. If you suppress the Workspace window, you can still find out what variables have been defined by using the commands `who` or `whos`. The command `who` lists the variable names, and `whos` lists the variable names together with their size and class. Exercise 2.11 illustrates this capability.

2.3.4 Current Directory Window

When MATLAB accesses files from and saves information to your hard drive, it uses the current directory. The default for the current directory depends on your version of the software and how it was installed. The current directory is listed at the top of the main window (see Figure 2.7). This can be changed by selecting another directory from the drop-down list to the right of the current directory name, or by browsing through your

Exercise 2.11 Using the `whos` command

```
>> whos
```

You should see the following display in the Command window:

```
Name    Size    Bytes    Class
A       1x1         8     double array
B       1x4        32     double array
C       3x4        96     double array
ans     1x1         8     double array
Grand total is 18 elements using 144 bytes
```

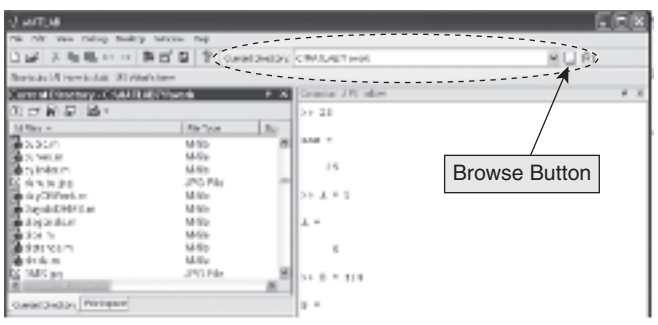

Figure 2.7 *The current directory*

computer files using the browse button located to the right of the drop-down list (circled in Figure 2.7).

2.3.5 Variable Editor

Double-clicking on any variable listed in the Workspace window automatically launches a Variable Editor window. Values stored in the variable are displayed in a spreadsheet-like format. You can change values in the Variable editor, or you can add new values.

2.3.6 Figure Window

A Figure window is created automatically when a MATLAB command requests a graph. Exercise 2.12 guides you through creating a graph. The MATLAB window opens automatically (see Figure 2.8). Any additional graphs you create will overwrite the plot in the current Figure window unless you specifically command MATLAB to open a new Figure window with the `figure` command. If you are using MATLAB version R2008a or newer, the first time you open a Figure window, a pop-up window appears with links to information about brushing and linking. As with the use of the Variable Editor, this

Hint 2.4

It is generally considered to be poor practice to edit the values of data by hand. A more rigorous approach would be to change the script that generated the data, thereby making the data changes repeatable.

