CHAPTER 3

MATLAB BASICS

WHAT WILL WE LEARN?

- What is MATLAB and why has it been selected to be the tool of choice for this book?
- What programming environment does MATLAB offer?
- What are M-files?
- What is the difference between MATLAB scripts and functions?
- How can I get started with MATLAB?

3.1 INTRODUCTION TO MATLAB

MATLAB (MATrix LABoratory) is a data analysis, prototyping, and visualization tool with built-in support for matrices and matrix operations, excellent graphics capabilities, and a high-level programming language and development environment.

MATLAB has become very popular with engineers, scientists, and researchers in both industry and academia, due to many factors, among them, the availability of rich sets of specialized functions—encapsulated in *toolboxes*—for many important areas, from neural networks to finances to image processing (which is the main focus of our interest and will be discussed in Chapter 4).[1]

MATLAB has an extensive built-in documentation. It contains descriptions of MATLAB's main functions, sample code, relevant demos, and general help pages. MATLAB documentation can be accessed in a number of different ways, from command-line text-only help to hyperlinked HTML pages (which have their online counterpart in MathWorks's web site: http://www.mathworks.com).

MATLAB's basic data type is the *matrix* (or *array*). MATLAB does not require dimensioning, that is, memory allocation prior to actual usage. All data are considered to be matrices of some sort. Single values are considered by MATLAB to be 1×1 matrices.

The MATLAB algorithm development environment provides a command-line interface, an interpreter for the MATLAB programming language, an extensive set of numerical and string manipulation functions, 2D and 3D plotting functions, and the ability to build graphical user interfaces (GUIs). The MATLAB programming language interprets commands, which shortens programming time by eliminating the need for compilation.

If you want to get started with MATLAB right away, this is a good time to try *Tutorial 3.1: MATLAB—A Guided Tour* on page 44.

3.2 BASIC ELEMENTS OF MATLAB

In this section, we present the basic elements of MATLAB, its environment, data types, and command-line operations.

3.2.1 Working Environment

The MATLAB working environment consists of the following:

- *MATLAB Desktop*: This typically contains five subwindows: the *Command Window*, the *Workspace Browser*, the *Current Directory Window*, the *Command History Window*, and one or more *Figure Windows*, which are visible when the user displays a graphic, such as a plot or image.
- *MATLAB Editor*: This is used to create and edit M-files. It includes a number of useful functions for saving, viewing, and debugging M-files.
- *Help System*: This includes particularly the *Help Browser*, which displays HTML documents and contains a number of search and display options.

In *Tutorial 3.1: MATLAB—A Guided Tour* (page 44), you will have a hands-on introduction to the MATLAB environment.

[1]In 2003, MathWorks released the *Image Acquisition Toolbox*, which contains a collection of functions for image acquisition. This toolbox is briefly described in Section 5.3.2 and will not be explored in more detail in this book.

TABLE 3.1 MATLAB Data Classes

Data Class	Description
`uint8`	8-Bit unsigned integers (1 byte per element)
`uint16`	16-Bit unsigned integers (2 bytes per element)
`uint32`	32-Bit unsigned integers (4 bytes per element)
`int8`	8-Bit signed integers (1 byte per element)
`int16`	16-Bit signed integers (2 bytes per element)
`int32`	32-Bit signed integers (4 bytes per element)
`single`	Single-precision floating numbers (4 bytes per element)
`double`	Double-precision floating numbers (8 bytes per element)
`logical`	Values are 0 (false) or 1 (true) (1 byte per element)
`char`	Characters (2 bytes per element)

3.2.2 Data Types

MATLAB supports the data types (or data classes) listed in Table 3.1.

The first eight data types listed in the table are known as *numeric* classes. A numeric value in MATLAB is of class `double` unless specified otherwise. Conversion between data classes (also known as *typecasting*) is possible (and often necessary). A *string* in MATLAB is simply a $1 \times n$ array of characters.

3.2.3 Array and Matrix Indexing in MATLAB

MATLAB arrays (vectors and matrices) are indexed using a 1-based convention. Therefore, `a(1)` is the syntax to refer to the first element of a one-dimensional array `a` and `f(1,1)` is the syntax to refer to the first element of a two-dimensional array, such as the top left pixel in a monochrome image `f`. The colon (`:`) operator provides powerful indexing capabilities, as you will see in Tutorial 3.2 (page 46).

MATLAB does not restrict the number of dimensions of an array, but has an upper limit on the maximum number of elements allowed in an array. If you type

```
[c, maxsize] = computer
```

variable `maxsize` will contain the maximum number of elements allowed in a matrix on the computer and version of MATLAB that you are using.

3.2.4 Standard Arrays

MATLAB has a number of useful, built-in standard arrays:

- `zeros(m,n)` creates an `m` × `n` matrix of zeros.
- `ones(m,n)` creates an `m` × `n` matrix of ones.
- `true(m,n)` creates an `m` × `n` matrix of logical ones.

