SCHOOL OF PHYSICS, ASTRONOMY & MATHEMATICS

4PAM1008 MATLAB

2 – Basic MATLAB Operation

Dr Richard Greenaway

2 Basic MATLAB Operation

2.1 Overview

2.1.1 The Command Line

In this Workshop you will learn how to construct and execute numeric expressions, run MATLAB commands, control how answers are displayed and other aspects of interacting with MATLAB. Because MATLAB is, at its core, a programming language the main form of interaction is via a *command line*, that is, a window in which the user types *statements* which are executed when the user presses *enter*. For some tasks there are standard user interfaces available for such things as formatting plots but the power of MATLAB is only really understood when you learn to do things *programmatically*.

Consider again exercise 1.2. You entered a command which called a function, `peaks`, returning a matrix of data assigned to a variable `z`. We then used the result of that command, ie `z`, as the input to a new function, `surf`, which plotted the data.

```
>> z = peaks(30);
>> surf(z)
```

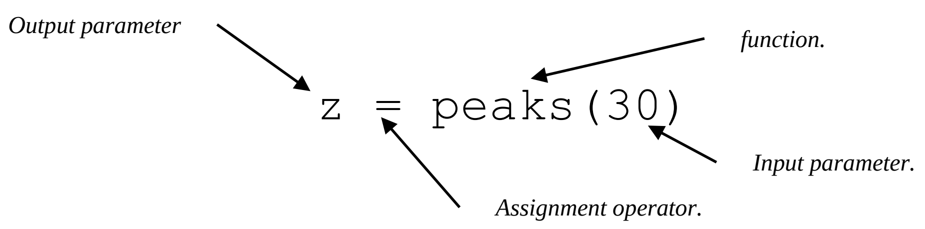
When you type as statement on the command line nothing happens until you press <enter>.

⚠ Once a command has executed you cannot *undo* it. If you make a mistake, simply re-type the command , correcting any errors, and run it again.

2.1.2 Operators, Variables, Functions and Commands

We construct an expression to execute from various components. A variable is simply 'named' data. We might store a number in a variable called `x`, and we can access that number by referring to its variable name. Operators exist to perform arithmetic and other operations. There are also *commands* which perform operations such as saving data to disk, clearing data from memory, displaying the contents of a directory etc. All of these various components can be used on the command line

All programming languages have libraries of functions. A function is a discrete set of instructions which perform some kind of operation, usually on data supplied by the user. The anatomy of a function call is shown below. A function takes zero or more inputs on which it performs some kind of operation. Once completed it returns zero or more outputs. The '=' is called the *assignment* operator; it assigns the output returned from the function to a variable called `z`.



```
z = peaks(30)
```

MATLAB contains thousands of functions for data processing, either built in or contained in *toolboxes*. A toolbox is an add-on for MATLAB which provides a library of functions for use in a specific application area such as image processing or statistics.

> Toolboxes will not be dealt with explicitly in this course with the exception of the **Symbolic Math** toolbox which we will cover in Workshop 5.

2.1.3 Repeating Commands

It would clearly be tedious to have to re-type an expression each time you wanted to use it. For example if you do a calculation which contains a typing mistake you don't want to re-type the whole thing again. You *never* have to do that. There are various ways to *re-play* your code.

1. **Use accelerator keys.**
   We can recall previous MATLAB commands by using the ↑ and ↓ cursor keys. Repeatedly pressing ↑ will review the previous commands the most recent first. To re-execute the command simply press the return key. To edit, use the cursor keys ← or →to move backwards and forwards through the line. Characters may be inserted by typing at the current cursor position or deleted using the **Del** key. This process is most commonly used when long command lines have been mistyped or when you want to re-execute a command that is very similar to one used previously. To recall the most recent command starting with **f**, say, type **f** at the prompt followed by ↑. Similarly, typing **fo** followed by ↑ will recall the most recent command starting with **fo**.

2. **Drag-and-drop commands from the Command History Window**
   Select the command in the history window and drag it to the command window, then edit it as desired and press **enter** to execute it.

3. **Write a MATLAB script or function**
   This is the most powerful means of carrying out tasks in MATLAB and we will introduce this topic at the end of the current Workshop.

2.1.4 Cleaning Up

During a session you may have create a lot of variables some or all of which you no longer require. Use the **clear** command to delete some or all of the data. Remember that the data is listed in the *workspace* window. You can delete *all* variables simply by typing clear

```
» clear
```

Alternatively, you delete selected variables by listing them after the command

```
>> clear a b x bigArray
```

Similarly, if you want to clear out all the clutter in the command window, simply type **clc** on the command line.