Exercise 2.12 Creating a graph

```
>> x = [ 1 2 3 4 5];
```

A new variable, x, appears in the Workspace window.

```
>> y = (x-3).^2;
```

To create a graph, use the `plot` command:

```
>> plot(x,y)
```

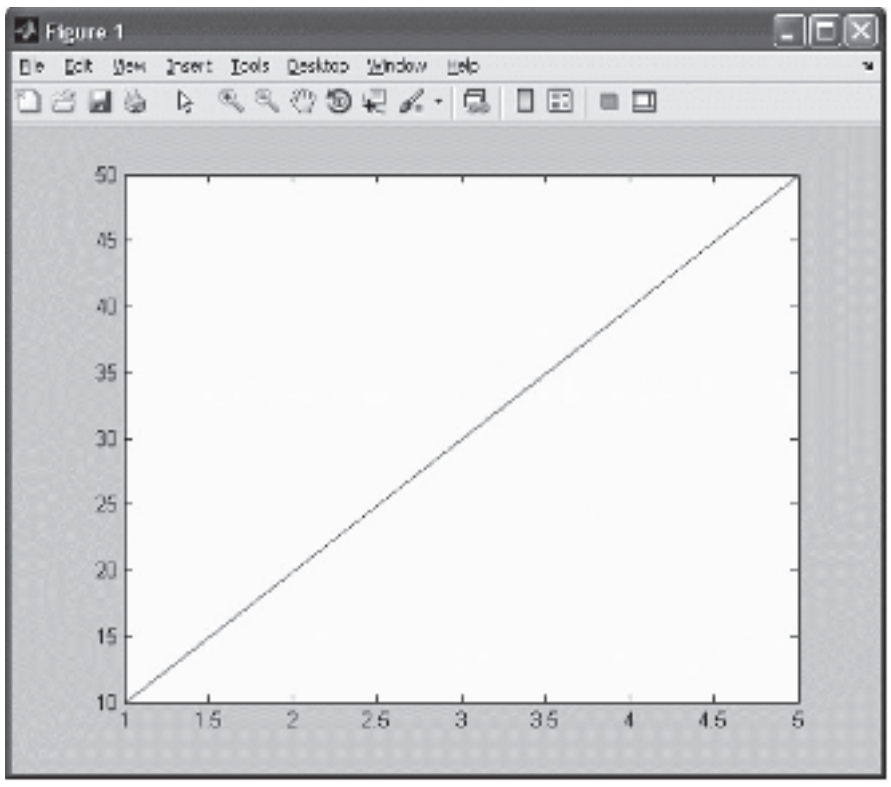

Figure 2.8 *A Graphics window*

is not the best way to modify data. See Hint 2.4. You can permanently hide this window by clicking the X at its right side.

MATLAB makes it easy to modify graphs by adding titles, *x* and *y* labels, multiple lines, and more with MATLAB built-in commands. Details of these commands will be presented in Chapter 11.

2.3.7 Editor Window

MATLAB provides a text editor, enabling you to create or modify text files that run in the Editor window. The Editor window is opened by choosing File > New > M-File. This window lets you type and save a series of commands without executing them. You can also open the Editor window by double-clicking a file name in the Current Directory window or by typing:

```
>> edit <file_name>
```

in the Command window, where `<file_name>` is the name of the file you want to open. You can open multiple files at the same time, using the

tabbed overlays to identify the files. An asterisk appears on the tab with the file name to indicate that the file has been modified since it was saved. Options under the Window menu let you organize the multiple files in various ways that make more than one file visible at once. When closing the Editor window, MATLAB displays a dialog asking if you need to save any changed files.

2.4 Scripts

This section describes the basic mechanism for creating, saving, and executing scripts as m-files. Building script files lets you save and reuse program statements without retyping them in the Command window.

2.4.1 Text Files

MATLAB uses text files as a permanent means of saving scripts (sets of instructions) rather than just entering commands in the Command window. As you will see in Chapter 8, text files are streams of characters stored sequentially with "markers" that indicate the end of each line of text. For now, think of a script much like writing an e-mail message—a number of lines of text written in a "smart" editor. The MATLAB Editor uses various techniques to help you format commands in these files.

2.4.2 Creating Scripts

A MATLAB script consists of a combination of executable instructions that MATLAB interprets and comment statements that help readers understand the script. You create **comments** by putting a percent sign (%) in the text file. MATLAB will ignore all text from that mark to the end of the current line. The MATLAB Editor colors all such comments green to distinguish them from the executable instructions. Most applications that use files specify a particular file name extension (the characters after the period in the file name) to identify how the text files will be used. MATLAB uses the extension `.m`, and the script files are often referred to as **m-files**. You create a new script file either by choosing File > New > M-File or by clicking the new file icon on the far left of the tool bar. The MATLAB Editor will then open a blank file in which you can enter the commands and comments of your script.

Import Note: Because MATLAB treats the names of .m files much like variable names, the names of your files must follow the same rules as those for variables in Section 2.2.2 above.

Hint 2.5

We began this first script with two commands: `clear` and `clc`. Every script should have these two commands (and later, also `close all`) before its first executable instruction. However, to avoid repetition, we will not include these commands in subsequent listings in this test.

Exercise 2.13 Creating a script

In this exercise, you will create a script derived from the Pythagorean theorem to compute the hypotenuse of a right triangle:

$$H^2 = A^2 + B^2$$

where A and B are the sides adjacent to the right angle, and H is the hypotenuse opposite. Open a new script file and type the commands shown in Listing 2.1 (don't type the accompanying line numbers—they will be automatically displayed).

