MATLAB INTRODUCTION & PROGRAMMING

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve specific classes of problems.

Some of the areas in which **toolboxes** are available include

• **Signal processing**: This Toolbox provides functions and applications to generate, measure, transform, filter, and visualize signals.

• **Control systems:** Provides industry-standard algorithms and applications for systematically analysing, designing, and tuning linear control systems.

• **Neural networks**: Provides functions and apps for modeling complex nonlinear systems that are not easily modelled with a closed-form equation. Neural Network Toolbox supports supervised learning with feed-forward, radial basis, and dynamic networks. It also supports unsupervised learning with self-organizing maps and competitive layers

• **Fuzzy logic:** This toolbox lets you model complex system behaviours using simple logic rules, and then implement these rules in a fuzzy inference system.

• **Wavelets:** Provides functions and an application for developing wavelet-based algorithms for the analysis, synthesis, denoising, and compression of signals and images. The toolbox lets you explore wavelet properties and applications such as speech and audio processing, image and video processing, biomedical imaging, and 1-D and 2-D applications in communications and geophysics.

The MATLAB System

The MATLAB system consists of five main parts:

1. **Development Environment:** This is a set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.

2. **The MATLAB Mathematical Function Library**: This is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

3. **The MATLAB Language:** This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both “programming in the small” to rapidly create quick and dirty throw-away programs, and “programming in the large” to create large and complex application programs.

4. Graphics: MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

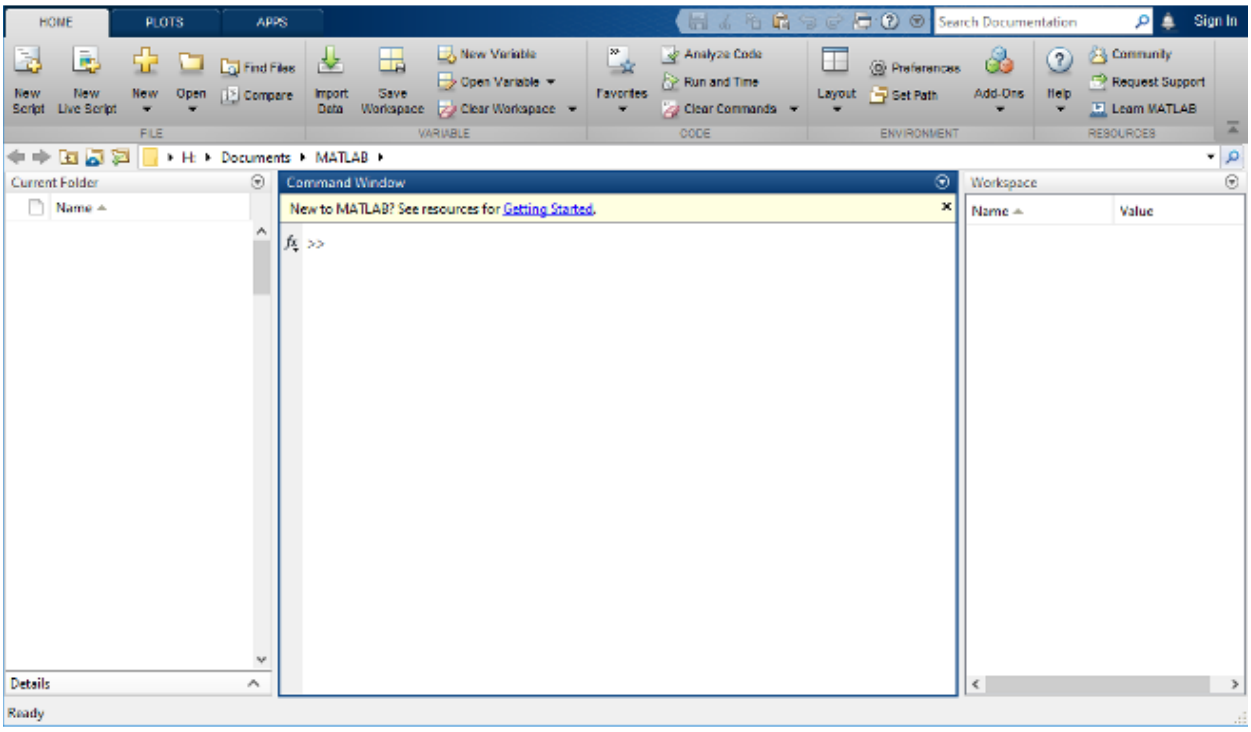
5. **The MATLAB External Interfaces/API:** This is a library that allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

