Part I: MATLAB® Overview

Chapters 1 through 5 provide an introduction and overview of some usable MATLAB coding approaches and how to extend these approaches by developing increasingly sophisticated scripts. Topics such as basic syntax, plotting, curve fitting, working with data files, and image processing are explored. These topics will form the basic MATLAB tools that will be used to varying extents in Part II.

Chapter 1
Introduction to MATLAB

Scientists and engineers eventually find themselves in the position where they need to perform long or repetitive calculations in order to solve a problem. There are lots of great programming languages available for such purposes, and the task at hand determines which language will provide the best solution in the time available. Being able to work with several computational languages and programs is a valuable skill, but being able to perform rapid engineering calculations that provide visualization or graphical representations of the results is a real plus in any computational exercise.

Optical engineers and scientists use a wide range of sophisticated software and simulation packages ranging from ray-tracing programs that track light rays through an optical system to designing interference filters and much more. Having the ability to write and tune custom-built software for particular applications will have great appeal for many practitioners in the field. It will also be appreciated that the skills needed to develop and customize software for both calculation and visualization can be readily gained with modern software packages tailored for scientists and engineers. With the aim of developing such skills, we will build the capability of developing basic software programs before we move to applications of interest to optical engineers.

The choice of software for many engineers and scientists is MATLAB, which provides much of the functionality needed for scientific and engineering programming tasks. MATLAB is very easy to learn and has a wide array of calculation and graphical capabilities. It also has a wide user base, so many excellent books and tutorials on numerous aspects of science and engineering are available. Also, the program has been adapted for different types of users, so professional, educational, and student versions are available—getting started is easy and affordable! MATLAB is available through

The MathWorks, Inc.
3 Apple Hill Drive,
Natic, MA 01760-2098
United States
http://www.mathworks.com

MATLAB is a scripting language that can be used to generate very sophisticated programs. It supports visualization and complex numerical calculations as well as advanced programming approaches. A complete description of the capabilities of MATLAB is beyond the scope of this introductory chapter, but a quick review of the MathWorks website or any of the supporting websites will quickly bring you up to speed on the range of capabilities.

The next section provides a quick way to start using MATLAB. After that, we will begin to develop the skills needed to be effective with MATLAB programming.

1.1 Getting Started with MATLAB

The starting point for this book will be a working version of MATLAB with the MATLAB Integrated Development Environment (IDE) open in front of you. We will only be using the base MATLAB functionality, so no MATLAB Toolboxes® will be needed. MATLAB Toolboxes are software collections of standard functions and applications commonly used in particular topic areas such as image processing, signal processing, testing, and data acquisition, to name a few of the many available topics. Much more detail can be found on the MathWorks website.

The starting point for using MATLAB will appear as a screen similar to that shown in Fig. 1.1. MATLAB versions can be obtained for use on many computer operating systems, and there are small variations in the look and feel of the MATLAB interfaces; however, what you see in Fig 1.1 is fairly representative. Ideally, the default window arrangement will be showing when the MATLAB interface is opened, setting the location of the smaller windows or panels in the frame. If you have the MATLAB interface open but the smaller windows are in a different configuration, you can get to the default version by looking on the menu bar and locating the Layout button then choosing Default. As you become more familiar with MATLAB, you may change the layout to support your programming style and needs.

On the default MATLAB IDE[1–3] you will see the windows Current Folder, Command Window, and Editor. These are the windows we will be using most often to open, display, and edit files. The Editor window will appear whenever you are working with a file, so the first thing we need to do is open a file for editing. The tasks we will be performing have as a prerequisite a basic knowledge of your computer, keyboard, and mouse, and how to interact with files.

The Workspace panel provides a list of the active variables that have been entered and are available; Command History shows a list of the commands that have been entered; and the Details panel shows information about the highlighted script file in the Current Folder. The displayed information is controlled through the commented lines using '%%' notation; i.e., any line with the '%%' comment will be shown in the Details panel.