Try creating the script described in Exercise 2.13 and shown in Listing 2.1.

Listing 2.1 Script to solve for the hypotenuse

```
1. clear
2. clc
3. A = 3;   % the first side of a triangle
4. B = 4;   % the second side of a triangle
5. hypSq = A^2 + B^2;   % the square of the
                        % hypotenuse
6. H = sqrt(hypSq)      % the answer
```

In Listing 2.1:

Line 1: Instructs MATLAB to delete all variables in your working directory.

Line 2: Instructs MATLAB to clear the Command window. Any text that now appears in the Command window will be the result of running this script, not the result of previous activities.

Lines 3–4: Assign values to A and B. The semicolons prevent MATLAB from displaying the result in the Command window; the percent sign begins the legible comment. Lines may contain nothing but a comment.

Line 5: Intermediate results with suitable names sometimes improve the legibility of the algorithm.

Line 6: Invokes the built-in library function `sqrt(...)` to compute the final result.

2.4.3 The Current Directory

After you have entered a script, you must name it and save it in a directory. MATLAB will need to find that directory—its working directory—in order to run the script. By default, MATLAB expects scripts to be stored in the

working directory, displayed in the tool bar at the top of the MATLAB main window. The specific path will vary with your version of MATLAB. However, the Current Directory window circled in Figure 2.7 always shows the default location when MATLAB starts. If you decide to store your scripts elsewhere, you will need to redirect MATLAB to that directory by typing it into the Current Directory window or using the browse button to the right of the display.

Once script files are saved in your working directory, you can edit them again by selecting and opening them with the MATLAB Editor. To open them, either use the File > Open menu command or double-click the file name in the Current Directory window. Before you close MATLAB, you should save the file created in Exercise 2.13.

2.4.4 Running Scripts

After you have built and saved a script, you can run it using any of the following methods:

- Type the name of the script in the Command window.
- Choose the Debug > Run menu item in the MATLAB Editor window.
- Press the F5 key when the script is visible in the editor. Doing this saves the script automatically before executing it.

The latest versions of MATLAB will echo the file name in the Command window when you invoke the script by the latter two methods. After you execute the script, the trace output is written to the Command window as if you had typed the script instructions there one at a time. For practice, run the script created in Exercise 2.13.

2.4.5 Punctuating Scripts

Many programming languages put a semicolon (;) at the end of a line to indicate the end of a command. Since the MATLAB language uses the end of a line to indicate the end of a command, it does not require an end-of-command character. If a long command needs to be extended to the next line for convenience in viewing the program, three periods, frequently referred to as **ellipses**, must be placed at the end of the line to continue the script.

The MATLAB language uses the semicolon for a different purpose. By default, all assignment commands display their results in the Command window in text form. For complex programs, the volume of this output can become too large. Whenever you really don't want to see all that output, putting a semicolon character at the end of a line will prevent the results of that assignment from displaying in the Command window.

Style Points 2.2

1. When writing scripts, you should invest some time to add comments. Comments make the scripts easier to understand as you are developing them, and make it more likely that you will be able to reuse the script later. Note: The listings included in this text will not have an appropriate level of commenting a. to save space and b. because they are explained in detail in the text.

2. Scripts should be written incrementally—build a little, test a little—rather than writing a whole script and then trying to find out where in that pile of code you made the mistake(s).

Common Pitfalls 2.1

You will quickly become accustomed to understanding the general flow of your script by observing the assignment statements reported in the Interactions window. However, especially if you have programmed in a language that requires semicolons at the end of a command, you may inadvertently put semicolons in your script. These will suppress the presentation of results and could mislead you into believing that a specific set of instructions has not been executed.

2.4.6 Debugging Scripts

MATLAB provides extensive debugging capabilities based on the use of **break points**, which are places in your program where you want to stop and verify that the code is doing what you expect. You insert break points as you edit a code segment by clicking the small dash between the line number and the start of the text. If the program is ready to run, a red dot appears in place of the dash where you clicked. If the file has been changed and hasn't been saved, the dot will be gray, in which case you should save the file. You can set any number of break points throughout your code.

