MATLAB BASICS

Objectives

1. What is MATLAB and why has it been selected to be the tool of choice for DIP?
2. What programming environment does MATLAB offer?
3. What are M-files?
4. What is the difference between MATLAB scripts and functions?
5. How can I get started with MATLAB?

INTRODUCTION

MATLAB (MATrix LABoratory) is a data analysis, prototyping, and visualization tool with built-in support for matrices and matrix operations, excellent graphics capabilities, and a high-level programming language and development environment.

MATLAB has become very popular with engineers, scientists, and researchers in both industry and academia, due to many factors, among them, the availability of rich sets of specialized functions—encapsulated in *toolboxes*—for many important areas, from neural networks to finances to image processing.

MATLAB has an extensive built-in documentation. It contains descriptions of MATLAB's main functions, sample code, relevant demos, and general help pages. MATLAB documentation can be accessed in a number of different ways, from command-line text-only help to hyperlinked HTML pages (which have their online counterpart in MathWorks's web site: http://www.mathworks.com).

MATLAB's basic data type is the *matrix* (or *array*). MATLAB does not require dimensioning, that is, memory allocation prior to actual usage. All data are considered to be matrices of some sort. Single values are considered by MATLAB to be 1×1 matrices.

The MATLAB algorithm development environment provides a command-line interface, an interpreter for the MATLAB programming language, an extensive set of numerical and string manipulation functions, 2D and 3D plotting functions, and the ability to build graphical user interfaces (GUIs). The MATLAB programming language interprets commands, which shortens programming time by eliminating the need for compilation.

Programming: Script and function

The MATLAB development environment interprets commands written in the MATLAB programming language, which is extremely convenient when our major goal is to rapidly prototype an algorithm. Once an algorithm is stable, it can be compiled with the MATLAB compiler (available as an add-on to the main product) for faster execution, which is particularly important for large data sets. The MATLAB compiler MATCOM converts MATLAB native code into C++ code, compiles the C++ code, and links it with the MATLAB libraries. Compiled code may be up to 10 times as fast as their interpreted equivalent, but the speedup factor depends on how vectorized the original code was; highly optimized vectorized code may not experience any speedup at all.

For faster computation, programmers may dynamically link C routines as MATLAB functions through the MEX utility.

M-Files

M-files in MATLAB can be *scripts* that simply execute a series of MATLAB *commands* (or *statements*) or can be *functions* that accept *arguments* (*parameters*) and produce one or more *output values*. User-created functions extend the capabilities of MATLAB and its toolboxes to address specific needs or applications.

An M-file containing a *script* consists of a sequence of commands to be interpreted and executed. In addition to calls to built-in functions, scripts may also contain variable declarations, calls to user-created functions (which may be stored in separate files), decision statements, and repetition loops. Scripts are usually created using a text editor (e.g., the built-in MATLAB Editor) and stored with a meaningful name and the .m extension. Once a script has been created and saved, it can be invoked from the command line by simply typing its name.

An M-file containing a *function* has the following components:

- *Function Definition Line*: It has the form
  function [outputs] = function_name(inputs)

  The keyword function is required. The output arguments are enclosed within square brackets, whereas the input arguments are enclosed within parentheses. If the function does not return any value, only the word function is used and there is no need for the brackets or the equal sign. Function names must begin with a letter, must not contain spaces, and be limited to 63 characters in length.

- *H1 Line*: It is a single comment line that follows the function definition line. There can be no blank lines or leading spaces between the *H1 line* and the function definition line. The *H1 line* is the first text that appears when the user types
  >> help function_name
  in the CommandWindow. Since this line provides important summary information about the contents and purpose of the M-file, it should be as descriptive as possible.
- *Help Text*: It is a block of text that follows the *H1 line*, without any blank lines between the two. The help text is displayed after the *H1 line* when a user types
  help function_name at the prompt.
- *Function Body*: This contains all the MATLAB code that performs computation and assigns values to output parameters.
- *Comments*: In MATLAB, these are preceded by the % symbol.