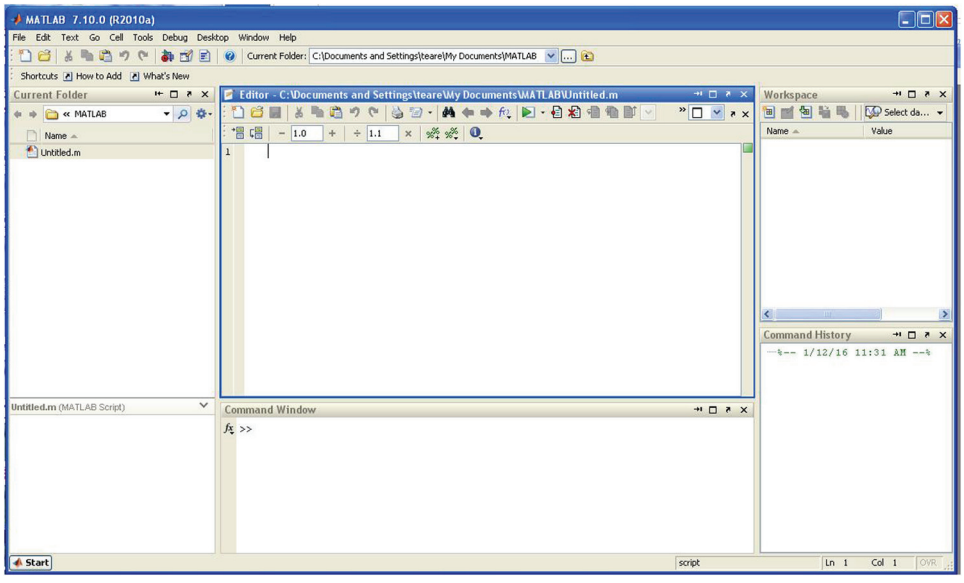

Figure 1.1 The MATLAB default interface at startup. Beginning with the panel in the upper left and continuing in the clockwise direction, we have the following panels: Current Folder, Editor, Workspace, Command History, Command Window, and Details.

Editing a file in Editor

The Current Folder shows the files that are in the directory as well as the location of, or path to, the files on your computer. This is where the files you create will be saved. Knowing this location is important when you want to locate the files and are working outside of the MATLAB IDE. Placing the mouse cursor into the Current Folder panel and right-clicking will give you a pulldown menu. Move the mouse cursor to New File and then to Script and left-click the mouse. You will see that a file called 'Untitled.m' is created. If you left-click on this file you will be able to rename it. Let's rename it as 'myFirstScript.m' and then double-click on it to open the file in Editor. If you click in the open window area of Editor, you will be able to type into this area. As you type, different kinds of text will have different colors, depending on whether you have entered a key word or a string, etc. Right-clicking in the Editor panel will show more-advanced controls.

Calculations in the Command Window

The Command Window is where the results of your work will be shown and where you can perform direct calculations. This can be demonstrated by placing the mouse cursor in the Command Window and left-clicking. Type: 8 – 4 and hit return. You will see the result 'ans=4'. You have made your first step in working with MATLAB calculations. We will build on this example type later in Section 1.4.

The MATLAB interface provides an incredible amount of control, configurability, and flexibility in how you program. But just these three panels, Current Folder, Command Window, and Editor, are enough to get started with MATLAB. The next sections provide an introduction to some useful commands for engineering and scientific calculations. There are many ways to do calculations in MATLAB, so consult with other users to see how they write their programs, and eventually you will find a methodology that works for you.

1.2 Anatomy of a Program

A program is a collection of commands organized to perform a particular task. These commands can be saved in a file to be used as needed. Much of the programming we will look at here is scientific programming, i.e., the use of computers and the development of programs to answer specific engineering and science questions. These programs often take a form that differs from enterprise-level programs designed for many users and long-term maintainability. Typically, the programs that we are writing will be very streamlined, and people who are unfamiliar with the particular program may not find them easy to use. While this may seem to be a limitation, quite often the writer of the program is the only user of that program, so this is not really a limitation!

