



Part 1

Chapter 3

Programming with MATLAB

PowerPoints organized by Dr. Michael R. Gustafson II, Duke University

Chapter Objectives

- ▶ Learning how to create well-documented M-files in the edit window and invoke them from the command window.
- ▶ Understanding how script and function files differ.
- ▶ Understanding how to incorporate help comments in functions.
- ▶ Knowing how to set up M-files so that they interactively prompt users for information and display results in the command window.
- ▶ Understanding the role of subfunctions and how they are accessed.
- ▶ Knowing how to create and retrieve data files.

Objectives (cont)

- ▶ Learning how to write clear and well-documented *M*-files by employing structured programming constructs to implement logic and repetition.
- ▶ Recognizing the difference between `if...elseif` and `switch` constructs.
- ▶ Recognizing the difference between `for...end` and `while` structures.
- ▶ Understanding what is meant by vectorization and why it is beneficial.
- ▶ Understanding how anonymous functions can be employed to pass function functions to function *M*-files.

M-files

- ▶ While commands can be entered directly to the command window, MATLAB also allows you to put commands in text files called *M-files*. *M-files* are so named because the files are stored with a `.m` extension.
- ▶ There are two main kinds of M-file
 - ▶ Script files
 - ▶ Function files

Script Files

- ▶ A *script file* is merely a set of MATLAB commands that are saved on a file - when MATLAB runs a script file, it is as if you typed the characters stored in the file on the command window.
- ▶ Scripts can be executed either by typing their name (without the .m) in the command window, by selecting the Debug, Run (or Save and Run) command in the editing window, or by hitting the F5 key while in the editing window. Note that the latter two options will save any edits you have made, while the former will run the file as it exists on the drive.

Function Files

- ▶ *Function files* serve an entirely different purpose from script files. Function files can accept input arguments from and return outputs to the command window, but variables created and manipulated `within` the function do not impact the command window.

Function File Syntax

- ▶ The general syntax for a function is:

```
function outvar = funcname(arglist)  
% helpcomments  
statements  
outvar = value;
```

where

- ▶ *outvar*: output variable name
- ▶ *funcname*: function's name
- ▶ *arglist*: input argument list - comma-delimited list of what the function calls values passed to it
- ▶ *helpcomments*: text to show with help *funcname*
- ▶ *statements*: MATLAB commands for the function

Subfunctions

- ▶ A function file can contain a single function, but it can also contain a *primary function* and one or more *subfunctions*
- ▶ The primary function is whatever function is listed first in the M-file - its function name should be the same as the file name.
- ▶ Subfunctions are listed below the primary function. Note that they are *only* accessible by the main function and subfunctions within the same M-file and *not* by the command window or any other functions or scripts.

Input

- ▶ The easiest way to get a value from the user is the input command:
 - ▶ `n = input('promptstring')`
MATLAB will display the characters in *promptstring*, and whatever value is typed is stored in *n*. For example, if you type *pi*, *n* will store 3.1416...
 - ▶ `n = input('promptstring', 's')`
MATLAB will display the characters in *promptstring*, and whatever characters are typed will be stored as a string in *n*. For example, if you type *pi*, *n* will store the letters *p* and *i* in a 2x1 char array.

Output

- ▶ The easiest way to display the value of a matrix is to type its name, but that will not work in function or script files. Instead, use the `disp` command

```
disp(value)
```

will show the *value* on the screen, and if it is a string, will enclose it in single quotes.

Formatted Output

- ▶ For formatted output, or for output generated by combining variable values with literal text, use the `fprintf` command:

```
fprintf('format', x, y, ...)
```

where *format* is a string specifying how you want the value of the variables *x*, *y*, and more to be displayed - including literal text to be printed along with the values.
- ▶ The values in the variables are formatted based on format codes.

Format and Control Codes

- ▶ Within the *format* string, the following format codes define how a numerical value is displayed:
 - %d - integer format
 - %e - scientific format with lowercase e
 - %E - scientific format with uppercase E
 - %f - decimal format
 - %g - the more compact of %e or %f
- ▶ The following control codes produce special results within the *format* string:
 - \n - start a new line
 - \t - tab
 - \\ - print the \ character
- ▶ To print a ' put two a pair of ' in the *format* string

Creating and Accessing Files

- ▶ MATLAB has a built-in file format that may be used to save and load the values in variables.
- ▶ `save filename var1 var2 ... varn` saves the listed variables into a file named `filename.mat`. If no variable is listed, all variables are saved.
- ▶ `load filename var1 var2 ... varn` loads the listed variables from a file named `filename.mat`. If no variable is listed, all variables in the file are loaded.
- ▶ Note - these are not text files!

ASCII Files

- ▶ To create user-readable files, append the flag `-ascii` to the end of a `save` command. This will save the data to a text file in the same way that `disp` sends the data to a screen.
- ▶ Note that in this case, MATLAB does *not* append anything to the file name so you may want to add an extension such as `.txt` or `.dat`.
- ▶ To load a rectangular array from a text file, simply use the `load` command and the file name. The data will be stored in a matrix with the same name as the file (but without any extension).