- `false(m,n)` creates an `m` × `n` matrix of logical zeros.
- `eye(n)` returns an $n \times n$ *identity matrix*.
- `magic(m)` returns a *magic square*[2] of order `m`.
- `rand(m,n)` creates an `m` × `n` matrix whose entries are pseudorandom numbers uniformly distributed in the interval [0, 1].
- `randn(m,n)` creates an `m` × `n` matrix whose entries are pseudorandom numbers that follow a normal (i.e., Gaussian) distribution with mean 0 and variance 1.

3.2.5 Command-Line Operations

Much of the functionality available in MATLAB can be accessed from the command line (as can be seen in the first steps of *Tutorial 3.2: MATLAB Data Structures* on page 46). A command-line operation is equivalent to executing one line of code, usually calling a built-in function by typing its name and parameters at the prompt (>>). If the line does not include a variable to which the result should be assigned, it is assigned to a built-in variable `ans` (as in `ans`wer). If the line does not end with a semicolon, the result is also echoed back to the command window.

Considering the large number of available built-in functions (and the parameters that they may take) in MATLAB, the command-line interface is a very effective way to access these functions without having to resort to a complex series of menus and dialog boxes. The time spent learning the names, syntax, and parameters of useful functions is also well spent, because whatever works on a command-line (one command at a time) fashion will also work as part of a larger program or user-defined function. Moreover, MATLAB provides additional features to make the command-line interaction more effective, such as the ability to easily access (with arrow keys) previous operations (and save typing time). Successful command-line operations can also be easily selected from the *Command History Window* and combined into a script.

3.3 PROGRAMMING TOOLS: SCRIPTS AND FUNCTIONS

The MATLAB development environment interprets commands written in the MATLAB programming language, which is extremely convenient when our major goal is to rapidly prototype an algorithm. Once an algorithm is stable, it can be compiled with the MATLAB compiler (available as an add-on to the main product) for faster execution, which is particularly important for large data sets. The MATLAB compiler MATCOM converts MATLAB native code into C++ code, compiles the C++ code, and links it with the MATLAB libraries. Compiled code may be up to 10 times as fast as their interpreted equivalent, but the speedup factor depends on how vectorized the original code was; highly optimized vectorized code may not experience any speedup at all.

[2]A *magic square* is a square array in which the sum along any row, column, or main diagonal results in the same value.

For faster computation, programmers may dynamically link C routines as MATLAB functions through the MEX utility.

3.3.1 M-Files

M-files in MATLAB can be *scripts* that simply execute a series of MATLAB *commands* (or *statements*) or can be *functions* that accept *arguments* (*parameters*) and produce one or more *output values*. User-created functions extend the capabilities of MATLAB and its toolboxes to address specific needs or applications.

An M-file containing a *script* consists of a sequence of commands to be interpreted and executed. In addition to calls to built-in functions, scripts may also contain variable declarations, calls to user-created functions (which may be stored in separate files), decision statements, and repetition loops. Scripts are usually created using a text editor (e.g., the built-in MATLAB Editor) and stored with a meaningful name and the `.m` extension. Once a script has been created and saved, it can be invoked from the command line by simply typing its name.

An M-file containing a *function* has the following components:

- *Function Definition Line*: It has the form
  `function [outputs] = function_name(inputs)`
  The keyword `function` is required. The output arguments are enclosed within square brackets, whereas the input arguments are enclosed within parentheses. If the function does not return any value, only the word `function` is used and there is no need for the brackets or the equal sign. Function names must begin with a letter, must not contain spaces, and be limited to 63 characters in length.
- *H1 Line*: It is a single comment line that follows the function definition line. There can be no blank lines or leading spaces between the H1 line and the function definition line. The H1 line is the first text that appears when the user types
  `>> help function_name`[3]
  in the Command Window. Since this line provides important summary information about the contents and purpose of the M-file, it should be as descriptive as possible.
- *Help Text*: It is a block of text that follows the H1 line, without any blank lines between the two. The help text is displayed after the H1 line when a user types `help function_name` at the prompt.
- *Function Body*: This contains all the MATLAB code that performs computation and assigns values to output parameters.[4]
- *Comments*: In MATLAB, these are preceded by the `%` symbol.

[3]Note that the prompt is different in the educational version of MATLAB (EDU>>).

[4]Even though the MATLAB programming language includes a "return" command, it does not take any arguments. This is in contrast with other programming languages that may use "return" followed by a parameter as a standard way of returning a single value or a pointer to where multiple values reside in memory.

Here is an example of a simple function (`raise_to_power`) that will be used in Tutorial 3.3 on page 53:

```
function z = raise_to_power(val,exp)
%RAISE_TO_POWER Calculate power of a value
% z = raise_to_power(val,exp) raise val to a power with value of exp
% and store it in z.

z = val ^ exp;
```

3.3.2 Operators

MATLAB operators can be grouped into three main categories:

- *Arithmetic Operators*: Perform numeric computations on matrices.
- *Relational Operators*: Compare operands.
- *Logical Operators*: Perform standard logical functions (e.g., AND, NOT, and OR)

Since MATLAB considers a matrix and its standard built-in data type, the number of array and matrix operators available in MATLAB far exceeds the traditional operators found in a conventional programming language. Table 3.2 contains a summary of them. All operands can be real or complex.