One observation about scientific programs is that they are usually developed rather quickly, often by a single programmer. As a result, these programs have few features and are often aimed at answering a specific question; i.e., they are meant to be a single-use program. While this strategy can make sense, there is a significant advantage to writing programs that can be reused or can form the starting point for a new program. Starting with a previously developed program can greatly speed up the process of making a new program. With this in mind, writing a program that follows some basic structure or format and has sufficient comments or notes on how it works (as well as descriptions of what sections of the code mean) can significantly shorten the time needed to morph a piece of old code into a new program or allow the complete code to be reused later.

The structure that we will use for our programs is shown as a block diagram in Fig. 1.2, which presents a skeleton program that contains each of the major components often found in programs. Of course, not every program will fit into our simple model, but a surprising number of solutions to scientific problems will!

The block diagram shows the major components of a scientific program. All of the blocks of the program can be included in a single script and thus a single MATLAB m-file; however, it can be convenient to break the program into several sections that could equally well be contained in several files. While it is easy to write very complicated programs, there is considerable value in keeping programs as simple as possible, and breaking the program into several files can help. Thus, it is common for the overall program to be composed of several files with one of the files serving as the starting or entry

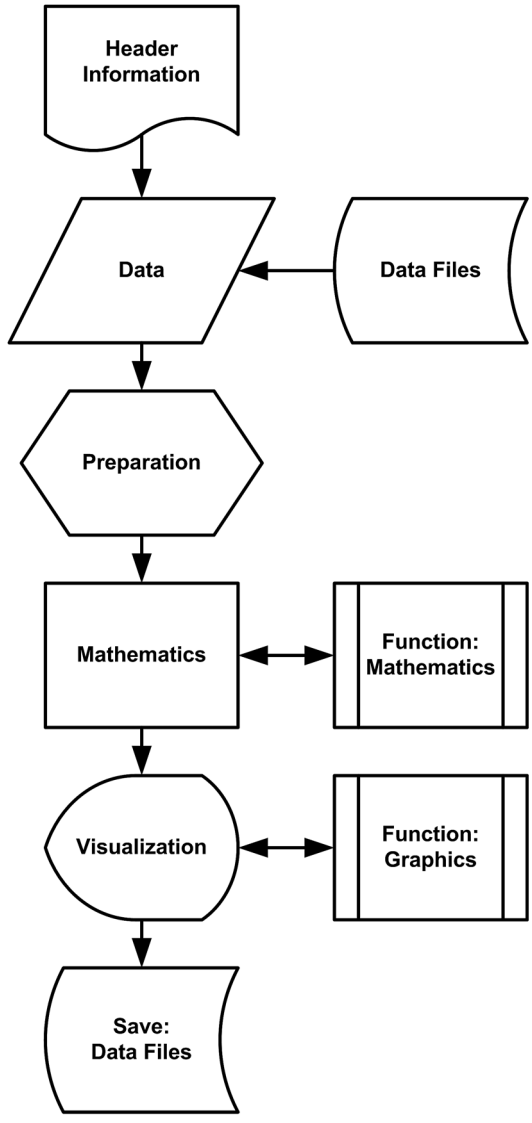

Figure 1.2 Block diagram of a scientific programming solution showing the various components.

point for running the complete program. In this book we will adopt the approach of having the initial program be a MATLAB *script*. This is not always the preferred approach, and some programmers will choose to use a MATLAB *function* for the initial file. Many new programmers find that this latter approach introduces unneeded complications that are not required for scientific programming, at least not currently in their investigation.

The program structure shown in Fig. 1.2 starts at the top with the Header Information block, which is used to record the filename of the program, the

author and date, and a complete description of the program. This block will commonly include the first programming commands that relate to initializing the program. The places where we move from one block to another indicate locations where effective comments will be most beneficial when looking back to the code and trying to remember what you were thinking at the time of writing it. In general, adding comments is always preferred, but in the locations *between* blocks, comments should be considered as *required*.

The Data and Data Files blocks are where any external data can be loaded or, alternatively, where initial data can be hard coded. There are merits to having both means of introducing data into the program. This is also a good location for setting up large arrays or other data structures that will be used. Data structures are ways of organizing data so that they can be accessed and used effectively.

