

Chapter Five

Programming in MATLAB (M-File)

1. Introduction

So far 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:

- To create a file with a list of commands.
- Save the file.
- 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.

2. M-File Scripts

Examples: Here are two simple scripts.

Ex.1

Consider the system of equations:

$$X + 2Y + 3Z = 1$$

$$3X + 3Y + 4Z = 1$$

$$X + 3Y + 3Z = 2$$

Find the solution x to the system of equations.

Solution:

Use the MATLAB editor to create a file: **File** → **New** → **Script**.

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.

There is another way to open the editor:

```
>> edit
```

or

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

To open filename.m.

Ex.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 $0 \leq x \leq 2\pi$. 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.

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.

3. M-File functions

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)
```

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

It is important to note that function name must begin with a letter, and must be no longer 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**

Table 10 : Difference between scripts and functions

SCRIPTS	FUNCTIONS
<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- 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

2- Input and output arguments

function [outputs] = function_name (inputs)

Table 11 : 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. Input to a script file

- The variable is defined in the script file.
- The variable is defined in the command prompt.
- The variable is entered when the script is executed.

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
```

>> 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

5. Output commands

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 12).

Table 12 : `disp` and `fprintf` commands

<code>disp</code>	<ul style="list-style-type: none">. Simple to use.. Provide limited control over the appearance of output
<code>fprintf</code>	<ul style="list-style-type: none">. Slightly more complicated than <code>disp</code>.. Provide total control over the appearance of output

6. Debugging M-files

This section introduces general techniques for finding errors in M-files. Debugging is the process by which you isolate and fix errors in your program or code. Debugging helps to correct two kinds of error:

- ❑ **Syntax errors:** For example omitting a parenthesis or misspelling a function name.
- ❑ **Run-time errors:** Run-time errors are usually apparent and difficult to track down. They produce unexpected results.

7. Debugging process

1. Preparing for debugging
2. Setting breakpoints
3. Examining values
4. Correcting problems
5. Ending debugging

1- Preparing for debugging

Do the following to prepare for debugging:

- Open the file
- Save changes
- Be sure the file you run and any files it calls are in the directories that are on the search path.

2- Setting breakpoints

Set breakpoints to pause execution of the function, so we can examine where the problem might be. There are **three basic types** of breakpoints:

- ❑ A standard breakpoint, which stops at a specified line.
- ❑ A conditional breakpoint, which stops at a specified line and under specified conditions.
- ❑ An error breakpoint that stops when it produces the specified type of warning, error, NaN, or infinite value.

✓ Open Files when Debugging

1.0 + ÷ 1.1

```
1 - A = [1 2 3 4; 2 4 6  
2 - B = eye(3);  
3 - C = zeros(3,1);  
4 - D = ones(2,8);  
5 - K=[A B C; D];  
6 - K(4,6)=8  
7 - K'  
8 - K([3 1], :) = K([1 3]  
9 - K*K'  
10  
11  
12  
13  
14
```


- ✓ Open Files when Debugging
- Step F10
- Step In F11
- Step Out Shift+F11
- Run m1q4 F5
- Run Configuration for m1q4.m ▶
- Go Until Cursor
- Set/Clear Breakpoint F12**
- Set/Modify Conditional Breakpoint...
- Enable/Disable Breakpoint
- Clear Breakpoints in All Files
- Stop if Errors/Warnings...
- Exit Debug Mode Shift+F5

Stack: Base fx

Pause a Running File

To pause the execution of a program while it is running, go to the **Editor** tab and click the **Pause**  button. MATLAB pauses execution at the next executable line, and the **Pause**  button changes to a **Continue**  button. To continue execution, press the **Continue**  button.

Pausing is useful if you want to check on the progress of a long running program to ensure that it is running as expected.