After you start running a program, when MATLAB reaches a break point, execution stops, an arrow overwrites the break point symbol, and you can examine the contents of the variables either in the Workspace window or by passing the mouse slowly over the variable in the Editor window. A Debugging tool bar is available with icons that let you:

- Continue executing the logic from this point (other break points may come into effect)
- Step over the logic in this line to the next line in this code block
- Step into any modules referenced by this line of code
- Step out of this current code block

Use the script from Exercise 2.13 to practice inserting break points.

2.5 Engineering Example—Spacecraft Launch

In 1996, the Ansari X Prize was offered for the first time for a private venture: a reusable spacecraft. The requirements were for the same vehicle to carry three people into outer space twice in a two-week time period. The competition was won in 2004 by Tier 1, a company led by Burt Rutan. Their concept was to have a mother ship take off and land on a conventional runway carrying Space Ship One (see Figure 2.9). The spacecraft would be launched at 25,000 feet altitude and would reach outer space (an altitude of 100 km), then glide back and land on the same runway. They repeated this within a week, and they won the prize.

Figure 2.9 *Space Ship One*

Problem:
Assuming that the spacecraft uses all its fuel to achieve a vertical velocity u at 25,000 feet, what is the value of u for the spacecraft to reach outer space?

Style Points 2.3

Notice that when presented in this manner, the "inner values" like cm and inch cancel to ensure that the conversions are consistent.

Solution:
There are two parts to this problem: converting units to the metric system, and choosing and solving an equation for motion under constant acceleration (the rocket motor is no longer burning).

1. ***Convert the launch altitude from feet to meters.*** I like to remember as few numbers as possible. I do remember that 1 inch = 2.54 cm, so we will use this in a MATLAB script to find the conversion from feet to meters. The appropriate chain of calculations is this:

$$meters = feet \times \frac{meters}{cm} \times \frac{cm}{inch} \times \frac{inch}{feet}$$

Listing 2.2 shows the beginning of the script to solve this problem.

In Listing 2.2:

Lines 1–3: Define general knowledge with meaningful variable names to enable subsequent use of these values without ambiguity.

Listing 2.2 Script to compute the spacecraft's velocity (Part 1)

```
1. cmPerInch = 2.54;    % general knowledge
2. inchesPerFt = 12;    % general knowledge
3. metersPerCm = 1/100; % general knowledge
4. MetersPerFt = metersPerCm * cmPerInch * inchesPerFt;
5. startFt = 25000; % ft - given
6. startM = startFt * MetersPerFt;
```

Line 4: The conversion factor we need. Notice that because the variable names are consistent with the logic, they help to avoid errors.

Lines 5–6: Develop the initial conditions with suitable units.

2. ***Find and solve the equation.*** Given the following:
   - Initial and final altitudes from which you can compute the distance traveled: s
   - The motion is under constant acceleration, the force of gravity: g
   - To just reach outer space, the final velocity, v, is 0
   - The initial velocity, u, is needed

So after some diligent head scratching, we remember the equation of motion under constant acceleration connecting u, v, s, and a is:

$$v^2 = u^2 + 2as$$

However, this is not yet in a useful form. For computers to be able to solve an equation, you need the unknown quantity on the left of the equation and everything known on the right. Since u is the unknown, we move this to the left side of the assignment, and organize the known quantities to the right. These are the final velocity, v (i.e., 0) the given distance, s, and the acceleration, a. Since the positive direction for u and s is upward, but gravity is downward, we must use $a = -g$, and the equation can be transformed to:

$$u = \sqrt{2gs}$$

With this information, you can now solve this problem. Listing 2.3 shows the rest of Listing 2.2 to complete this calculation.

Listing 2.3 Script to complete the computation of the spacecraft's velocity

```
1. g = 9.81; % m/sec^2
2. top = 100; % km - given
3. s = (top*1000) - startM; % m
4. initialV = (2*g*s)^0.5 % the final answer
```

In Listing 2.3:

Line 1: The standard value for the acceleration due to gravity.
Line 2: The altitude of outer space is given in the problem statement.
Line 3: Computes the distance traveled, including the unit conversion from kilometers to meters. Note the optional, and in this case unnecessary, use of parentheses to define the order of operations.
Line 4: The final computation. The operator ^ is the MATLAB expression for exponentiation; x^y in MATLAB results in computing x^y. Notice that the parentheses are required here to force the multiplication to happen before the exponentiation.