The Preparation block is for preparing the data that will be used in the remainder of the program. This may include unit conversions, scaling, or making some composite number such as an area from radius data or other simple calculations. This block also is used for populating data structures if that is required.

The Mathematics block is where the data inputs are converted into the answer to the question of interest. This is where mathematical equations are entered, or functions can be called, as part of the calculation process. These functions can be part of MATLAB or can be from a previously user-written program. Having a collection of user-written functions is one of the easiest ways to reuse your own code.

For many users, one of the best reasons to use MATLAB is the wide range of visualization tools that are part of the software package. These tools are part of the Visualization block and include built-in functions that provide simple plots and can extend to, e.g., 3D rendering of very complicated systems.

The final block is for saving the calculated data. There are many good ways to save data in MATLAB, and we will explore some of the options in this book.

Now that we have looked at the overview of a program, we can begin to look at the individual commands that make up a program.

1.3 MATLAB Basic Functions and Operators

MATLAB has been designed to support the wide-ranging mathematical needs of scientists and engineers across many disciplines. This large user base has contributed an amazing quantity of code that has made it easier than ever to create better scripts and validate newly created algorithms. As a starting point, MATLAB scripts are made by chaining together data, functions, and

operators. As such, many of the "commands" used in MATLAB are really functions. With all of this in mind, it is time to introduce a subset of built-in functions that are particularly useful to new MATLAB programmers. These functions often have a form that is similar to commands in other programming languages and operate in a similar fashion. While not all-encompassing, the built-in functions[1,2] shown in Table 1.1 will allow you to start generating your own MATLAB programs. Notice the parenthesis that follow the function name; these indicate that there are arguments that can be included to tune the operation of the function.

The best way to understand how these built-in functions work is to try them out in the MATLAB IDE. There is excellent built-in "help" for the built-in functions;[1–3] however, as an attempt to cover all possibilities, the Help screens contain a great deal of information, which, admittedly, can be a little hard for first-time users to interpret.

MATLAB is case-sensitive; i.e., an upper-case (capitalized) letter means something different from a lower-case letter.[3] The built-in MATLAB functions are all written as lower-case letters. MATLAB does not require data types to be defined for a variable before using them. MATLAB determines from the context of the operation being performed, whether the variable is a string (a group of characters), an integer (a number without a decimal point), or a double (a number that has digits before and after the decimal point).

Much of scientific and engineering programming makes use of operators. The power of operators and the simplicity of their use make MATLAB a great choice as a programming language. Table 1.2 shows a few of the available operators.[1,2] These operators are common to many programming languages and will be somewhat familiar to most scientists and engineers. The next sections present the mechanics of using commands and operators to generate useable code.

Table 1.1 Some useful MATLAB built-in functions.

clear()	while()	mean()	disp()	plot()
clc()	if()	std()	fprintf()	axis()
clf()	elseif()	histogram()	sprintf()	xlabel()
figure()	for()	polyfit()	load()	ylabel()
grid()	abs()	polyval()	save()	hold()

Table 1.2 Some useful operators in MATLAB.

Addition	+	Transpose	′	NOT	~
Subtraction	−	AND	&	XOR	xor
Multiplication	*	OR	|	Greater than	>
Division	/	Equal	=	Less than	<

1.4 Simple Calculations using MATLAB

The best way to start learning a new programming language is simply to start working with it. The MATLAB functions and operations that will be used here are the most basic form of coding that can be written; no MATLAB Toolboxes will be used, and all of the calculations will work with the student versions. Our starting point will be the MATLAB IDE, the interface that comes up when you launch MATLAB. Until you are familiar with the IDE and how to configure the interface, it will be easiest to work with it in the "default" layout. There is lots of information on the screen when you are in the default layout, but it will all make sense once you are working with it. To start, locate the Command Window, where the results of calculations will appear and you can enter commands that can include MATLAB built-in functions, operators, or custom functions.

Let's get started with some simple mathematics to show how the Command Window can be used. Enter the following, including the prompt >> as follows:

```
>> 5 + 2
```

and press return. You should see

```
ans = 7
```

as expected.