2.1.5 Error Messages

Typing mistakes will happen frequently, especially when constructing a complex expression.

What's wrong with this ?

```
>>sin(pi/5)*((3 + 15/(45^(1/3)) + tan(0.8*pi))
```

It's not immediately obvious but when you press **enter** …

```
>> sin(pi/5)*((3 + 15/(45^(1/3)) + tan(0.8*pi))
??? sin(pi/5)*((3 + 15/(45^(1/3)) + tan(0.8*pi))
                                                 |
Error: Expression or statement is incorrect--possibly unbalanced (, {, or [.
```

MATLAB prints an error message and if possible will show you on the line where the error is (look for the little vertical line), but not always. Above it has correctly determined an unbalanced bracket but it's not at the end of the line.

```
>> sin(pi/5)*((3 + 15/(45^(1/3))) + tan(0.8*pi))

ans =

   3.815091183583086
```

2.2 Variables

A MATLAB variable is essentially a tag that you assign to a value while that value remains in memory. The tag gives you a way to reference the value in memory so that your programs can read it, operate on it with other data, and save it back to memory. To define a variable simply assign a value to a variable name a follows. '=' is called the assignment operator. It does **not** mean 'equal to'.

```
>>  x = 10;
```

⚠ If you have experience of generic programming languages then you will be used to the idea of declaring a variable before using it. The declaration specifies the type, such as integer or double. In MATLAB this is not necessary, simply assigning a value to a name will define the variable.

MATLAB variable names must begin with a letter, which may be followed by any combination of letters, digits, and underscores. **MATLAB variables are case-sensitive**.

```
>> x =10
>> X = 20
```

are two different variables.

2.2.1.1 Special Values

There are some 'built in values', special functions which return pre-defined values such as **pi** which returns the value π.

```
>> pi

ans =

    3.1416
```

You notice in the above example that the value returned is called **ans** which is another special value. **ans** always contains the most recent result *if you haven't assigned it to an explicit output.*

The following table lists the special values defined in MATLAB.

Function	Output
ans	Most recent answer (variable). If you do not assign an output variable to an expression, MATLAB automatically stores the result in ans.
eps	Floating-point relative accuracy. This is the tolerance the MATLAB software uses in its calculations.
intmax	Largest 8-, 16-, 32-, or 64-bit integer your computer can represent.
intmin	Smallest 8-, 16-, 32-, or 64-bit integer your computer can represent.
realmax	Largest floating-point number your computer can represent.
realmin	Smallest positive floating-point number your computer can represent.
pi	π
i, j	imaginary unit
inf	Infinity. Calculations like n/0, where n is any nonzero real value, result in inf.
NaN	Not a Number, an invalid numeric value. Expressions like 0/0 and inf/inf result in a NaN, as do arithmetic operations involving a NaN. Also, if n is complex with a zero real part, then n/0 returns a value with a NaN real part.
computer	Computer type.
version	MATLAB version string.

Table 2-1

Exercise 2-1

Enter the following commands and look at the results.

```
>> 1/0
>> 0/0
>> i^2
>> 10 - NaN
```

Any computation involving an **NaN** always results in an **NaN**.

2.3 Operators

Arithmetic operators perform numeric computations, for example, adding two numbers or raising the elements of an array to a given power. The following table summarises the standard arithmetic operators.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
+	Unary plus
-	Unary minus
/	division
^	Power

Table 2-2

⚠ Operators will be dealt with in more detail in Workshop 3 when we look at matrices. For now we confine ourselves to simple variable types.

As well as standard arithmetic operators there are *relational* and *logical* operators, all of which will be covered in Workshop 3.

Calculating Roots

The function **sqrt** can be used to calculate the square root of a value but for any order root the general approach is to use the power operator, '^' since a fractional power gives you the root. **Be sure to use brackets to force the correct order of operation.**

To calculate $\sqrt[3]{2345}$ type:

```
2345^(1/3)
```

Exercise 2-2

Use MATLAB to calculate the following.