Here is an example of a simple function (raise_to_power) that will be used in Tutorial 3.3:

```
function z = raise_to_power(val,exp)
%RAISE_TO_POWER Calculate power of a value
% z = raise_to_power(val,exp) raise val to a power with value of exp
% and store it in z.
z = val ^ exp;
```

Operators

MATLAB operators can be grouped into three main categories:

- *Arithmetic Operators*: Perform numeric computations on matrices.
- *Relational Operators*: Compare operands.
- *Logical Operators*: Perform standard logical functions (e.g., AND, NOT, and OR)

Since MATLAB considers a matrix and its standard built-in data type, the number of array and matrix operators available in MATLAB far exceeds the traditional operators found in a conventional programming language. Table 2 contains a summary of them. All operands can be real or complex.

Table 3 shows a list of some of the most useful specialized matrix operations that can also be easily performed in MATLAB.

TABLE 2 MATLAB Array and Matrix Arithmetic Operators

Operator	Name	MATLAB Function
+	Array and matrix addition	`plus(a,b)`
-	Array and matrix subtraction	`minus(a,b)`
.*	Element-by-element array multiplication	`times(a,b)`
*	Matrix multiplication	`mtimes(a,b)`
./	Array right division	`rdivide(a,b)`
.\	Array left division	`ldivide(a,b)`
/	Matrix right division	`mrdivide(a,b)`
\	Matrix left division	`mldivide(a,b)`
.^	Array power	`power(a,b)`
^	Matrix power	`mpower(a,b)`
.'	Vector and matrix transpose	`transpose(a)`
'	Vector and matrix complex conjugate transpose	`ctranspose(a)`
+	Unary plus	`uplus(a)`
—	Unary minus	`uminus(a)`
:	Colon	`colon(a,b)` or `colon(a,b,c)`

TABLE 3 Examples of MATLAB Specialized Matrix Operations

Name	MATLAB Operator or Function
Matrix transpose	Apostrophe (') operator
Inversion	`inv` function
Matrix determinant	`det` function
Flip up and down	`flipud` function
Flip left and right	`fliplr` function
Matrix rotation	`rot90` function
Matrix reshape	`reshape` function
Sum of the diagonal elements	`trace` function

Since monochrome images are essentially 2D arrays, that is, matrices, all operands in Tables 2 and 3 are applicable to images. However, the MATLAB Image Processing Toolbox (IPT) also contains specialized versions of the main arithmetic operations involving images, whose main advantage is the support of integer data classes (MATLAB math operators require inputs of class double). These functions are listed in Table 4.

The relational operators available in MATLAB parallel the ones you would expect in any programming language. They are listed in Table 5.

TABLE 4 Specialized Arithmetic Functions Supported by the IPT

Function	Description
`imadd`	Adds two images or adds a constant to an image
`imsubtract`	Subtracts two images or subtracts a constant from an image
`immultiply`	Multiplies two images (element-by-element) or multiplies a constant times an image
`imdivide`	Divides two images (element-by-element) or divides an image by a constant
`imabsdiff`	Computes the absolute difference between two images
`imcomplement`	Complements an image
`imlincomb`	Computes a linear combination of two or more images

TABLE 5 Relational Operators

Operator	Name
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

TABLE 6 Logical Operators

Operator	Name
&	AND
|	OR
~	NOT

TABLE 7 Logical Functions

Function	Description
`xor`	Performs the exclusive-or (XOR) between two operands
`all`	Returns a 1 if all the elements in a vector are nonzero or a 0 otherwise. Operates columnwise on matrices
`any`	Returns a 1 if any of the elements in a vector are nonzero or a 0 otherwise. Operates columnwise on matrices

MATLAB includes a set of standard logical operators. They are listed in Table 6. MATLAB also supports the logical functions listed in Table 7 and a vast number of functions that return a logical 1 (true) or logical 0 (false) depending on whether the value or condition in their arguments is true or false, for example, isempty(a), isequal(a,b), isnumeric(a), and many others (refer to the MATLAB online documentation).

Important Variables and Constants

MATLAB has a number of built-in variables and constants, some of which are listed in Table 8.