Table 3.3 shows a list of some of the most useful specialized matrix operations that can also be easily performed in MATLAB.

TABLE 3.2 MATLAB Array and Matrix Arithmetic Operators

Operator	Name	MATLAB Function
`+`	Array and matrix addition	`plus(a,b)`
`-`	Array and matrix subtraction	`minus(a,b)`
`.*`	Element-by-element array multiplication	`times(a,b)`
`*`	Matrix multiplication	`mtimes(a,b)`
`./`	Array right division	`rdivide(a,b)`
`.\`	Array left division	`ldivide(a,b)`
`/`	Matrix right division	`mrdivide(a,b)`
`\`	Matrix left division	`mldivide(a,b)`
`.^`	Array power	`power(a,b)`
`.^`	Matrix power	`mpower(a,b)`
`.'`	Vector and matrix transpose	`transpose(a)`
`'`	Vector and matrix complex conjugate transpose	`ctranspose(a)`
`+`	Unary plus	`uplus(a)`
`—`	Unary minus	`uminus(a)`
`:`	Colon	`colon(a,b)` or `colon(a,b,c)`

TABLE 3.3 Examples of MATLAB Specialized Matrix Operations

Name	MATLAB Operator or Function
Matrix transpose	Apostrophe (`'`) operator
Inversion	`inv` function
Matrix determinant	`det` function
Flip up and down	`flipud` function
Flip left and right	`fliplr` function
Matrix rotation	`rot90` function
Matrix reshape	`reshape` function
Sum of the diagonal elements	`trace` function

Since monochrome images are essentially 2D arrays, that is, matrices, all operands in Tables 3.2 and 3.3 are applicable to images. However, the MATLAB Image Processing Toolbox (IPT) also contains specialized versions of the main arithmetic operations involving images, whose main advantage is the support of integer data classes (MATLAB math operators require inputs of class `double`). These functions are listed in Table 3.4 and will be discussed in more detail in Chapter 6.

The relational operators available in MATLAB parallel the ones you would expect in any programming language. They are listed in Table 3.5.

TABLE 3.4 Specialized Arithmetic Functions Supported by the IPT

Function	Description
`imadd`	Adds two images or adds a constant to an image
`imsubtract`	Subtracts two images or subtracts a constant from an image
`immultiply`	Multiplies two images (element-by-element) or multiplies a constant times an image
`imdivide`	Divides two images (element-by-element) or divides an image by a constant
`imabsdiff`	Computes the absolute difference between two images
`imcomplement`	Complements an image
`imlincomb`	Computes a linear combination of two or more images

TABLE 3.5 Relational Operators

Operator	Name
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

TABLE 3.6 Logical Operators

Operator	Name
&	AND
|	OR
~	NOT

TABLE 3.7 Logical Functions

Function	Description
`xor`	Performs the exclusive-or (XOR) between two operands
`all`	Returns a 1 if all the elements in a vector are nonzero or a 0 otherwise. Operates columnwise on matrices
`any`	Returns a 1 if any of the elements in a vector are nonzero or a 0 otherwise. Operates columnwise on matrices

MATLAB includes a set of standard logical operators. They are listed in Table 3.6. MATLAB also supports the logical functions listed in Table 3.7 and a vast number of functions that return a logical 1 (true) or logical 0 (false) depending on whether the value or condition in their arguments is `true` or `false`, for example, `isempty(a)`, `isequal(a,b)`, `isnumeric(a)`, and many others (refer to the MATLAB online documentation).

3.3.3 Important Variables and Constants

MATLAB has a number of built-in variables and constants, some of which are listed in Table 3.8.

3.3.4 Number Representation

MATLAB can represent numbers in conventional decimal notation (with optional decimal point and leading plus or minus sign) as well as in scientific notation (using the letter `e` to specify a power-of-10 exponent). Complex numbers are represented using either `i` or `j` as a suffix for the imaginary part.

TABLE 3.8 Selected Built-In Variables and Constants

Name	Value Returned
`ans`	Most recent answer
`eps`	Floating-point relative accuracy
`i` (or `j`)	Imaginary unit ($\sqrt{-1}$)
`NaN` (or `nan`)	Not-a-number (e.g., the result of 0/0)
`Inf`	Infinity (e.g., the result of a division by 0)

All numbers are stored internally using the IEEE floating-point standard, resulting in a range of approximately 10^{-308}–10^{+308}.

3.3.5 Flow Control

The MATLAB programming language supports the usual flow control statements found in most other contemporary high-level programming languages: `if` (with optional `else` and `elseif`) and `switch` decision statements, `for` and `while` loops and the associated statements (`break` and `continue`), and the `try...catch` block for error handling. Refer to the online documentation for specific syntax details.

3.3.6 Code Optimization

As a result of the matrix-oriented nature of MATLAB, the MATLAB programming language is a *vectorized language*, which means that it can perform many operations on numbers grouped as vectors or matrices without explicit loop statements. Vectorized code is more compact, more efficient, and parallelizable.[5] In addition to using vectorized loops, MATLAB programmers are encouraged to employ other optimization tricks, such as preallocating the memory used by arrays. These ideas—and their impact on the execution speed of MATLAB code—are discussed in Tutorial 3.3 (page 53).