1. $3^4/18$ (*ans.* 4.5)

2. $\left(\frac{5}{2^3}\times 6\right)\times(4-3\times 8)^3$ (*ans.* -30000)

3. $(\sqrt{7225}-4)\big/_{3^2}$ (*ans.* 9)

4. Solve the following equations

a. $2x^2 + 22x + 36$ (*ans.* -2, -9)
b. $x^2 - 2x - 15$, (*ans.* 5, -3)

using the standard quadratic formula,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Define variables *a*, *b*, and *c* for equation **a** and then create the equation for each root. To repeat for equation **b** assign the new coefficient values and re-execute the expression by using one of the methods detailed earlier for re-executing commands. **Be sure to use brackets to enforce correct order of evaluation.**

2.4 Functions

MATLAB comes with numerous functions for solving maths problems. The core MATLAB program comes with a default set of basic math functions for trigonometry, exponentials/logs, complex numbers etc as well as functions for more complex maths – polynoimals, coordinate conversion, interpolation and so on. Then there are the toolboxes which extend the capabilities of MATLAB in areas such as statistics, signal processing, optimisation and many more.

Table 2-3 provides a very short list of some key maths functions, some of which you will need in the exercises.

Function	Description
`sin, cos, tan`	Standard trig functions. The input must be in **radians**.
`asin, acos, atan`	The inverse trig functions. The angle returned is in **radians**.
`sinh, cosh, tanh`	Hyperbolic functions.
`exp`	Exponential `exp(x)` = e^x
`log`	*natural* logarithm.
`log10`	Base 10 log
`nthroot`	Returns the real nth root of a specified number. `nthroot(-2, 3)` returns the cubed root of -2. This does not behave in exactly the same way as the '^' operator described earlier which will return a complex root.
`sqrt`	Square root
`abs`	The absolute value or, if the number is complex, the magnitude.

Table 2-3. Common functions

Here are a few examples.

```
>> y = 2*sin(pi/4)

y =

    1.4142

>> angle = acos(-1)

angle =

    3.1416

>> log(exp(1))

ans =

     1
```

2.4.1 Function Overloading

This is a somewhat strange idea if you are not used to object oriented programming but it is important to understand the concept because you will be meeting it.

Do the following exercise.

<u>Exercise 2-3</u>

1. Create an array of values as follows.

   ```
   >> array = 10*rand([1  10])
   ```

   This creates an array of ten random numbers between 0 and 10.

2. Now apply the `diff` function to that array.

   ```
   >> diff(array)
   ```

   The function `diff` calculates the differences between adjacent values in the array.

3. Now try the following.

   ```
   >> diff('x^2')
   ```

   What do you see? When an expression is entered as a string, diff treats it as a *symbolic* expression and determines the differential.

The function `diff` behaves differently depending on the type of input. Such behaviour is called *overloading*.

2.5 Controlling Output

2.5.1 Suppressing Output

We often do not want to see the result of intermediate calculations. To do this we terminate the assignment statement or expression with a semi-colon.

```
>> x=9; y=2*x/3; z=y^2+y*sqrt(x)

z =

    54
```

The values of **x** and **y** are hidden. We also note here that we can place several statements on one line, separated by commas or semi- colons.

It is a good idea to get used to using the semi-colon especially when using arrays. It is possible to have pages and pages of output. If you find a calculation going on and on you can stop it by pressing the keys **Ctrl** and **C** simultaneously.

2.5.2 Controlling output with the `format` command

The `format` command can be used to control the output format of numeric values displayed in the Command Window.

⚠ The `format` function affects only how numbers display, not how MATLAB computes or saves them.

To set the format, use the following syntax.

```
>> format type
```

Where *`type`* is options listed in the following table.

Type	Result
`short` (default)	Scaled fixed-point format, with 4 digits after the decimal point. For example, `3.1416`. If you are displaying a matrix with a wide range of values, consider using `short g`.
`long`	Scaled fixed-point format with 14 to 15 digits after the decimal point for double; and 7 digits after the decimal point for single. For example, `3.141592653589793`.
`short e`	Floating-point format, with 4 digits after the decimal point. For example, `3.1416e+000`.
`long e`	Floating-point format, with 14 to 15 digits after the decimal point for double; and 7 digits after the decimal point for single. For example, `3.141592653589793e+000`.
`short g`	Fixed- or floating-point, whichever is more readable, with 4 digits after the decimal point.