Number Representation

MATLAB can represent numbers in conventional decimal notation (with optional decimal point and leading plus or minus sign) as well as in scientific notation (using the letter e to specify a power-of-10 exponent). Complex numbers are represented using either i or j as a suffix for the imaginary part.

TABLE 8 Selected Built-In Variables and Constants

Name	Value Returned
`ans`	Most recent answer
`eps`	Floating-point relative accuracy
`i` (or `j`)	Imaginary unit ($\sqrt{-1}$)
`NaN` (or `nan`)	Not-a-number (e.g., the result of 0/0)
`Inf`	Infinity (e.g., the result of a division by 0)

All numbers are stored internally using the IEEE floating-point standard, resulting in a range of approximately 10^{-308}–10^{+308}.

Flow Control

The MATLAB programming language supports the usual flow control statements found in most other contemporary high-level programming languages: if (with optional else and elseif) and switch decision statements, for and while loops and the associated statements (break and continue), and the try...catch block for error handling. Refer to the online documentation for specific syntax details.

Code Optimization

As a result of the matrix-oriented nature of MATLAB, the MATLAB programming language is a *vectorized language*, which means that it can perform many operations on numbers grouped as vectors or matrices without explicit loop statements. Vectorized code is more compact, more efficient, and parallelizable. In addition to using vectorized loops, MATLAB programmers are encouraged to employ other optimization tricks, such as preallocating the memory used by arrays. These ideas—and their impact on the execution speed of MATLAB code—are discussed in Tutorial 3.3.

Input and Output

Basic input and output functionality can be achieved with functions input (to request user input and read data from the keyboard) and disp (to display a text or array on the screen). MATLAB also contains many support functions to read from and write to files.

GRAPHICS AND VISUALIZATION

MATLAB has a rich set of primitives for plotting 2D and 3D graphics. The next LAB will explore some of the 2D plotting capabilities in connection with the plotting of image histograms and transformation functions, whereas Tutorials in the optimal topics will show examples of 3D plots in connection with the design of image processing filters in the frequency domain.

MATLAB also includes a number of built-in functions to display (and inspect the contents of) images, some of which will be extensively used throughout the book (and discussed in more detail in the image geometric transformation).

LAB1: MATLAB—A GUIDED TOUR

Goal

The goal of LAB1 is to give a brief overview of the MATLAB environment.

Objectives

- Become familiar with the working environment in MATLAB.
- Learn how to use the working directory and set paths.
- Become familiar with the MATLAB help system.
- Explore functions that reveal system information and other useful built-in functions.

Procedure

The environment in MATLAB has a simple layout, consisting of several key areas (Figure 1). A description of each is given here:

- A: This pane consists of two tabbed areas: one that displays all files in your current *working directory*, and another that displays your *workspace*. The workspace lists all the variables you are currently using.
- B: This pane shows your *history* of commands.
- C: This is where you can modify your current *working directory*. To change the current directory, you can either type it directly in the text box or click on the button to select the directory. You can also change the working directory using the path command. See help documents for more information.
- D: This is the *command window*. Here you control MATLAB by typing in commands.