Although most modern computing environments, including MATLAB, have tools that actually solve symbolic equations, these tools are not appropriate for an introduction to programming and will not be discussed in this book.

Chapter Summary

This chapter presented some fundamental notions of computing and introduced you to the nature of MATLAB, its user interface, and the fundamental tools for making programs work.

- Abstraction lets you refer to collections of data or instructions as a whole
- An algorithm is a set of instructions at an appropriate level of abstraction for solving a specific problem
- A data class describes the type of data and the nature of operations that can be performed on that data
- An object is a specific instance of a class with specific values that can be assigned to a variable
- The Command window lets you experiment with ideas by entering commands line-by-line and seeing immediate results
- The Command History window lets you review and recall previous commands
- The Workspace window lists the names, values, and class of your local variables.
- The Current Directory window lists the current files in the directory to which MATLAB is currently pointed
- A Document window opens when a variable in the Workspace window is selected, to let you view and edit data items
- A Figure window presents data and/or images when invoked by programs
- The Editor window lets you view and modify text files
- Scripts provide the basic mechanism for implementing solutions to problems

Special Characters, Reserved Words, and Functions

Special Characters, Reserved Words, and Functions	Description	Discussed in This Section
`'abc'`	Single quotes enclose a literal character string	2.2.3
`%`	A percent sign indicates a comment in an M-file	2.4.2
`;`	A semicolon suppresses output from assignment statements	2.4.5
`...`	Ellipses continue a MATLAB command to the next line	2.4.5
`=`	The assignment operator assigns a value to a memory location; this is not the same as an equality test	2.2.2
`ans`	The default variable name for results of MATLAB calculations	2.3.1
`clc`	Clears the Command window	2.3.2
`clear`	Clears the Workspace window	2.4.2
`sqrt(x)`	Calculates the square root of `x`	2.4.2

Self Test

Use the following questions to check your understanding of the material in this chapter:

True or False

1. A bag of groceries is an example of abstraction.
2. An algorithm is a series of logical steps that solves one specific problem.
3. It is impossible to write a complete, practical program in any paradigm other than procedural.
4. To be useful to an algorithm, the result of every computation must be assigned to a variable.
5. In programming, if you know the values of `z` and `x` in the expression `z = x + y`, you can derive the value of `y`.
6. Untyped languages are free to ignore the nature of the data in variables.
7. Anything assigned to be the value of a variable is an object.
8. Class is a concept restricted to object-oriented programming.
9. You can permanently save the commands entered in the Command window.

10. Double-clicking an entry in the Command History window lets you rerun that command.
11. You can manually change the values of variables displayed in the Workspace window.
12. You double-click a file name in the Current Directory window to run that script.
13. A Document window lets you view and edit data items.
14. MATLAB permits multiple Figure windows to be open simultaneously.
15. An asterisk on the File Name tab in the Editor window indicates that this is a script that can be executed.
16. MATLAB echoes comments entered in a script in the Command window.
17. When the name of script is typed in the Command window, it will be saved if necessary before it is executed.

Fill in the Blanks

1. _____ means expressing a quality apart from a particular implementation.
2. _____is a sequence of instructions for solving a problem.
3. Without _____, a programming solution can be mathematically proven to be correct.
4. Variable names must not begin with _____.
5. Armed with both the _____and _____of a variable, a compiler can do a better job of ensuring that the programmer isn't misinterpreting data.
6. An instance of a _____ is a(n)_____that is usually stored in a variable of that _____.
7. You can _____ in the Command window in a manner similar to the way you _____ on a scientific calculator.
8. You _____an entry in the Command History window to _____that command.
9. The columns in the Workspace window show the _____of the variable, its _____, and its _____.
10. You _____ the name of a file in the Current Directory window to edit that file.