	For example, `3.1416`.
`long g`	Fixed- or floating-point, whichever is more readable, with 14 to 15 digits after the decimal point for double; and 7 digits after the decimal point for single. For example, `3.14159265358979`.
`short eng`	Engineering format that has 4 digits after the decimal point, and a power that is a multiple of three. For example, `3.1416e+000`.
`long eng`	Engineering format that has exactly 16 significant digits and a power that is a multiple of three. For example, `3.14159265358979e+000`.

Table 2-4 Format command options

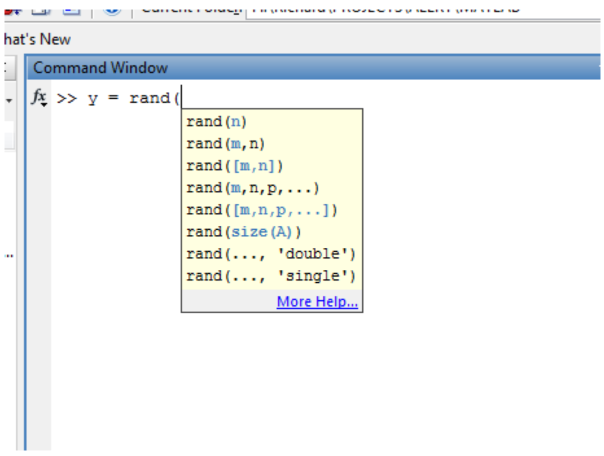
Example.

```
>> 10*pi

ans =

   31.4159

>> format long
>> 10*pi

ans =

  31.415926535897931
```

2.6 Getting Help

MATLAB provides an excellent help system available in the command window and as a separate help interface.

You may have already noticed the auto-prompting which appears when you type a function name as illustrated in fig. 1. These function hints show the parameter options available and a link to more information.



```
>> y = rand(
rand(n)
rand(m,n)
rand([m,n])
rand(m,n,p,...)
rand([m,n,p,...])
rand(size(A))
rand(..., 'double')
rand(..., 'single')
```

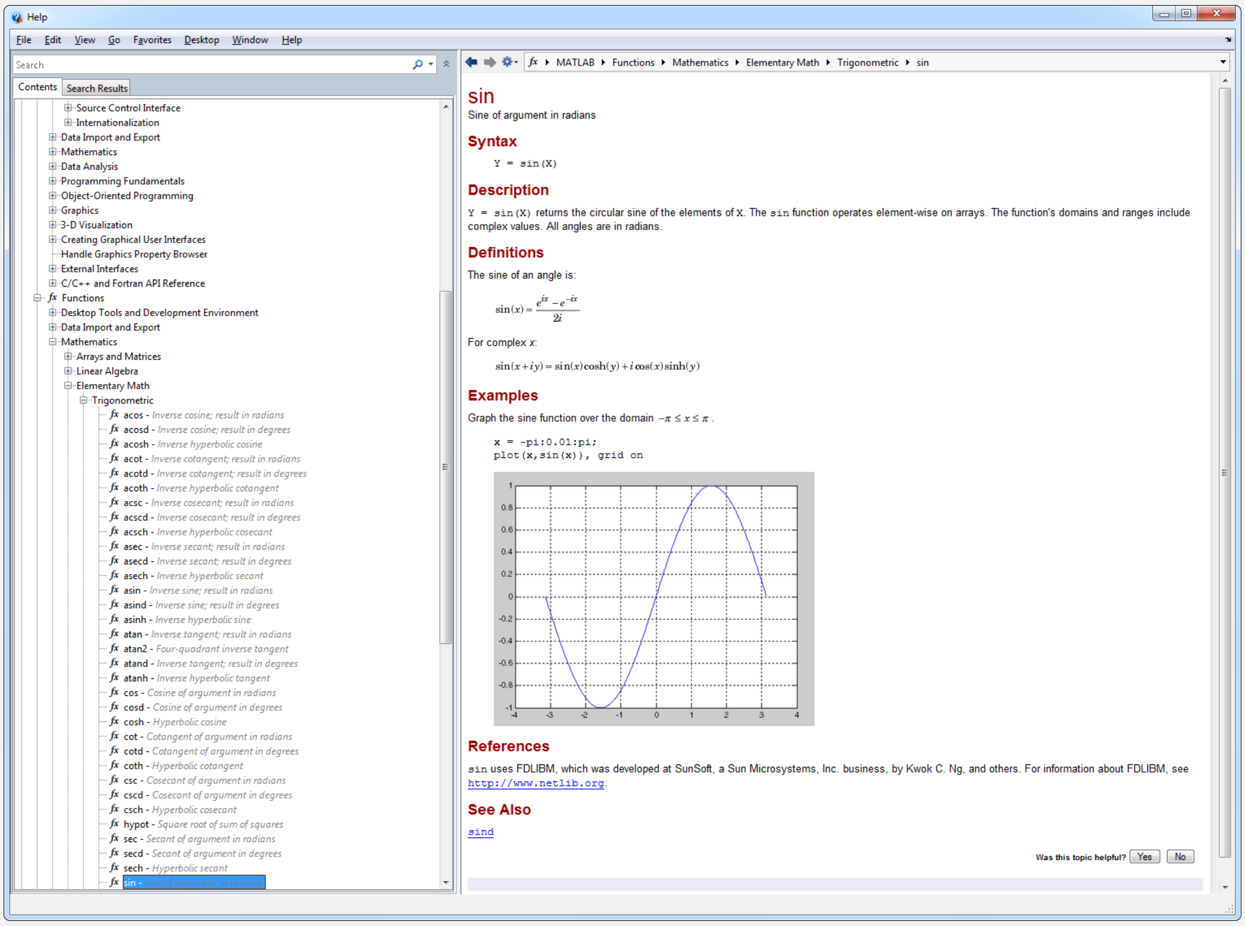
Figure 1. Function prompting.