Features

• Basic data element: *matrix*
    - Auto dimensioning
    - eliminates data type declarations
    - ease of programming
    - Operator overloading
    - Case sensitive
• Advanced visualization
    - 2D and 3D
    - Simple programming
    - colour graphics
• Open environment

Desktop basics

When you start MATLAB®, the desktop appears in its default layout.



The desktop includes these panels:

Current Folder — Access your files.

Command Window — Enter commands at the command line, indicated by the prompt (>>).

Workspace — Explore data that you create or import from files.

As you work in MATLAB, you issue commands that create variables and call functions. For example, create a variable named a by typing this statement at the command line:

```
a = 1
```

MATLAB adds variable a to the workspace and displays the result in the Command Window.

```
a =

     1
```

Create a few more variables.

```
b = 2

b =

     2

c = a + b

c =
```

3

d = cos(a)

d =

0.5403

When you do not specify an output variable, MATLAB uses the variable ans, short for *answer*, to store the results of your calculation.

sin(a)

ans =

0.8415

If you end a statement with a semicolon, MATLAB performs the computation but suppresses the display of output in the Command Window.

e = a*b;

You can recall previous commands by pressing the up- and down-arrow keys, ↑ and ↓. Press the arrow keys either at an empty command line or after you type the first few characters of a command. For example, to recall the command b = 2, type b, and then press the up-arrow key.

Matrices and Arrays

MATLAB is an abbreviation for "matrix laboratory." While other programming languages mostly work with numbers one at a time, MATLAB® is designed to operate primarily on whole matrices and arrays.

All MATLAB variables are multidimensional *arrays*, no matter what type of data. A *matrix* is a two-dimensional array often used for linear algebra.

Array Creation

To create an array with four elements in a single row, separate the elements with either a comma (,) or a space.

a = [1 2 3 4]

a = *1×4*

1 2 3 4

This type of array is a *row vector*.

To create a matrix that has multiple rows, separate the rows with semicolons.

a = [1 2 3; 4 5 6; 7 8 10]

a = *3×3*

1 2 3

4 5 6

7 8 10

Another way to create a matrix is to use a function, such as ones, zeros, or rand. For example, create a 5-by-1 column vector of zeros.

z = zeros(5,1)

z = *5×1*

0

0

0

0

0

Matrix and Array Operations

MATLAB allows you to process all of the values in a matrix using a single arithmetic operator or function.

a + 10

ans = *3×3*

11 12 13

14 15 16

17 18 20

sin(a)

ans = *3×3*

0.8415 0.9093 0.1411

-0.7568 -0.9589 -0.2794

0.6570 0.9894 -0.5440

To transpose a matrix, use a single quote (''):

a'

ans = *3×3*

1 4 7

2 5 8

3 6 10

You can perform standard matrix multiplication, which computes the inner products between rows and columns, using the * operator. For example, confirm that a matrix times its inverse returns the identity matrix:

```
p = a*inv(a)
p = 3×3
1.0000         0         -0.0000
0              1.0000    0
0              0          1.0000
```

Notice that p is not a matrix of integer values. MATLAB stores numbers as floating-point values, and arithmetic operations are sensitive to small differences between the actual value and its floating-point representation. You can display more decimal digits using the format command:

```
format long
p = a*inv(a)
p = 3×3
1.000000000000000              0                         -0.000000000000000
0                              1.000000000000000          0
0                              0                          0.999999999999998
```

Reset the display to the shorter format using

```
format short
```

format affects only the display of numbers, not the way MATLAB computes or saves them.