3.3.7 Input and Output

Basic input and output functionality can be achieved with functions `input` (to request user input and read data from the keyboard) and `disp` (to display a text or array on the screen). MATLAB also contains many support functions to read from and write to files.

3.4 GRAPHICS AND VISUALIZATION

MATLAB has a rich set of primitives for plotting 2D and 3D graphics. Tutorial 9.1 (page 188) will explore some of the 2D plotting capabilities in connection with the plotting of image histograms and transformation functions, whereas Tutorials in Chapter 11 will show examples of 3D plots in connection with the design of image processing filters in the frequency domain.

MATLAB also includes a number of built-in functions to display (and inspect the contents of) images, some of which will be extensively used throughout the book (and discussed in more detail in Tutorial 4.2 on page 74).

[5]Getting used to writing code in a vectorized way—as opposed to the equivalent nested for loops in a conventional programming language—is not a trivial process and it may take time to master it.

3.5 TUTORIAL 3.1: MATLAB—A GUIDED TOUR

Goal

The goal of this tutorial is to give a brief overview of the MATLAB environment.

Objectives

- Become familiar with the working environment in MATLAB.
- Learn how to use the working directory and set paths.
- Become familiar with the MATLAB help system.
- Explore functions that reveal system information and other useful built-in functions.

Procedure

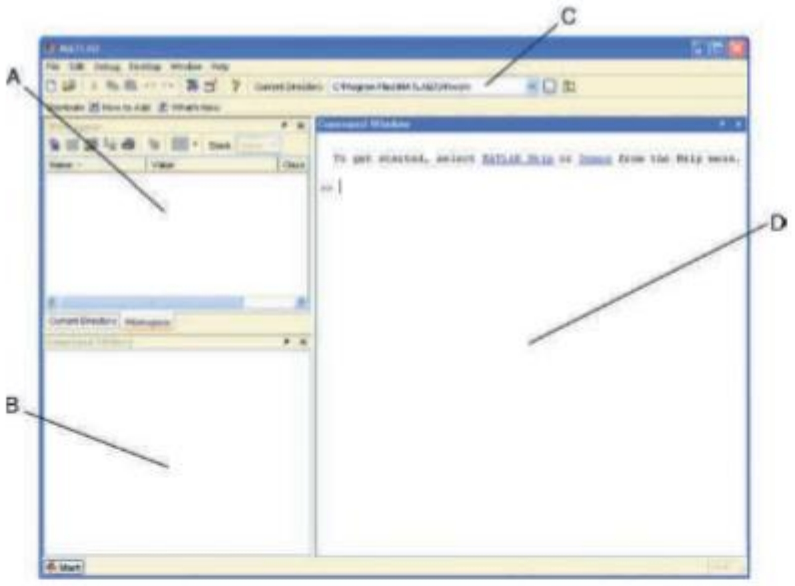
The environment in MATLAB has a simple layout, consisting of several key areas (Figure 3.1).[6] A description of each is given here:

- A: This pane consists of two tabbed areas: one that displays all files in your current *working directory*, and another that displays your *workspace*. The workspace lists all the variables you are currently using.
- B: This pane shows your *history* of commands.
- C: This is where you can modify your current *working directory*. To change the current directory, you can either type it directly in the text box or click on the button to select the directory. You can also change the working directory using the `path` command. See help documents for more information.
- D: This is the *command window*. Here you control MATLAB by typing in commands.

In order for you to use files such as M-files and images, MATLAB must know where these files are located. There are two ways this can be done: by setting the *current directory* to a specific location, or by adding the location to a list of set paths known to MATLAB. The *current directory* should be used for temporary locations or when you need to access a directory only once. This directory is reset every time MATLAB is restarted. To change the current directory, see description C above. Setting a path is a permanent way of telling MATLAB where files are located—the location will remain in the settings when MATLAB is closed. The following steps will illustrate how to set a path in MATLAB:

1. From the **File** menu, select **Set Path...**

[6] The exact way your MATLAB window will look may differ from Figure 3.1, depending on the operating system, MATLAB version, and which windows and working areas have been selected—under the *Desktop* menu—to be displayed.



To get started, select MATLAB Help or Demos from the Help menu.

```
>>
```

FIGURE 3.1 MATLAB environment.

The paths are presented in the list box in decreasing precedence order.

2. If the directory you wish to add has subfolders within it, then use the **Add with Subfolders...** button. If your directory only contains files, you can use the **Add Folder...** button. To change the precedence of your directory, use the **Move** buttons.
3. **Save** your changes and close the **Set Path** window.

The help system in MATLAB is very useful. It provides information about functions, such as syntax, descriptions, required and optional input parameters, and output parameters. It also includes a number of demos. The following steps will show you how to access help documents and navigate the help window.

4. To access the help system, type `doc` in the command window. Open the help system now.

The left pane of the help window displays a tree of help information. The right pane will show the help document that you choose. If you know exactly what function you need help for, you can use the `doc`, `helpwin`, or `help` commands to directly access the help information.