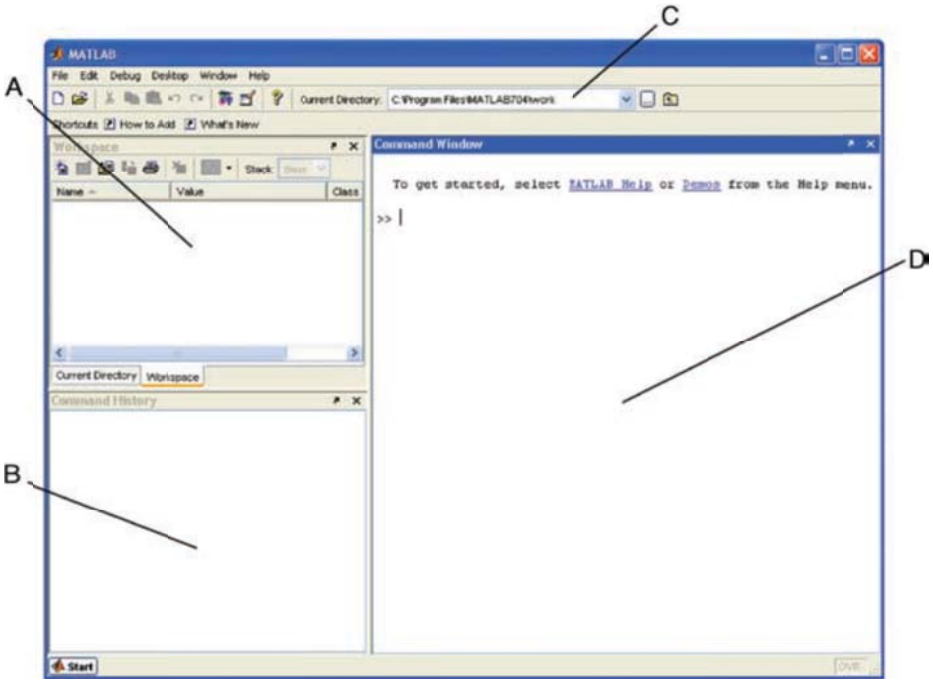

FIGURE 1 MATLAB environment.

In order for you to use files such as M-files and images, MATLAB must know where these files are located. There are two ways this can be done: by setting the *current directory* to a specific location, or by adding the location to a list of set paths known to MATLAB. The *current directory* should be used for temporary locations or when you need to access a directory only once. This directory is reset every time MATLAB is restarted. To change the current directory, see description C above. Setting a path is a permanent way of telling MATLAB where files are located—the location will remain in the settings when MATLAB is closed. The following steps will illustrate how to set a path in MATLAB:

1. From the **File** menu, select **Set Path...**
   The paths are presented in the list box in decreasing precedence order.
2. If the directory you wish to add has subfolders within it, then use the **Add with Subfolders...** button. If your directory only contains files, you can use the **Add Folder...** button. To change the precedence of your directory, use the **Move** buttons.
3. **Save** your changes and close the **Set Path** window.

The help system in MATLAB is very useful. It provides information about functions, such as syntax, descriptions, required and optional input parameters, and output parameters. It also includes a number of demos. The following steps will show you how to access help documents and navigate the help window.

4. To access the help system, type doc in the command window. Open the help system now.

The left pane of the help window displays a tree of help information. The right pane will show the help document that you choose. If you know exactly what function you need help for, you can use the *doc*, *helpwin*, or *help* commands to directly access the help information.

5. In the command window in MATLAB, execute the following commands to view the help document for the function computer.
   - help computer
   - helpwin computer
   - doc computer

Question 1 What is the difference between the commands *help, helpwin*, and *doc*?

There are several commands that can be used to gain information about the system. Explore them in this step.

6. Execute the following commands, one at a time, to see their function.
   realmin
   realmax
   bitmax
   computer
   ver
   version
   hostid
   license

Question 2 What is the difference between *ver* and *version*?

7. For some MATLAB humor, repeatedly type the command why.

LAB2: MATLAB DATA STRUCTURES

Goal

The goal of LAB2 is to learn how to create, initialize, and access some of the most useful data structures available in MATLAB.

Objectives

- Learn how to use MATLAB for basic calculations involving variables, arrays, matrices, and vectors.
- Explore multidimensional and cell arrays.
- Review matrix operations.
- Learn how to use MATLAB structures.
- Explore useful functions that can be used with MATLAB data structures.

Procedure

1. Execute the following lines of code one at a time in the *Command Window* to see how MATLAB can be used as a calculator.

```
2 + 3
2*3 + 4*5 + 6*7;
```

Question 1 What is the variable ans used for?

Question 2 What is the purpose of using a semicolon (;) at the end of a statement?