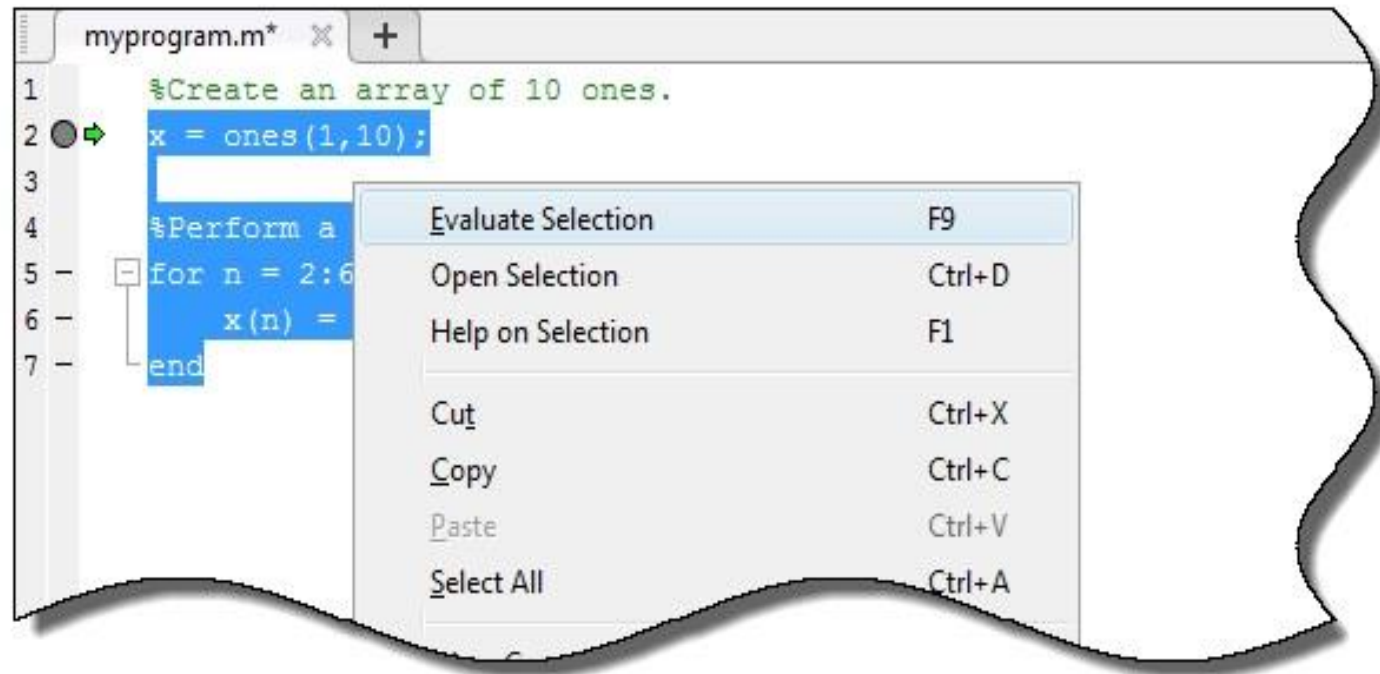
Note: Clicking the pause button can cause MATLAB to pause in a file outside your own program file. Pressing the **Continue**  button resumes normal execution without changing the results of the file.

3- Examining values

While the program is stopped, we can view the value of any variable currently in the workspace. Examine values when we want to see whether a line of code has produced the expected result or not. If the result is as expected, step to the next line, and continue running. If the result is not as expected, then that line, or the previous line, contains an error. Use **whos** to list the variables in the current workspace.

To modify a program while debugging:

1. While your code is paused, modify a part of the file that has not yet run.
Breakpoints turn gray, indicating they are invalid.
2. Select all the code after the line at which MATLAB is paused, right-click, and then select **Evaluate Selection** from the context menu.



After the code evaluation is complete, stop debugging and save or undo any changes made before continuing the debugging process.