The Command Window can perform simple mathematical operations. If you look in the Workspace window, you will see that the answer shown as `ans` has the value of 7. The Workspace window lets you keep track of the variable and calculated values. If you type "clear" in the Command Window, the values in Workspace will be removed, while the function `clc` will clear the Command Window.

Here's another example: enter the following sequence in the Command Window:

```
>> A = 10
>> B = 5
>> C = A + B
```

Compare what you typed to the values in the Workspace area. These variables will be available for additional calculations until they are cleared. Very long calculation strings can be entered into the Command Window, but doing this can be inconvenient, so MATLAB supports the ability to enter a

list of commands into a text file that can then be executed as a package. This is known as *scripting*, and in MATLAB, scripts have a file name of '<filename>.m', where '.m' is the file name extension. Scripting provides versatility to program development, and these programs can be extended to work like most any other program with windows, dialog boxes, file loading, and saving, to name a few.

MATLAB is the concatenation of the words "matrix" and "laboratory" and from its inception was developed to support numerical computation using arrays of numbers of various dimensions. A linear array of numbers can be written as a column or as a row, and is commonly referred to as a vector. A collection of vectors combined into a two-dimensional (2D) array is referred to as a matrix.[1] Here we will use the term *vector* to refer to a one-dimensional (1D) array and *matrix* to refer to a multidimensional array.

MATLAB is very good tool for working with matrices and vectors as well as all of the manipulations[4] that are common to these representations. If there are two vectors, A and B, defined as

A = [1 2 3] and B = [4 5 6], then
A + B = [5 7 9]
A – B = [–3 –3 –3]
A .* B = [4 10 18]
A ./ B = [0.25 0.40 0.50]
A / B = 0.4156

Note that if you try to run A * B you will get an error. In the above example, there is a '.' in front of the multiplication operator (and the division operator) that indicates element-by-element multiplication. Also, when we want to do matrix multiplication (or division), we need to make sure that the vectors are properly oriented. We can use the transpose operator (′) for this purpose:

$$A' = [1\,2\,3]' = [1;2;3] = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

A * B′ = 32

$$A' * B = \begin{bmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{bmatrix}$$

The matrix operations[1,4] in MATLAB can take a little getting used to, but they are very powerful. There are also operators that allow the matrix to be rearranged or reformatted; for example,

$$C = A' * B = \begin{bmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{bmatrix}$$

$$C(:)' = [4\ 5\ 6\ 8\ 10\ 12\ 12\ 15\ 18]$$

Without the transpose operator, C(:) would be a simple column matrix. Reforming matrices into column vectors can be very useful when using MATLAB functions that take a column vector as an input, such as when calculating the mean and standard deviation using `mean()` and `std()`, respectively, or generating a histogram using the `histogram()` function.

Recall that when using functions, it is useful to put the parenthesis after the function name, even if there is nothing inside the parentheses. As already mentioned, the parentheses indicate that the command is a function, and functions can take arguments as well as return results in specific formats.

A short example of the way to write these commands as scripts should clarify how to combine operators and functions to perform useful tasks, as will be seen in the next section.

1.5 Vectorization and Matrix Indexing

The real power of MATLAB lies in its ability to work with vectors and matrices, which is one of the driving strengths of the language. This ability to work with vectors has led to a program style known as *vectorization*. The basic idea of vectorization can be appreciated by considering that in many scientific programming languages calculation loops are created to step through a particular counter while calculating the value of a function for the particular point. Vectorization uses matrices and vectors directly, and through special operators it can calculate the result. This can appear similar to the calculations that would be seen in topics associated with linear algebra. The use of matrices as the basis for so many calculations has led to a great deal of sophistication in using and manipulating matrices. Several techniques for working with matrices will be introduced here that will be used throughout this book.

A matrix is a rectangular array of elements, and its size can be described by the number of rows and columns it comprises. Figure 1.3 illustrates how a matrix is represented in MATLAB as a single variable and is organized by element. The individual elements in a matrix can be selected by the matrix's row and column indices. The indices start with the value of 1 and can take nearly any practical range of rows and columns. A vector will have one of either the row or column values fixed at 1, and the other index will identify the length of the vector.