2. Perform calculations using variables.

```
fruit_per_box = 20; num_of_boxes = 5;
total_num_of_fruit = fruit_per_box * num_of_boxes
```

Question 3 Experiment with creating your own variables. Are variables case sensitive?
Question 4 What is the value/purpose of these variables: pi, eps, inf, i? Is it possible to overwrite any of these variables? If so, how can it be undone?

3. Execute the commands who and whos, one at a time, to see their function and the difference between them.

   There are several commands that will keep the MATLAB environment clean. Use them whenever you feel your command window or workspace is cluttered with statements and variables.

4. Clear a variable in the workspace. After execution, note how the variable disappears from the workspace.

```
clear fruit_per_box
```

5. Clear the command window and all variables with the following lines of code (one at a time to see their effects individually).

```
clc
clear all
```

6. Create a 3 × 3 matrix by executing the following line of code.

```
A = [1 2 3;4 5 6;7 8 9]
```

Question 5 What is the use of the semicolon in this statement?

The Colon Operator

7. A very useful operator in MATLAB is the colon (:). It can be used to create a vector of numbers.

```
1:5
```

8. A third parameter determines how to count between the starting and ending numbers. It is specified between the start and end values.

```
1:1:5
5:-1:1
1:2:9
9:-2:1
```

Question 6 Write a statement that will generate a vector of values ranging from 0 to π in increments of $\pi/4$.

9. The colon operator can also be used to return an entire row or column of a matrix.

```
A = [1 2 3;4 5 6;7 8 9]
A(:,1)
A(1,:)
```

Question 7 Write a line of code that would generate the same 3×3 matrix as in the variable A above, but using the colon operator to generate the sequence of numbers in each row instead of explicitly writing them.

10. The colon operator can be replaced with the function colon, which performs the same operation.

```
colon(1,5)
```

As seen in the steps above, creating a vector of evenly spaced numbers is easily done with the colon (:) operator when we know where the vector starts, ends, and how large the space in between each value is. In certain cases, we may wish to create a vector of numbers that range between two numbers, but we only know the quantity of values needed (for example, create a vector that consists of 4 values between $\pi/4$ and π) . To do this, we use the linspace function.

11. Execute this command to see how the function linspace operates.

```
linspace(pi/4,pi,4)
```

12. Compare the result from the previous step with these values.

```
pi/4
pi/2
3*pi/4
pi
```

Special Built-In Matrices

MATLAB has several built-in functions that will generate frequently used matrices automatically.

13. Execute the following lines of code one at a time.

```
zeros(3,4)
ones(3,4)
ones(3,4) * 10
rand(3,4)
randn(3,4)
```

Question 8 What is the difference between the functions rand(M,N) and randn(M,N)?

Matrix Concatenation

Concatenation of matrices is done with brackets ([]) or using the cat function. Take, for example, the statement

```
A = [1 2 3;4 5 6;7 8 9]
```

The brackets are combining three rows. Instead of explicitly defining each row all at once, they can be defined individually as vectors and then combined into a matrix using brackets.

14. Combine the three individual vectors into a 3 × 3 matrix.

```
X = [1 2 3]; Y = [4 5 6]; Z = [7 8 9];
A = [X;Y;Z]
B = cat(1,X,Y,Z)
```

Similarly, the brackets can be used to delete a row of a matrix.

15. Delete the last row (row 3) of the matrix A. Note how the colon operator is used to specify the entire row.

```
A(3,:) = []
```

A vector with *N* elements is an array with one row and *N* columns. An element of a vector can be accessed easily by addressing the number of the element, as in X(5), which would access the fifth element of vector X. An element of a two-dimensional matrix is accessed by first specifying the row, then the column, as in X(2,5), which would return the element at row 2, column 5. Matrices of dimensions higher than 2 can be accessed in a similar fashion. It is relevant to note that arrays in MATLAB are 1-based—the first element of an array is assigned or accessed using 1, as opposed to 0, which is the standard in many programming languages.

16. Use the ones and rand functions to create multidimensional arrays.

```
A = ones(4,3,2);
B = rand(5,2,3);
size(A)
size(B)
disp(A)
disp(B)
```

Question 9 What does the size function do?
Question 10 What does the disp function do?

