Chapter 4

Introduction to programming in MATLAB

4.1 Introduction

So far in these lab sessions, all the commands were executed in the Command Window. The problem is that the commands entered in the Command Window cannot be saved and executed again for several times. Therefore, a different way of executing repeatedly commands with MATLAB is:

1. to *create* a file with a list of commands,
2. *save* the file, and
3. *run* the file.

If needed, corrections or changes can be made to the commands in the file. The files that are used for this purpose are called script files or *scripts* for short.

This section covers the following topics:

- M-File Scripts
- M-File Functions

4.2 M-File Scripts

A *script file* is an external file that contains a sequence of MATLAB statements. Script files have a filename extension `.m` and are often called M-files. M-files can be *scripts* that simply execute a series of MATLAB statements, or they can be *functions* that can accept arguments and can produce one or more outputs.

4.2.1 Examples

Here are two simple scripts.

Example 1

Consider the system of equations:

$$\begin{cases} x + 2y + 3z &= 1 \\ 3x + 3y + 4z &= 1 \\ 2x + 3y + 3z &= 2 \end{cases}$$

Find the solution x to the system of equations.

SOLUTION:

- Use the MATLAB *editor* to create a file: **File** → **New** → **M-file**.
- Enter the following statements in the file:

```
A = [1 2 3; 3 3 4; 2 3 3];
b = [1; 1; 2];
x = A\b
```

- Save the file, for example, `example1.m`.
- Run the file, in the command line, by typing:

```
>> example1
x   =
    -0.5000
     1.5000
    -0.5000
```

When execution completes, the variables (`A`, `b`, and `x`) remain in the workspace. To see a listing of them, enter `whos` at the command prompt.

NOTE: The MATLAB editor is both a text editor specialized for creating M-files and a graphical MATLAB debugger. The MATLAB editor has numerous menus for tasks such as *saving*, *viewing*, and *debugging*. Because it performs some simple checks and also uses color to differentiate between various elements of codes, this text editor is recommended as the tool of choice for writing and editing M-files.

There is another way to open the editor:

```
>> edit
```

or

```
>> edit filename.m
```

to open `filename.m`.

Example 2

Plot the following cosine functions, $y_1 = 2\cos(x)$, $y_2 = \cos(x)$, and $y_3 = 0.5 * \cos(x)$, in the interval $0 \leq x \leq 2\pi$. This example has been presented in previous Chapter. Here we put the commands in a file.

- Create a file, say `example2.m`, which contains the following commands:

```
x = 0:pi/100:2*pi;
y1 = 2*cos(x);
y2 = cos(x);
y3 = 0.5*cos(x);
plot(x,y1,'--',x,y2,'-',x,y3,':')
xlabel('0 \leq x \leq 2\pi')
ylabel('Cosine functions')
legend('2*cos(x)','cos(x)','0.5*cos(x)')
title('Typical example of multiple plots')
axis([0 2*pi -3 3])
```

- Run the file by typing `example2` in the Command Window.

4.2.2 Script side-effects

All variables created in a script file are added to the workspace. This may have undesirable effects, because:

- Variables already existing in the workspace may be overwritten.
- The execution of the script can be affected by the state variables in the workspace.

As a result, because scripts have some undesirable side-effects, it is better to code any complicated applications using rather function M-file.

4.3 M-File functions

As mentioned earlier, functions are programs (or *routines*) that accept *input* arguments and return *output* arguments. Each M-file function (or *function* or *M-file* for short) has its *own* area of workspace, separated from the MATLAB base workspace.

4.3.1 Anatomy of a M-File function

This simple function shows the basic parts of an M-file.

```
function f = factorial(n)                    (1)
% FACTORIAL(N) returns the factorial of N.   (2)
% Compute a factorial value.                 (3)

f = prod(1:n);                               (4)
```

The first line of a function M-file starts with the keyword `function`. It gives the function *name* and order of *arguments*. In the case of function `factorial`, there are up to one output argument and one input argument. Table 4.1 summarizes the M-file function.

As an example, for $n = 5$, the result is,

```
>> f = factorial(5)
f =
  120
```

Table 4.1: Anatomy of a M-File function

Part no.	M-file element	Description
(1)	Function definition line	Define the function name, and the number and order of input and output arguments
(2)	H1 line	A one line summary description of the program, displayed when you request Help
(3)	Help text	A more detailed description of the program
(4)	Function body	Program code that performs the actual computations

Both *functions* and *scripts* can have all of these parts, except for the *function definition line* which applies to *function* only.

In addition, it is important to note that *function name* must begin with a letter, and must be no longer than than the maximum of 63 characters. Furthermore, the name of the text file that you save will consist of the function name with the extension `.m`. Thus, the above example file would be `factorial.m`.

Table 4.2 summarizes the differences between *scripts* and *functions*.