For all functions, operators, keywords etc, you can display help in the command window simply by typing `help <item>`.

```
>> help sin
 SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded methods:
       codistributed/sin

    Reference page in Help browser
       doc sin
```

Alternatively if you type `doc <item>` the help browser window will open providing fully-formatted help.

```
>> doc sin
```


Help
File Edit View Go Favorites Desktop Window Help
Search
Contents Search Results
MATLAB ▸ Functions ▸ Mathematics ▸ Elementary Math ▸ Trigonometric ▸ sin

Source Control Interface
Internationalization
Data Import and Export
Mathematics
Data Analysis
Programming Fundamentals
Object-Oriented Programming
Graphics
3-D Visualization
Creating Graphical User Interfaces
Handle Graphics Property Browser
External Interfaces
C/C++ and Fortran API Reference
Functions
Desktop Tools and Development Environment
Data Import and Export
Mathematics
Arrays and Matrices
Linear Algebra
Elementary Math
Trigonometric
acos - *Inverse cosine; result in radians*
acosd - *Inverse cosine; result in degrees*
acosh - *Inverse hyperbolic cosine*
acot - *Inverse cotangent; result in radians*
acotd - *Inverse cotangent; result in degrees*
acoth - *Inverse hyperbolic cotangent*
acsc - *Inverse cosecant; result in radians*
acscd - *Inverse cosecant; result in degrees*
acsch - *Inverse hyperbolic cosecant*
asec - *Inverse secant; result in radians*
asecd - *Inverse secant; result in degrees*
asech - *Inverse hyperbolic secant*
asin - *Inverse sine; result in radians*
asind - *Inverse sine; result in degrees*
asinh - *Inverse hyperbolic sine*
atan - *Inverse tangent; result in radians*
atan2 - *Four-quadrant inverse tangent*
atand - *Inverse tangent; result in degrees*
atanh - *Inverse hyperbolic tangent*
cos - *Cosine of argument in radians*
cosd - *Cosine of argument in degrees*
cosh - *Hyperbolic cosine*
cot - *Cotangent of argument in radians*
cotd - *Cotangent of argument in degrees*
coth - *Hyperbolic cotangent*
csc - *Cosecant of argument in radians*
cscd - *Cosecant of argument in degrees*
csch - *Hyperbolic cosecant*
hypot - *Square root of sum of squares*
sec - *Secant of argument in radians*
secd - *Secant of argument in degrees*
sech - *Hyperbolic secant*
sin - *Sine of argument in radians*

sin
Sine of argument in radians

Syntax

```
Y = sin(X)
```

Description

`Y = sin(X)` returns the circular sine of the elements of `X`. The `sin` function operates element-wise on arrays. The function's domains and ranges include complex values. All angles are in radians.

Definitions

The sine of an angle is:

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

For complex *x*:

$$\sin(x+iy) = \sin(x)\cosh(y) + i\cos(x)\sinh(y)$$

Examples

Graph the sine function over the domain $-\pi \le x \le \pi$.

```
x = -pi:0.01:pi;
plot(x,sin(x)), grid on
```

References

`sin` uses FDLIBM, which was developed at SunSoft, a Sun Microsystems, Inc. business, by Kwok C. Ng, and others. For information about FDLIBM, see http://www.netlib.org.