5. In the command window in MATLAB, execute the following commands to view the help document for the function `computer`.

```
help computer
helpwin computer
doc computer
```

Question 1 What is the difference between the commands `help`, `helpwin`, and `doc`?

There are several commands that can be used to gain information about the system. Explore them in this step.

6. Execute the following commands, one at a time, to see their function.

```
realmin
realmax
bitmax
computer
ver
version
hostid
license
```

Question 2 What is the difference between `ver` and `version`?

7. For some MATLAB humor, repeatedly type the command `why`.

3.6 TUTORIAL 3.2: MATLAB DATA STRUCTURES

Goal

The goal of this tutorial is to learn how to create, initialize, and access some of the most useful data structures available in MATLAB.

Objectives

- Learn how to use MATLAB for basic calculations involving variables, arrays, matrices, and vectors.
- Explore multidimensional and cell arrays.
- Review matrix operations.
- Learn how to use MATLAB structures.
- Explore useful functions that can be used with MATLAB data structures.

Procedure

1. Execute the following lines of code one at a time in the *Command Window* to see how MATLAB can be used as a calculator.

```
2 + 3
2*3 + 4*5 + 6*7;
```

Question 1 What is the variable `ans` used for?

Question 2 What is the purpose of using a semicolon (`;`) at the end of a statement?

2. Perform calculations using variables.

```
fruit_per_box = 20; num_of_boxes = 5;
total_num_of_fruit = fruit_per_box * num_of_boxes
```

Question 3 Experiment with creating your own variables. Are variables case sensitive?

Question 4 What is the value/purpose of these variables: `pi`, `eps`, `inf`, `i`? Is it possible to overwrite any of these variables? If so, how can it be undone?

3. Execute the commands `who` and `whos`, one at a time, to see their function and the difference between them.

There are several commands that will keep the MATLAB environment clean. Use them whenever you feel your command window or workspace is cluttered with statements and variables.

4. Clear a variable in the workspace. After execution, note how the variable disappears from the workspace.

```
clear fruit_per_box
```

5. Clear the command window and all variables with the following lines of code (one at a time to see their effects individually).

```
clc
clear all
```

6. Create a 3×3 matrix by executing the following line of code.

```
A = [1 2 3;4 5 6;7 8 9]
```

Question 5 What is the use of the semicolon in this statement?

The Colon Operator

7. A very useful operator in MATLAB is the colon (`:`). It can be used to create a vector of numbers.

```
1:5
```

8. A third parameter determines how to count between the starting and ending numbers. It is specified between the start and end values.

```
1:1:5
5:-1:1
1:2:9
9:-2:1
```

Question 6 Write a statement that will generate a vector of values ranging from 0 to π in increments of $\pi/4$.

9. The colon operator can also be used to return an entire row or column of a matrix.

```
A = [1 2 3;4 5 6;7 8 9]
A(:,1)
A(1,:)
```

Question 7 Write a line of code that would generate the same 3×3 matrix as in the variable `A` above, but using the colon operator to generate the sequence of numbers in each row instead of explicitly writing them.

10. The colon operator can be replaced with the function `colon`, which performs the same operation.

```
colon(1,5)
```

As seen in the steps above, creating a vector of evenly spaced numbers is easily done with the colon (`:`) operator when we know where the vector starts, ends, and how large the space in between each value is. In certain cases, we may wish to create a vector of numbers that range between two numbers, but we only know the quantity of values needed (for example, create a vector that consists of 4 values between $\pi/4$ and π) . To do this, we use the `linspace` function.

11. Execute this command to see how the function `linspace` operates.

```
linspace(pi/4,pi,4)
```

12. Compare the result from the previous step with these values.

```
pi/4
pi/2
3*pi/4
pi
```

Special Built-In Matrices

MATLAB has several built-in functions that will generate frequently used matrices automatically.

13. Execute the following lines of code one at a time.

```
zeros(3,4)
ones(3,4)
ones(3,4) * 10
rand(3,4)
randn(3,4)
```

Question 8 What is the difference between the functions `rand(M,N)` and `randn(M,N)`?

Matrix Concatenation

Concatenation of matrices is done with brackets (`[]`) or using the `cat` function. Take, for example, the statement

```
A = [1 2 3;4 5 6;7 8 9]
```

The brackets are combining three rows. Instead of explicitly defining each row all at once, they can be defined individually as vectors and then combined into a matrix using brackets.

14. Combine the three individual vectors into a 3 × 3 matrix.

```
X = [1 2 3]; Y = [4 5 6]; Z = [7 8 9];
A = [X;Y;Z]
B = cat(1,X,Y,Z)
```

Similarly, the brackets can be used to delete a row of a matrix.

15. Delete the last row (row 3) of the matrix `A`. Note how the colon operator is used to specify the entire row.

```
A(3,:) = []
```

A vector with *N* elements is an array with one row and *N* columns. An element of a vector can be accessed easily by addressing the number of the element, as in `X(5)`, which would access the fifth element of vector `X`. An element of a two-dimensional matrix is accessed by first specifying the row, then the column, as in `X(2,5)`, which would return the element at row 2, column 5. Matrices of dimensions higher than 2 can be accessed in a similar fashion. It is relevant to note that arrays in MATLAB are 1-based—the first element of an array is assigned or accessed using 1, as opposed to 0, which is the standard in many programming languages.