Table 4.2: Difference between scripts and functions

Scripts	Functions
- Do not accept input arguments or return output arguments. - Store variables in a workspace that is shared with other scripts - Are useful for automating a series of commands	- Can accept input arguments and return output arguments. - Store variables in a workspace internal to the function. - Are useful for extending the MATLAB language for your application

4.3.2 Input and output arguments

As mentioned above, the input arguments are listed inside parentheses following the function name. The output arguments are listed inside the brackets on the left side. They are used to transfer the output from the function file. The general form looks like this

```
function [outputs] = function_name(inputs)
```

Function file can have none, one, or several output arguments. Table 4.3 illustrates some possible combinations of input and output arguments.

Table 4.3: Example of input and output arguments

`function C=FtoC(F)`	One input argument and one output argument
`function area=TrapArea(a,b,h)`	Three inputs and one output
`function [h,d]=motion(v,angle)`	Two inputs and two outputs

4.4 Input to a script file

When a script file is executed, the variables that are used in the calculations within the file must have assigned values. The assignment of a value to a variable can be done in three ways.

1. The variable is defined in the script file.
2. The variable is defined in the command prompt.
3. The variable is entered when the script is executed.

We have already seen the two first cases. Here, we will focus our attention on the third one. In this case, the variable is defined in the script file. When the file is executed, the user is *prompted* to assign a value to the variable in the command prompt. This is done by using the `input` command. Here is an example.

```
% This script file calculates the average of points
% scored in three games.
% The point from each game are assigned to a variable
% by using the 'input' command.

game1 = input('Enter the points scored in the first game ');
```

```
game2 = input('Enter the points scored in the second game ');
game3 = input('Enter the points scored in the third game ');
average = (game1+game2+game3)/3
```

The following shows the command prompt when this script file (saved as `example3`) is executed.

```
>> example3
>> Enter the points scored in the first game   15
>> Enter the points scored in the second game  23
>> Enter the points scored in the third game   10

average =
        16
```

The `input` command can also be used to assign *string* to a variable. For more information, see MATLAB documentation.

A typical example of M-file function programming can be found in a recent paper which related to the solution of the ordinary differential equation (ODE) [12].

4.5 Output commands

As discussed before, MATLAB automatically generates a *display* when commands are executed. In addition to this automatic display, MATLAB has several commands that can be used to generate displays or outputs.

Two commands that are frequently used to generate output are: `disp` and `fprintf`. The main differences between these two commands can be summarized as follows (Table 4.4).

Table 4.4: `disp` and `fprintf` commands

`disp`	. Simple to use. . Provide limited control over the appearance of output
`fprintf`	. Slightly more complicated than `disp`. . Provide total control over the appearance of output

4.6 Exercises

1. Liz buys three apples, a dozen bananas, and one cantaloupe for $2.36. Bob buys a dozen apples and two cantaloupe for $5.26. Carol buys two bananas and three cantaloupe for $2.77. How much do single pieces of each fruit cost?

2. Write a function file that converts temperature in degrees Fahrenheit (°F) to degrees Centigrade (°C). Use `input` and `fprintf` commands to display a mix of text and numbers. Recall the conversion formulation, $C = 5/9 * (F - 32)$.

3. Write a user-defined MATLAB function, with two input and two output arguments that determines the height in centimeters (`cm`) and mass in kilograms (`kg`)of a person from his height in inches (`in.`) and weight in pounds (`lb`).

   (a) Determine in SI units the height and mass of a 5 ft.15 in. person who weight 180 lb.

   (b) Determine your own height and weight in SI units.

Chapter 5

Control flow and operators

5.1 Introduction

MATLAB is also a *programming language*. Like other computer programming languages, MATLAB has some decision making structures for control of command execution. These decision making or *control flow* structures include `for` loops, `while` loops, and `if-else-end` constructions. Control flow structures are often used in script M-files and function M-files.

By creating a file with the extension `.m`, we can easily write and run programs. We do not need to *compile* the program since MATLAB is an interpretative (not compiled) language. MATLAB has thousand of *functions*, and you can add your own using m-files.

MATLAB provides several tools that can be used to control the *flow* of a program (*script* or *function*). In a simple program as shown in the previous Chapter, the commands are executed one after the other. Here we introduce the flow control structure that make possible to skip commands or to execute specific group of commands.

5.2 Control flow

MATLAB has four control flow structures: the `if` statement, the `for` loop, the `while` loop, and the `switch` statement.

5.2.1 The ‘‘`if...end`’’ structure

MATLAB supports the variants of “`if`” construct.

- `if ... end`
- `if ... else ... end`

- `if ... elseif ... else ... end`

The simplest form of the `if` statement is

```
if expression
   statements
end
```

Here are some examples based on the familiar quadratic formula.