The group of elements in the matrix can also be considered as a plane of elements, and these planes can be stacked to make a hypercube of 2D data

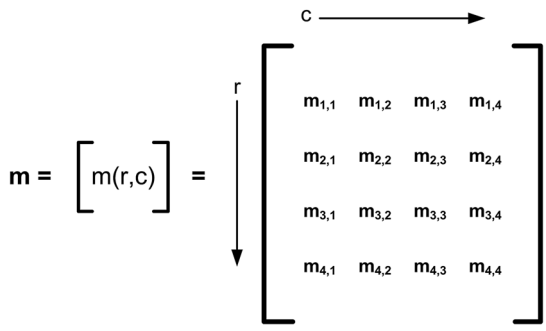

Figure 1.3 A variable m can be used in MATLAB to represent a matrix whose elements are addressed as m(r,c) for m(row,column). This matrix is a 4×4 element array of numbers.

planes, for which individual elements would be specified in the matrix **m** by adding a third index 'p', such as 'm(r,c,p)'.

Expanding these ideas into practical usage, a 9-element matrix M can be entered directly into MATLAB as M = [1 2 3; 4 5 6; 7 8 9]. The individual element 6 can be selected using M(2,3). The power of using MATLAB becomes evident when we want to select the middle row. The command `mR = M(2,1:3)` will select the middle row values from M and assign these values to the vector mR. Similarly, a new row could be added to M by writing M = [M; mR]. The various techniques for matrix manipulation and vectorization will be expanded on as we advance in programming skill.

1.6 MATLAB Scripts

MATLAB has all of the advantages of a scripting language,[1] and one such advantage is the ability to use code that is stored and read as text files. As we move on to exploring scripting, all that is needed is to copy the code examples included here into a MATLAB-compatible file, and the code will be ready to run. The discussion here will start from the point where you have MATLAB open on your computer and are ready to start writing scripts.

The MATLAB script form used here is the m-file—which more or less means that our filenames will end with '.m'—and our working example will be 'myProgram.m'. It is easiest to show by example, so we will be constructing our first program one step at a time. (An alternative way of writing these files would be to write them as independent functions, but we will save that for Section 1.7.)

Before we begin to write scripts, we will discuss a few formalities that will become more valuable as you begin to write longer programs. The first of these is adding comments into your coding projects. Comments are important to help you understand the purpose of a piece of code or a section within the code. The symbol used to identify a comment is the percent sign '%', which is the first symbol we will normally see in the very first line of a MATLAB

script. Comments are used for informing someone reading the code what the code is about. To get this discussion going, let's examine a segment of a script.

```
% myProgram.m
% SWT 9-4-15
% This is an example program to show how to begin a program!

% Housekeeping
clear; clc; % clears variables and command window
```

The code fragment above will become very familiar, as it will be used in nearly every MATLAB script included in this book. The content of the comments changes with each program and will provide insight into the operation of the script. The first three comment lines identify the name of the program, the author and date, and a short sentence about the purpose of the program, respectively. As programs become more complex, the description of the program may need to grow. It can be helpful to keep a list of the description revisions at the beginning of each file.

The `% Housekeeping` comment is just a way of saying that this area of the code is used to set the starting point for the rest of the programming activities. Notice the commands `clear` and `clc`. These ensure that the program starts running from a clean slate; i.e., variables and functions have been removed, and the Command Window is empty. To see the results of these commands, enter this information into a MATLAB script labeled 'myProgram.m'. The script can be run from the Editor window by pushing the green arrow. Basically, doing so clears the variable list and the Command Window; this is the starting point for the next programming steps.

It is important to choose the names of files carefully! Filenames must start with a letter followed by numbers, letters, or the underscore (_) character, as desired; be careful not to include spaces. Also, don't use file names of other MATLAB functions. For example, you may write a plotting program and be tempted to name your new program as 'plot.m'. Doing so could cause issues, as there is a function in MATLAB called `plot()`, and the program wouldn't know which program you are calling. To check whether an m.file name you are considering using is already taken, search 'help'.