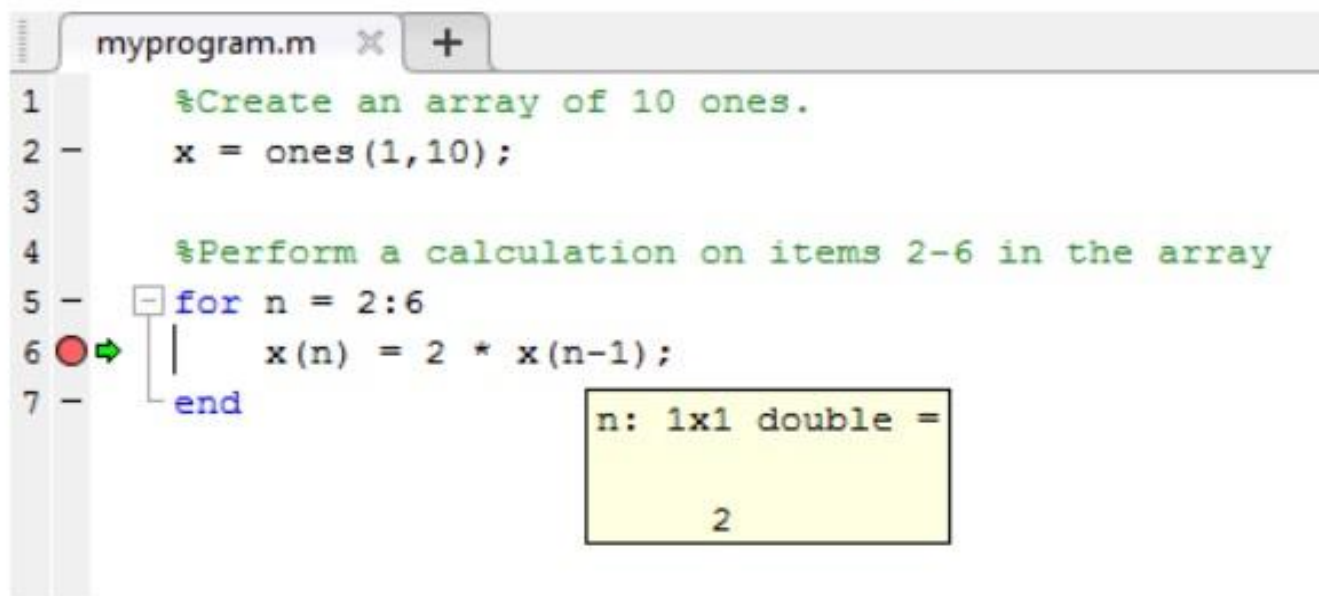
Find and Fix a Problem

While your code is paused, you can view or change the values of variables, or you can modify the code.

View or Change Variable While Debugging


```
for n = 2:6
    n: 1x1 double =
        6
end
```

For example, here MATLAB is paused inside a for loop where $n = 2$:



```
myprogram.m x +
1 %Create an array of 10 ones.
2 x = ones(1,10);
3
4 %Perform a calculation on items 2-6 in the array
5 for n = 2:6
6 x(n) = 2 * x(n-1);
7 end
```

n: 1x1 double =
2

- Type $n = 7$; in the command line to change the current value of n from 2 to 7.
- Press **Continue**  to run the next line of code.

MATLAB runs the code line $x(n) = 2 * x(n-1)$; with $n = 7$.

4- Correcting

While debugging, we can change the value of a variable to see if the new value produces expected results. While the program is stopped, assign a new value to the variable in the Command Window, Workspace browser, or Array Editor. Then continue running and stepping through the program.








5- Ending debugging

After identifying a problem, end the debugging session. It is best to quit debug mode before editing an M-file. Otherwise, you can get unexpected results when you run the file. To end debugging, select Exit Debug Mode from the Debug menu.

Step Through File

While debugging, you can step through a MATLAB file, pausing at points where you want to examine values.

This table describes available debugging actions and the different methods you can use to execute them.

Description	Toolbar Button	Function Alternative
Continue execution of file until the line where the cursor is positioned. Also available on the context menu.	 Run to Cursor	None
Execute the current line of the file.	 Step	<code>dbstep</code>
Execute the current line of the file and, if the line is a call to another function, step into that function.	 Step In	<code>dbstep in</code>
Resume execution of file until completion or until another breakpoint is encountered.	 Continue	<code>dbcont</code>
After stepping in, run the rest of the called function or local function, leave the called function, and pause.	 Step Out	<code>dbstep out</code>
Pause debug mode.	 Pause	None
Exit debug mode.	 Quit Debugging	<code>dbquit</code>