To perform element-wise multiplication rather than matrix multiplication, use the .* operator:

```
p = a.*a
p = 3×3
1     4   9
16  25  36
49  64  100
```

The matrix operators for multiplication, division, and power each have a corresponding array operator that operates element-wise. For example, raise each element of a to the third power:

```
a.^3
```

ans = *3×3*

1 8 27

64 125 216

343 512 1000

Concatenation

Concatenation is the process of joining arrays to make larger ones. In fact, you made your first array by concatenating its individual elements. The pair of square brackets [] is the concatenation operator.

A = [a,a]

A = *3×6*

1 2 3 1 2 3

4 5 6 4 5 6

7 8 10 7 8 10

Concatenating arrays next to one another using commas is called *horizontal* concatenation. Each array must have the same number of rows. Similarly, when the arrays have the same number of columns, you can concatenate *vertically* using semicolons.

A = [a; a]

A = *6×3*

1 2 3

4 5 6

7 8 10

1 2 3

4 5 6

7 8 10

Complex Numbers

Complex numbers have both real and imaginary parts, where the imaginary unit is the square root of -1.

sqrt(-1)

ans = 0.0000 + 1.0000i

To represent the imaginary part of complex numbers, use either i or j .

c = [3+4i, 4+3j; -i, 10j]

c = *2×2 complex*

3.0000 + 4.0000i 4.0000 + 3.0000i

0.0000 - 1.0000i 0.0000 +10.0000i

Operators

• **Arithmetic operators**

+ : Addition **-** : Subtraction

***** : Multiplication **/** : Division

\ : Left Division ^ : Power

• **Relational operators**

< : Less than <= : Less than or equal to

> : Greater than >= : Greater than or equal to

== : Equal ~= : Not equal

• **Logical operators**

& : AND | : OR ~ : NOT

• **Array operations**

Matrix operations preceded by a **.***(dot)* indicates array operation.

• **Simple math functions**

sin, cos, tan, asin, acos, atan, sinh, cosh ….

log, log10, exp, sqrt …

• **Special constants**

pi, inf, i, j, eps …..

Control Flow Statements

• **for loop** : for k = 1:m

.....

end

• **while loop** : while *condition*

.....

end

• **if statement** : if *condition1*

.....

else if *condition2*

.....

else

.....

end

Array indexing

Every variable in MATLAB® is an array that can hold many numbers. When you want to access selected elements of an array, use indexing.

For example, consider the 4-by-4 magic square A:

A = magic(4)

A = *4×4*

16 2 3 13

5 11 10 8

9 7 6 12

4 14 15 1

There are two ways to refer to a particular element in an array. The most common way is to specify row and column subscripts, such as

A(4,2)

ans = 14

Less common, but sometimes useful, is to use a single subscript that traverses down each column in order:

A(8)

ans = 14

Using a single subscript to refer to a particular element in an array is called *linear indexing*.

If you try to refer to elements outside an array on the right side of an assignment statement, MATLAB throws an error.

test = A(4,5)

Index exceeds matrix dimensions.

However, on the left side of an assignment statement, you can specify elements outside the current dimensions. The size of the array increases to accommodate the newcomers.

```
A(4,5) = 17
A = 4×5
    16     2     3    13     0
     5    11    10     8     0
     9     7     6    12     0
     4    14    15     1    17
```

To refer to multiple elements of an array, use the colon operator, which allows you to specify a range of the form start:end. For example, list the elements in the first three rows and the second column of A:

```
A(1:3,2)
ans = 3×1
     2
    11
     7
```

The colon alone, without start or end values, specifies all of the elements in that dimension. For example, select all the columns in the third row of A:

```
A(3,:)
ans = 1×5
     9     7     6    12     0
```

The colon operator also allows you to create an equally spaced vector of values using the more general form start:step:end.

```
B = 0:10:100
B = 1×11
     0    10    20    30    40    50    60    70    80    90   100
```

If you omit the middle step, as in start:end, MATLAB uses the default step value of 1.