16. Use the `ones` and `rand` functions to create multidimensional arrays.

```
A = ones(4,3,2);
B = rand(5,2,3);
size(A)
size(B)
disp(A)
disp(B)
```

Question 9 What does the `size` function do?

Question 10 What does the `disp` function do?

Operations Involving Matrices

Performing arithmetic operations on matrices can be achieved with the operators `+ - * /`. The default for the multiply (`*`) and divide (`/`) operators is matrix multiplication and matrix division. To perform arithmetic operations on individual elements of a matrix, precede the operator with a dot (`.`).

17. Perform matrix multiplication on two matrices.

```
X = [1 2 -2; 0 -3 4; 7 3 0]
Y = [1 0 -1; 2 3 -5; 1 3 5]
X * Y
```

18. Perform element-by-element multiplication.

```
X .* Y
```

19. Perform another matrix multiplication on two matrices.

```
X = eye(3,4)
Y = rand(4,2)
X * Y
Y * X
```

Question 11 Why did the last operation fail?

20. Use the `diag` and `trace` functions to perform operations on the diagonal of a matrix.

```
Y = rand(3,3)*4
Y_diag = diag(Y)
Y_trace = trace(Y)
```

Question 12 What does the `diag` function do?

Question 13 What does the `trace` function do?

Question 14 Write an alternative statement that would produce the same results as the `trace` function.

21. Calculate the transpose of a matrix.

```
Y
Y_t = Y'
```

22. Calculate the inverse of a matrix and show that $YY^{-1} = Y^{-1}Y = I$, where I is the identity matrix.

```
Y_inv = inv(Y)
Y * Y_inv
Y_inv * Y
```

23. Calculate the determinant of a matrix.

```
Y_det = det(Y)
```

Cell Array

As demonstrated earlier, a matrix is the fundamental data type in MATLAB. It resembles the classical definition of an array as a homogeneous data structure, that is, one in which all its components are of the same type. *Cell arrays*, on the other hand, are another type of array where each cell can be any data type allowed by MATLAB. Each cell is independent of another and, therefore, can contain any data type that MATLAB supports. When using cell arrays, one must be careful when accessing or assigning a value to a cell; instead of parentheses, curly braces (`{}`) must be used.

24. Execute the following lines of code one at a time to see how cell arrays are handled in MATLAB.

```
X{1} = [1 2 3;4 5 6;7 8 9];    %Cell 1 is a matrix
X{2} = 2+3i;                   %Cell 2 is complex
X{3} = 'String';               %Cell 3 is a string
X{4} = 1:2:9;                  %Cell 4 is a vector
X
celldisp(X)
X(1)
X{1}
```

Question 15 What does the `celldisp` function do?

Question 16 What does the percent (`%`) character do?

Question 17 What is the difference between the last two lines in the code above (`X(1)` as opposed to `X{1}`)?

There is another way to assign values to a cell array that is syntactically different, but yields the same results. Note in the next step how the cell index is enclosed within normal parentheses (`()`), but the data that will be saved to the cell is encapsulated by curly braces (`{}`).

25. Execute this line to see another way of assigning cell array values.

```
X(1) = {[1 2 3;4 5 6;7 8 9]};
```

26. The next few lines of code will demonstrate proper and improper ways of cell array assignment when dealing with strings.

```
X(3) = 'This produces an error'
X(3) = {'This is okay'}
X{3} = 'This is okay too'
```

Structures

Structures are yet another way of storing data in MATLAB. The syntax for structures is similar to that of other programming languages. We use the dot (`.`) operator to refer to different fields in a structure. Structures with identical layout (number of fields, their names, size, and meaning) can be combined in an array (of structures).

27. Create an array of two structures that represents two images and their sizes.

```
my_images(1).imagename = 'Image 1';
my_images(1).width = 256;
my_images(1).height = 256;
my_images(2).imagename = 'Image 2';
my_images(2).width = 128;
my_images(2).height = 128;
```

28. View details about the structure and display the contents of a field.

```
my_images(1)
my_images(2).imagename
```

29. Display information about the structure.

```
num_of_images = prod(size(my_images))
fieldnames(my_images)
class(my_images)
isstruct(my_images)
isstruct(num_of_images)
```

Question 18 What does the `fieldnames` function do?

Question 19 What does it mean when the result from the function `isstruct` is 1? What does it mean when it is 0?

Question 20 Use the help system to determine what function can be used to delete a field from a structure.

3.7 TUTORIAL 3.3: PROGRAMMING IN MATLAB

Goal

The goal of this tutorial is to explore programming concepts using scripts and functions.

Objectives

- Learn how to write and execute scripts.
- Explore the MATLAB Editor.
- Explore functions and how to write them.
- Learn the basics of loop vectorization.

What You Will Need

- `script3_3_1.m`
- `script3_3_2.m`
- `raise_to_power.m`

Procedure

Although the command window is simple and convenient to use, it does not provide a way of saving or editing your code. The commands stored in the *Command History* window can, however, be easily made into a script file. A script is a text file with the extension **.m** and is used to store MATLAB code. Scripts can be created and modified using the built-in editor.