11. A Document window opens automatically when you _____ a(n) _____ in the Workspace window.
12. Graphics windows are created when a(n) _____requests a graph.
13. You create comments by putting a(n) _____in the text file.
14. MATLAB will _____all text from the comment mark to _____.

Programming Projects

1. You are given two sides of a triangle, `a = 4.5` and `b = 6`. The angle between them is 35 degrees. Write a script to find the length of the third side and the area of the triangle.
2. In the bottom of the ninth inning, the bases are loaded and the Braves are down by three runs. Chipper Jones steps to the plate. Twice he swings and misses. The crowd heads for the exits. The next pitch is a fast ball down the middle. He swings and makes perfect contact with the ball, sending it up at a 45-degree angle toward the fence 400 ft away.
   a. Write a script to determine how fast he must hit the ball to land at the base of the fence, neglecting the air resistance.
   b. Perform a brief experiment to determine whether there was a better angle at which to hit the ball so that it could clear a 12 ft fence.
3. If an ice cream cone is 6 inches tall, and its rim has a diameter of 2 inches, write a script to determine the weight of the ice cream that can fit in the cone, assuming that the ice cream above the cone is a perfect hemisphere. You may neglect the thickness of the cone material. Assume that a gallon of ice cream weighs 8 lb and occupies 7.5 cubic feet.
4. Write a script that validates the relationship between `sin` θ, `cos` θ, and `tan` θ by evaluating these functions at suitably chosen values of θ.
5. I like my shower to remain hot for hours at 100°F, but am too cheap to buy one of those on-demand hot water systems. I don't care how slowly the water runs. The water supply is at 50°F, and the water heater is rated at 50,000 BTU/hour. Write a script to compute the maximum flow rate of my shower (in cubic feet per minute) that keeps the water temperature above 100°F.
6. It takes an average of 45 horsepower to run an electric car at an average speed of 35 mph. Write a script to compute the electrical

storage capacity of the battery system that would make this car practical for a 25-mile commute, recharging the batteries only at home at night when the electricity is cheap. How many D cell alkaline batteries would be needed for this?

7. You want to buy a $300,000 home with 20% down payment. The current compound interest rate is 4.5%.
   a. Write a script to determine:
      - the monthly payments for a 30-year loan,
      - the equivalent simple interest rate,
      - the total interest paid over the life of the loan.
   b. Now, repeat the computation for a 15-year loan at 5%. Is this a better deal?

8. The distance from my house to my office is 1.5 miles. Every morning, I have to decide whether to take the bus that averages (once it arrives) 25 mph, or to walk. I can walk at 4 mph. Write a script to determine how frequently the buses should run to give them a 50% chance of getting me to the office faster than walking.

9. A glass has the shape of a truncated cone of height 5 inches. Its top diameter is 3.5 inches, and its base diameter is 2 inches. If water is poured into the glass at 2 gallons per minute, write a script to calculate how long it takes to fill the glass to the brim. One gallon is 7.5 cubic feet.

10. You can calculate the aerodynamic drag on an object by the formula:

$$\text{Drag} = \tfrac{1}{2}\,\rho\,V^2\,C_d\,S$$

The air density, ρ, is 1.3 kg/m3 and the value of the drag area, C_dS, is a measure of the resistance of the object as it moves through the air. An object falling through air reaches terminal velocity when the aerodynamic drag equals the object's weight.

A sky diver weighing 80 kg has a $C_d\,S$ value of 0.7 when horizontal with arms and legs extended, and 0.15 when head down with arms and legs in line. One diver jumps from a plane at an altitude of 5,000 m in the horizontal position. After 20 sec, another diver jumps. Write a script to determine how much time the second diver must spend head down in order to catch up to the first diver. Also compute the height above the ground where they first meet. For simplicity, you may assume that the sky divers immediately reach their terminal velocity when jumping.

11. You are given a circle with radius 5 centered at `x = 1, y = 2`. You want to calculate the intersection of some lines with that circle. Write a script to find the `x` and `y` coordinates of both points of intersection. You should test this code at least with these lines:

```
y = 2x - 1
y = -2x - 10
y = x + 5.9054
```