Structured Programming

- ▶ Structured programming allows MATLAB to make decisions or selections based on conditions of the program.
- ▶ Decisions in MATLAB are based on the result of logical and relational operations and are implemented with `if`, `if...else`, and `if...elseif` structures.
- ▶ Selections in MATLAB are based on comparisons with a test expression and are implemented with `switch` structures.

Relational Operators

- ▶ From Table 3.2: Summary of relational operators in MATLAB:

Example	Operator	Relationship
<code>x == 0</code>	<code>==</code>	Equal
<code>unit ~= 'm'</code>	<code>~=</code>	Not equal
<code>a < 0</code>	<code><</code>	Less than
<code>s > t</code>	<code>></code>	Greater than
<code>3.9 <= a/3</code>	<code><=</code>	Less than or equal to
<code>r >= 0</code>	<code>>=</code>	Greater than or equal to

Logical Operators

- ▶ $\sim x$ (Not): true if x is false (or zero); false otherwise
- ▶ $x \ \& \ y$ (And): true if both x and y are true (or non-zero)
- ▶ $x \ | \ y$ (Or): true if either x or y are true (or non-zero)

Order of Operations

- ▶ Priority can be set using parentheses. After that, Mathematical expressions are highest priority, followed by relational operators, followed by logical operators. All things being equal, expressions are performed from left to right.
- ▶ *Not* is the highest priority logical operator, followed by *And* and finally *Or*
- ▶ Generally, do not combine two relational operators!
If $x=5$, $3 < x < 4$ should be false (mathematically), but it is calculated as an expression in MATLAB as:
 $3 < 5 < 4$, which leads to `true < 4` at which point `true` is converted to `1`, and $1 < 4$ is true!
- ▶ Use $(3 < x) \& (x < 4)$ to properly evaluate.

Decisions

- ▶ Decisions are made in MATLAB using `if` structures, which may also include several `elseif` branches and possibly a catch-all `else` branch.
- ▶ Deciding which branch runs is based on the result of *conditions* which are either true or false.
 - ▶ If an `if` tree hits a *true* condition, that branch (and that branch only) runs, then the tree terminates.
 - ▶ If an `if` tree gets to an `else` statement without running any prior branch, that branch will run.
- ▶ *Note* - if the *condition* is a matrix, it is considered true if and only if all entries are true (or non-zero).

Selections

- ▶ Selections are made in MATLAB using switch structures, which may also include a catch-all otherwise choice.
- ▶ Deciding which branch runs is based on comparing the value in some test expression with values attached to different cases.
 - ▶ If the test expression matches the value attached to a case, that case's branch will run.
 - ▶ If no cases match and there is an otherwise statement, that branch will run.

Loops

- ▶ Another programming structure involves loops, where the same lines of code are run several times. There are two types of loop:
 - ▶ A for loop ends after a specified number of repetitions established by the number of columns given to an index variable.
 - ▶ A while loop ends on the basis of a logical condition.

for Loops

- ▶ One common way to use a for...end structure is:

```
for index = start:step:finish  
    statements
```

```
end
```

where the *index* variable takes on successive values in the vector created using the : operator.

Vectorization

- ▶ Sometimes, it is more efficient to have MATLAB perform calculations on an entire array rather than processing an array element by element. This can be done through *vectorization*.

for loop	Vectorization
<pre>i = 0; for t = 0:0.02:50 i = i + 1; y(i) = cos(t); end</pre>	<pre>t = 0:0.02:50; y = cos(t);</pre>

while Loops

- ▶ A while loop is fundamentally different from a for loop since while loops can run an indeterminate number of times. The general syntax is

```
while condition
  statements
end
```

where the *condition* is a logical expression. If the *condition* is true, the *statements* will run and when that is finished, the loop will again check on the *condition*.
- ▶ Note - though the *condition* may become false as the *statements* are running, the only time it matters is after all the statements have run.

Early Termination

- ▶ Sometimes it will be useful to break out of a for or while loop early - this can be done using a break statement, generally in conjunction with an if structure.

- ▶ **Example:**

```
x = 24
```

```
while (1)
```

```
    x = x - 5
```

```
    if x < 0, break, end
```

```
end
```

will produce x values of 24, 19, 14, 9, 4, and -1, then stop.

Nesting and Indentation

- ▶ Structures can be placed within other structures. For example, the `statements` portion of a `for` loop can be comprised of an `if...elseif...else` structure.
- ▶ For clarity of reading, the statements of a structure are generally indented to show which lines of controlled are under the control of which structure.

Anonymous & Inline Functions

- ▶ *Anonymous functions* are simple one-line functions created without the need for an **M-file**

```
fhandle = @(arg1, arg2, ...) expression
```

- ▶ *Inline functions* are essentially the same as anonymous functions, but with a different syntax:

```
fhandle = inline('expression', 'arg1', 'arg2',...)
```

- ▶ Anonymous functions can use workspace the values of variables upon creation, while inlines cannot.

Function Functions

- ▶ *Function functions* are functions that operate on other functions which are passed to it as input arguments. The input argument may be the handle of an anonymous or inline function, the name of a built-in function, or the name of a M-file function.
- ▶ Using function functions will allow for more dynamic programming.