1.7 MATLAB Functions

Scripts are a great way to write programs that can be used and reused with ease. Scripts can become long, i.e., contain a large number of calculations, some of which might be repeated several times in the code. It can be much more convenient to move the repeated code into a separate file and then call the separate script from the main script when needed, or to chain a number of scripts together. This approach is an easily implemented MATLAB feature:

a simple script can be called from another script. Consider a script called `main.m`. If we add the line `newScript.m` somewhere inside `main.m`, this script will be called and run, after which, control will be returned to the script `main.m`. This second script uses the `disp()` function to print "Hi!" to the screen, as illustrated in Example 1.1.

Example 1.1 Combine two scripts, `main.m` and `newScript.m`, such that `main.m` calls and runs `newScript.m`. The result is that running `main.m` displays Hi! in the Command Window.

```
%% main.m

newScript;

% end main.m
```

```
%% newScript.m

disp('Hi!');

% end newScript.m
```

```
>> main.m
Hi!
```

In fact, all built-in MATLAB functions are called and behave similarly to the script `newScript.m` that was called from inside `main.m`. This is our first introduction to the concept of calling other scripts, and is also used here for introducing the concept behind MATLAB functions. The process of calling other script files is formalized through the use of a MATLAB function, which adds much more capability to the language. A function is shown in Example 1.2.

Example 1.2 Create a function by right-clicking in the Current Folder and selecting New File → Function. The script that is created can be renamed as 'newFunction' and when opened in Editor looks like this:

```
function [ output_args ] = newFunction( input_args )
%NEWFUNCTION Summary of this function goes here
% Detailed explanation goes here

disp(Hi!);

end
```

```
>> newFunction()
Hi!
```

If we want to pass information to the function, we can use the `input_args` area; to pass information back to the calling program, we can

use the `output_args` area.[1–3] The code `disp('Hi!')` just before the end statement prints the word 'Hi!'. The function can be run from the Command Window by typing the function name and pressing the Return or Enter key.

These examples show how to call scripts and incorporate functions that you have defined into your program. Notice that there are parentheses at the end of the function name. This is not always required for a function, but, you will notice that the parentheses make user-written functions look much more like built-in functions. Recalling that MATLAB built-in functions are all written in lower case, it is convenient to capitalize some letters in user-written functions to distinguish them from the built-in functions.

There is another way to use functions that can make your code easier to read. Functions used in this way are referred to as *inline* functions and are very easy to implement. Also, they are self-contained and don't make use of variables in a script. As a very simple but illustrative example,

$$C = \text{inline ('A + B', 'A', 'B');}$$

is a summation of the values 'A' and 'B' and would be called as C(5,4), which would return the value of 9. The power of this notation becomes apparent when working in complicated mathematical problems and tracking parts of calculations becomes more challenging.

While simple to use, functions provide a very powerful tool that allows sophisticated programs to be developed. Such programs can be extremely complicated; however, with the use of functions, they are quite easy to read and manage during the development cycle.

1.8 Practice Problems

1. For A = [1 2 3] and B = [4 5 6], write a script to calculate (a) A + B; (b) A′ * B; (c) A. * B; and (d) A. / B.

Answers

(a) [5 7 9]
(b) [4 5 6; 8 10 12; 12 15 18]
(c) [4 10 18]
(d) [0.25 0.4 0.5]

References

1. D. Hanselman and B. R. Littlefield, *Mastering MATLAB® 7*, Prentice-Hall, Upper Saddle River, New Jersey (2011).

2. H. Moore, *MATLAB® for Engineers*, 4th Edition, Prentice-Hall, Upper Saddle River, New Jersey (2014).
3. R. Pratap, *Getting Started with MATLAB 7: A Quick Introduction for Scientists and Engineers*, Oxford University Press, Oxford (2006).
4. C. Rorres and H. Anton, *Applications of Linear Algebra*, John Wiley & Sons, Hoboken, New Jersey (1984).