1. To start a new script, navigate to **File > New > M-File**.

The MATLAB Editor may open in a separate window or it may be docked within the MATLAB environment, depending on how your environment was previously set up.

2. Open the file named `script3_3_1.m`. If the M-file is located in the current directory, you may also open it from the *Current Directory* listing by double-clicking the file name.

M-files are syntax color coded to aid in reading them. As you may have noticed, comments can be added to any script using two methods: percent (`%`) signs, or wrapping the comments with `%{` and `%}`. The second method is used for the header information in the script.

Question 1 Based on the script in file `script3_3_1.m`, what is the difference between using a percent (`%`) sign and using `%{` and `%}`?

To execute one or several lines of code in a script, highlight the code and press F9.[7]

3. Highlight all the code in `script3_3_1.m` and press F9 to execute the script.

The Cell Mode

A new editing mode (available in MATLAB 7.0 and above) called *cell mode* allows you to make minor adjustments to variables, and re-execute a block of code easily.

[7]Shortcut keys may vary for different versions of MATLAB for different operating systems. The shortcut keys listed in this book work for the PC version of MATLAB.

4. Open file `script3_3_2.m`.
5. Enable cell mode by navigating to **Cell > Enable Cell Mode** in the MATLAB environment.

With cell mode activated, you will note that the comments that are preceded with two percent signs (`%%`) are slightly bolder than the other comments. This corresponds to a *block title*. To create a cell block, all you need to do is give it a block title. Cell blocks allow you to modify and execute one block of code at a time. More than one cell block can exist in a single script.

6. Execute all the code in the script.

This script is an example of stretching the histogram of a monochrome image to the full dynamic range of allowed values, achieved by the `imadjust` function. This function also allows us to make gamma adjustments to the image. At this time, the concepts of histogram stretching and gamma correction are not important; they will be discussed in detail in Chapter 10. Instead, let us focus on how cell mode may be useful to us later on. By making small changes to the value of gamma, we can see the effects of gamma on the adjusted image. Cell mode makes this task easy.

7. Locate the line of code where the variable gamma is assigned a value of 1. Highlight *only* the value 1.

When cell mode is active, a new menu bar appears in the Editor. Among the icons, there are two text boxes that allow us to make adjustments to the value we just selected in the code. The first text box allows us to subtract and add, and the other allows us to divide and multiply the value we highlighted in the code.

8. In the add/subtract text box, type a value of 0.1 and press the plus (+) sign to the right.

Question 2 What happened to the value of the variable `gamma` in the code?

Question 3 In addition to the modification of code, what else happened when the plus button was pressed? What happens when it is pressed again?

Question 4 Other than the ones described in this tutorial, what features are available when cell mode is active?

Functions

Functions are also **.m** files that contain MATLAB code, but with the added ability of being able to be called from another script or function as well as receiving parameters. Open the `raise_to_power` function to see how the file is constructed.

9. Open `raise_to_power.m` in the Editor.

The M-file of a function contains all necessary information to call the function, execute it, and display its help documentation. The first line in the file defines the function with its input and output parameters. In the case of the `raise_to_power` function, it takes two parameters and returns one. Any comments immediately after the function definition are considered help documentation.

10. Ensure that the `raise_to_power.m` file is located in a directory that is part of either a set path or the current directory.
11. View the help information for the function by typing the following statement into the command window:

```
help raise_to_power
```

Question 5 Compare the help documentation with the comments in the M-file. How does MATLAB determine which comments are to be displayed for help and which are just comments in the function code?

Because we are using MATLAB for image manipulation, it is important to ensure that we are taking advantage of CPU and memory-efficient coding practices—particularly *loop vectorization*. Take, for example, the following block of pseudocode that could easily be implemented in any programming language. *NB*: Depending on your computer's speed, you may want to change the value of `MAX_CNT` in the code snippets below.

```
MAX_CNT = 10000
for i = 1 to MAX_CNT
   do x(i) = i ^ 2
```

Here, we are populating an array in which each element is the index of that element squared. The following MATLAB code implements the pseudocode above using typical programming techniques. In these examples, the `tic` and `toc` commands are used to calculate the execution time.

12. Implement the pseudocode above with the following statements.

```
tic
MAX_CNT = 10000
for i = 1:MAX_CNT
   x(i) = i ^ 2;
end
toc
```

We could greatly influence the speed of this loop by simply preallocating the memory used by the array. This is effective because every time we add data to the matrix, new memory must be allocated to hold the larger variable if there is no room for it. If the correct amount of memory is already allocated, then MATLAB only needs to alter the data in each cell.

13. Implement the previous loop, but with preallocation.

```
tic
MAX_CNT = 10000
x = zeros(1,MAX_CNT);
for i = 1:MAX_CNT
   x(i) = i ^ 2;
end
toc
```

Question 6 By what factor did preallocating the array x increase performance?