See Also

sind

The help browser provides detailed help on all aspects of using MATLAB.

2.7 M – Files: Creating MATLAB Scripts and Functions

It would obviously be tedious and difficult to do complex tasks if we had to manually enter everything in the command window each time we wanted to carry out the task. However, we can automate an activity, that is we can write a program, by placing commands and expressions in a special file called an m-file which is simply a text file with the extension **.m**. There are two types of m-file. The first and simplest is a *script*, which simply lists the commands you wish to execute as if you had entered them in the command window. To run those commands, type the name of the script in the command window (without the extension).

Here is an example which calculates the solutions to Exercise 2.2, Q 4.

```
% ExampleScript.m
% Example M-file script
% Lines begining with '%' are comment lines which are ignored


% This script calculates the answers to Exercise 2.2, Q4

disp('Solution to equation (a)');
a = 2; b = 22; c = 36;
x = (-b + (b^2-(4*a*c))^(1/2))/(2*a)
x = (-b - (b^2-(4*a*c))^(1/2))/(2*a)

disp('Solution to equation (b)');
a = 1; b = -2; c = -15;
x = (-b + (b^2-(4*a*c))^(1/2))/(2*a)
x = (-b - (b^2-(4*a*c))^(1/2))/(2*a)
```

Note the lines beginning with the `%` character. Anything following is ignored by MATLAB, you can write comments to notate the script. The `disp()` function is simply a way to put some output in the command window.

Download the script and try running it.

```
>> ExampleScript
Solution to equation (a)

x =

    -2


x =

    -9

Solution to equation (b)

x =

     5


x =

    -3
```

To create or edit an existing script use the `edit` command in the command window

```
>> edit ExampleScript
```

If the file exists, it will be loaded into a special editor. If not, the editor will run with a blank page. The editor is simply a specialised text editor, you can create an m-file using notepad if you prefer. However, it has powerful features to enable you to debug code and so it is recommended that you use it.

⚠ The editor will not be covered in any more detail here, you can always consult the MATLAB help .

⚠ It is recommended that you use scripts to save and document your work. It's easy and powerful.

2.7.1 Functions

You can create your own functions to add to those supplied with MATLAB. Functions are m-files with a special header line which defines them as a function. The main difference between a script and a function is that a function accepts input from and returns output to its caller, whereas scripts do not. You define MATLAB functions in a file that begins with a line containing the **function** key word. You cannot define a function within a script file or at the MATLAB command line.

Here is an example which re-writes the solution to Exercise 2.2, Q4 as function. Instead of defining the quadratic coefficients as variables you pass the values as parameters to the function. It then returns the roots as output parameters.

```
function [x1 x2] = quadratic(a,b,c)
% QUADRATIC
%
% [x1 x2] = quadratic(a,b,c)
%
% Return the roots of the quatratic equation
%
%   ax^2 + bx + c
%
%

x1 = (-b + (b^2-(4*a*c))^(1/2))/(2*a);
x2 = (-b - (b^2-(4*a*c))^(1/2))/(2*a);
```

The m-file should have the same name as the function (ie **quadratic.m**). You then run it as you would any standard MATLAB function.

The first line in the file must have the format shown in the example

```
    function [y1 y2 ..] = functionname(x1, x2, …)
```

Any comments placed directly after the first line will be displayed in the MATLAB help system in exactly the same way as any of the built-in functions.

```
>> help quadratic
  QUADRATIC

  [x1 x2] = quadratic(a,b,c)

  Return the roots of the quatratic equation

    ax^2 + bx + c
```

Execute it as you would any function …

```
>> [x1 x2] = quadratic(2,22,36)

x1 =

    -2


x2 =

    -9

>>
```

2.8 Further Exercises

Exercise 2-4

1. Evaluate $(a + b)^c$
   For a =5, b = -2, c = 3

2. Given that u = 2, v = $3u^2$, $w = \sqrt{2 + \frac{v}{u}}$, evaluate w

3. Evaluate the following.
   (i) $\sin\frac{\pi}{3}$

   (ii) $\cos\frac{-5\pi}{6}$

   (iii) $\tan^{-1}(-1)$

4. Determine $(\pi^4 + \pi^5)^{1/6} - e$

5. Determine $\sqrt[3]{e^{\ln(27)}}$