1.
```
discr = b*b - 4*a*c;
if discr < 0
   disp('Warning: discriminant is negative, roots are
   imaginary');
end
```

2.
```
discr = b*b - 4*a*c;
if discr < 0
   disp('Warning: discriminant is negative, roots are
   imaginary');
else
   disp('Roots are real, but may be repeated')
end
```

3.
```
discr = b*b - 4*a*c;
if discr < 0
   disp('Warning: discriminant is negative, roots are
   imaginary');
elseif discr == 0
   disp('Discriminant is zero, roots are repeated')
else
   disp('Roots are real')
end
```

It should be noted that:

- `elseif` has no space between `else` and `if` (one word)
- no semicolon (`;`) is needed at the end of lines containing `if`, `else`, `end`
- indentation of `if` block is not required, but facilitate the reading.
- the `end` statement is required

5.2.2 Relational and logical operators

A relational operator compares two numbers by determining whether a comparison is *true* or *false*. Relational operators are shown in Table 5.1.

Table 5.1: Relational and logical operators

Operator	Description
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
~=	Not equal to
&	AND operator
|	OR operator
~	NOT operator

Note that the "equal to" relational operator consists of two equal signs (==) (with no space between them), since = is reserved for the *assignment* operator.

5.2.3 The ''for...end'' loop

In the `for ... end` loop, the execution of a command is repeated at a fixed and predetermined number of times. The syntax is

```
for variable = expression
    statements
end
```

Usually, `expression` is a vector of the form `i:s:j`. A simple example of `for` loop is

```
for ii=1:5
    x=ii*ii
end
```

It is a good idea to indent the loops for readability, especially when they are nested. Note that MATLAB editor does it automatically.

Multiple `for` loops can be nested, in which case *indentation* helps to improve the readability. The following statements form the 5-by-5 symmetric matrix `A` with (i, j) element i/j for $j \geq i$:

```
n = 5; A = eye(n);
for j=2:n
    for i=1:j-1
        A(i,j)=i/j;
        A(j,i)=i/j;
    end
end
```

5.2.4 The ‘‘`while...end`’’ loop

This loop is used when the number of *passes* is not specified. The looping continues until a stated condition is satisfied. The `while` loop has the form:

```
while expression
      statements
end
```

The `statements` are executed as long as `expression` is true.

```
x = 1
while x <= 10
      x = 3*x
end
```

It is important to note that if the condition inside the looping is not well defined, the looping will continue *indefinitely*. If this happens, we can stop the execution by pressing **Ctrl-C**.

5.2.5 Other flow structures

- The `break` statement. A `while` loop can be terminated with the `break` statement, which passes control to the first statement after the corresponding `end`. The `break` statement can also be used to exit a `for` loop.
- The `continue` statement can also be used to exit a `for` loop to pass immediately to the next iteration of the loop, skipping the remaining statements in the loop.
- Other control statements include `return`, `continue`, `switch`, etc. For more detail about these commands, consul MATLAB documentation.

5.2.6 Operator precedence

We can build expressions that use any combination of *arithmetic*, *relational*, and *logical operators*. Precedence rules determine the order in which MATLAB evaluates an expression. We have already seen this in the "Tutorial Lessons".

Here we add other operators in the list. The precedence rules for MATLAB are shown in this list (Table 5.2), ordered from *highest* (1) to *lowest* (9) precedence level. Operators are evaluated from left to right.

Table 5.2: Operator precedence

Precedence	Operator
1	Parentheses ()
2	Transpose (.′), power (.^), matrix power (^)
3	Unary plus (+), unary minus (−), logical negation (∼)
4	Multiplication (.∗), right division (./), left division (.\), matrix multiplication (∗), matrix right division (/), matrix left division (\)
5	Addition (+), subtraction (−)
6	Colon operator (:)
7	Less than (<), less than or equal to (≤), greater (>), greater than or equal to (≥), equal to (==), not equal to (∼=)
8	Element-wise AND, (&)
9	Element-wise OR, (|)

5.3 Saving output to a file

In addition to displaying output on the screen, the command `fprintf` can be used for writing the output to a *file*. The saved data can subsequently be used by MATLAB or other softwares.

To save the results of some computation to a file in a text format requires the following steps:

1. Open a file using `fopen`
2. Write the output using `fprintf`
3. Close the file using `fclose`

Here is an example (script) of its use.

```
% write some variable length strings to a file
op = fopen('weekdays.txt','wt');
fprintf(op,'Sunday\nMonday\nTuesday\nWednesday\n');
fprintf(op,'Thursday\nFriday\nSaturday\n');
fclose(op);
```

This file (`weekdays.txt`) can be opened with any program that can read `.txt` file.

5.4 Exercises

NOTE: Due to the teaching class during this Fall Quarter 2005, the *problems* are *temporarily* removed from this section.