In MATLAB, this is still considered poor programming technique. MATLAB acts as an interpreter between the code you write and the hardware of your computer. This means that each time a statement is encountered, it is interpreted to a lower level language that is understood by the hardware. Because of this, loops, such as the one above, cause MATLAB to interpret the statement in the loop, however, many times the loop executes—in the case above, 10,000 times. Interpretation is slow compared to the speed of computer hardware. Since MATLAB operates on matrices natively, we can perform the same operation using *loop vectorization*, thus getting rid of the loop.

14. Implement a more efficient version of the loop.

```
tic
MAX_CNT = 10000
i = 1:MAX_CNT;
x = i .^ 2;
toc
```

Question 7 In the code above, why do we use `.^` instead of just `^`?

Question 8 How much faster is loop vectorization than our previous two implementations?

Here we do not need to explicitly tell MATLAB to perform the operation on each element because that is the nature of the MATLAB environment. Loop vectorization also takes care of the preallocation problem we saw in the first implementation.

Question 9 Consider the following pseudocode. Write a vectorized equivalent.

```
i = 0
for t = 0 to 2*pi in steps of pi/4
   do i = i + 1
      x(i) = sin(t)
```

Earlier it was mentioned that a script can be made from the *Command History* window. Now that we have entered some commands, we can see how this can be done.

15. To create a script from the *Command History* window, locate the last four statements entered. To select all the four statements, hold down the **ctrl** key and select each statement, and then right-click the highlighted statements, and select **Create M-File**.

WHAT HAVE WE LEARNED?

- MATLAB (MATrix LABoratory) is a data analysis, prototyping, and visualization tool with built-in support for matrices and matrix operations, excellent graphics capabilities, and a high-level programming language and development environment. It has been selected as the tool of choice for this book because of its ease of use and extensive built-in support for image processing operations, encapsulated in the Image Processing Toolbox.
- The MATLAB working environment consists of the *MATLAB Desktop*, the *MATLAB Editor*, and the *Help System*.
- M-files in MATLAB can be *scripts* that simply execute a series of MATLAB *commands* (or *statements*) or can be *functions* that can accept *arguments* (*parameters*) and produce one or more *output values*. User-created functions extend the capabilities of MATLAB and its toolboxes to address specific needs or applications.
- An M-file containing a *script* consists of a sequence of commands to be interpreted and executed much like a batch file. An M-file containing a *function*, on the other hand, contains a piece of code that performs a specific task, has a meaningful name (by which other functions will call it), and (optionally) receives parameters and returns results.
- The best ways to get started with MATLAB are to explore its excellent built-in product documentation and the online resources at MathWorks web site.

LEARN MORE ABOUT IT

There are numerous books on MATLAB, such as the following:

- Hanselman, D. and Littlefield, B., *Mastering MATLAB 7*, Upper Saddle River, NJ: Prentice Hall, 2005. Excellent reference book.
- Pratap, R., *Getting Started with MATLAB*, New York, NY: Oxford University Press, 2002. Great book for beginners.

ON THE WEB

There are many free MATLAB tutorials available online. The book's companion web site (http: //www.ogemarques.com/) contains links to several of them.

3.8 PROBLEMS

3.1 Using MATLAB as a calculator, perform the following calculations:

(a) 24.4×365

(b) $\cos(\pi/4)$

(c) $\sqrt[3]{45}$

(d) $e^{-0.6}$

(e) 4.6^5

(f) $y = \sum_{i=1}^{6} (i^2 - 3)$

3.2 Use the `format` function to answer the following questions:

(a) What is the precision used by MATLAB for floating-point calculations?

(b) What is the default number of decimal places used to display the result of a floating-point calculation in the command window?

(c) How can you change it to a different number of decimal places?

3.3 Initialize the following variables: $x = 345.88$; $y = \log_{10}(45.8)$; and $z = \sin(3\pi/4)$. Note that the MATLAB function `log` calculates the natural logarithm; you need to use `log10` to calculate the common (base-10) logarithm of a number.

Use them to calculate the following:

(a) $e^{(z^3 - y^{1.2})}$

(b) $x^2 - \sqrt{(4y + z)}$

(c) $\frac{x-y}{z+5y}$

3.4 Initialize the following matrices:

$$X = \begin{bmatrix} 4 & 5 & 1 \\ 0 & 2 & 4 \\ 3 & 4 & 1 \end{bmatrix}$$

$$Y = \begin{bmatrix} -7 & 6 & -1 \\ 3 & -2 & 0 \\ 13 & -4 & 1 \end{bmatrix}$$

$$Z = \begin{bmatrix} 0 & 7 & 1 \\ 0 & 2 & -5 \\ 3 & 3 & -1 \end{bmatrix}$$

Use them to calculate the following:

(a) $3X + 2Y - Z$

(b) $X^2 - Y^3$

(c) $X^T Y^T$

(d) XY^{-1}

3.5 What is the difference between the following MATLAB functions:

(a) `log` and `log10`

(b) `rand` and `randn`

(c) `power` and `mpower`

(d) `uminus` and `minus`

3.6 What is the purpose of function `lookfor`? Provide an example in which it can be useful.

3.7 What is the purpose of function `which`? Provide an example in which it can be useful.