Operations Involving Matrices

Performing arithmetic operations on matrices can be achieved with the operators + - * /. The default for the multiply (*) and divide (/) operators is matrix multiplication and matrix division. To perform arithmetic operations on individual elements of a matrix, precede the operator with a dot (.).

17. Perform matrix multiplication on two matrices.

```
X = [1 2 -2; 0 -3 4; 7 3 0]
Y = [1 0 -1; 2 3 -5; 1 3 5]
X * Y
```

18. Perform element-by-element multiplication.

```
X .* Y
```

19. Perform another matrix multiplication on two matrices.

```
X = eye(3,4)
Y = rand(4,2)
X * Y
Y * X
```

Question 11 Why did the last operation fail?

20. Use the diag and trace functions to perform operations on the diagonal of a matrix.

```
Y = rand(3,3)*4
Y_diag = diag(Y)
Y_trace = trace(Y)
```

Question 12 What does the diag function do?

Question 13 What does the trace function do?

Question 14 Write an alternative statement that would produce the same results as the trace function.

21. Calculate the transpose of a matrix.

```
Y
Y_t = Y'
```

22. Calculate the inverse of a matrix and show that $YY^{-1} = Y^{-1}Y = I$, where I is the identity matrix.

```
Y_inv = inv(Y)
Y * Y_inv
Y_inv * Y
```

23. Calculate the determinant of a matrix.

```
Y_det = det(Y)
```

Cell Array

As demonstrated earlier, a matrix is the fundamental data type in MATLAB. It resembles the classical definition of an array as a homogeneous data structure, that is, one in which all its components are of the same type. *Cell arrays*, on the other hand, are another type of array where each cell can be any data type allowed by MATLAB. Each cell is independent of another and, therefore, can contain any data type that MATLAB supports. When using cell arrays, one must be careful when accessing or assigning a value to a cell; instead of parentheses, curly braces ({}) must be used.

24. Execute the following lines of code one at a time to see how cell arrays are handled in MATLAB.

```
X{1} = [1 2 3;4 5 6;7 8 9]; %Cell 1 is a matrix
X{2} = 2+3i; %Cell 2 is complex
X{3} = 'String'; %Cell 3 is a string
X{4} = 1:2:9; %Cell 4 is a vector
X
celldisp(X)
X(1)
X{1}
```

Question 15 What does the celldisp function do?

Question 16 What does the percent (%) character do?

Question 17 What is the difference between the last two lines in the code above (X(1) as opposed to X{1})?

There is another way to assign values to a cell array that is syntactically different, but yields the same results. Note in the next step how the cell index is enclosed within normal parentheses (()), but the data that will be saved to the cell is encapsulated by curly braces ({}).

25. Execute this line to see another way of assigning cell array values.

```
X(1) = {[1 2 3;4 5 6;7 8 9]};
```

26. The next few lines of code will demonstrate proper and improper ways of cell array assignment when dealing with strings.

```
X(3) = 'This produces an error'
X(3) = {'This is okay'}
X{3} = 'This is okay too'
```

Structures

Structures are yet another way of storing data in MATLAB. The syntax for structures is similar to that of other programming languages. We use the dot (.) operator to refer to different fields in a structure. Structures with identical layout (number of fields, their names, size, and meaning) can be combined in an array (of structures).

27. Create an array of two structures that represents two images and their sizes.

```
my_images(1).imagename = 'Image 1';
my_images(1).width = 256;
my_images(1).height = 256;
my_images(2).imagename = 'Image 2';
my_images(2).width = 128;
my_images(2).height = 128;
```

28. View details about the structure and display the contents of a field.

```
my_images(1)
my_images(2).imagename
```

29. Display information about the structure.

```
num_of_images = prod(size(my_images))
fieldnames(my_images)
class(my_images)
isstruct(my_images)
isstruct(num_of_images)
```

Question 18 What does the fieldnames function do?

Question 19 What does it mean when the result from the function isstruct is 1? What does it mean when it is 0?

Question 20 Use the help system to determine what function can be used to delete a field from a structure.