Workspace variables

The *workspace* contains variables that you create within or import into MATLAB® from data files or other programs. For example, these statements create variables A and B in the workspace.

A = magic(4);

B = rand(3,5,2);

You can view the contents of the workspace using whos.

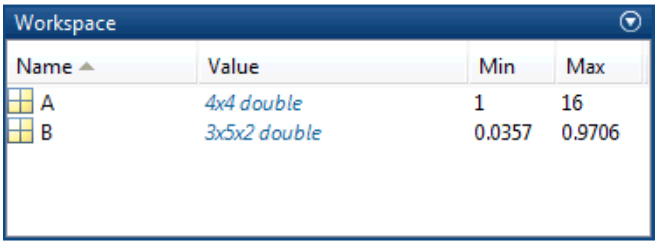
whos

Name Size Bytes Class Attributes

A 4x4 128 double

B 3x5x2 240 double

The variables also appear in the Workspace pane on the desktop.



Workspace variables do not persist after you exit MATLAB. Save your data for later use with the save command,

save myfile.mat

Saving preserves the workspace in your current working folder in a compressed file with a .mat extension, called a MAT-file.

To clear all the variables from the workspace, use the clear command.

Restore data from a MAT-file into the workspace using load.

load myfile.mat

Some useful commands

who whos clc clf clear load save pause help pwd ls dir

Getting started with MATLAB

• Using **Windows Explorer,** create a folder *user_name* in the directory *c:\cslab\batch_index*

For uniformity, let the *batch_index* be *A1, A2, A3, B1, B2,* or *B3* and *user_name* be the *roll number*

• Invoke **MATLAB**

Running MATLAB creates one or more windows on your monitor. Of these the ***Command Window*** is the primary place where you interact with MATLAB. The prompt >> is displayed in

the Command window and when the Command window is active, a blinking cursor should appear to the right of the prompt.

Interactive Computation, Script files

Objective: Familiarize with MATLAB Command window, do some simple calculations using array and vectors.

Exercises

The basic variable type in MATLAB is a matrix. To declare a variable, simply assign it a value at the MATLAB prompt. Let's try the following examples:

1. Elementary matrix/array operations

To enter a row vector

```
>> a = [5 3 7 8 9 2 1 4]
>> b = [2 6 4 3 8 7 9 5]; % output display suppressed
```

To enter a matrix with real elements

```
>> A = [5 3 7; 8 9 2; 1 4.2 6e-2]
```

To enter a matrix with complex elements

```
>> X = [5+3j 7+8j; 9+2j 1+4j];
```

Transpose of a matrix

```
>> A_trans = A'
```

Determinant of a matrix

```
>> A_det = det(A)
```

Inverse of a matrix

```
>> A_inv = inv(A)
```

Matrix multiplication

```
>> C = A * A_trans
```

Array multiplication

```
>> c = a .* b % a.*b denotes element-by-element multiplication.
% vectors a and b must have the same dimensions.
```

2. Few useful commands

```
>>who % lists variables in workspace (The MATLAB workspace consists of the variables you create and store in memory during a MATLAB session.)
>>whos % lists variables and their sizes
```

>> help inv

The online help system is accessible using the help command. Help is available for functions e.g. help inv and for Punctuation help punct. A useful command to get started is intro, which covers the basic concepts in MATLAB language. Demonstration programs can be started with the command demo. Demos can also be invoked from the **Help Menu** at the top of the window.

>>lookfor inverse

The function *lookfor* becomes useful when one is not sure of the MATLAB function name.

>>clc %clear command window

>> save session1 % workspace variables stored in session1.mat

>>save myfile.dat a b -ascii % saves a,b in myfile.dat in ascii format

>>clear % clear workspace

>>who

>>load session1 % To load variables to workspace

>>who

>>pwd % present working directory

>>disp(' I have successfully completed MATLAB basics')

More Matrix manipulations

A = [1 3 5;2 4 4; 0 0 3]

B = [1 0 1; 2 2 2; 3 3 3]

Accessing a submatrix A(1:2,2:3); A(1,:); B(:,2)

Concatenating two matrices D = [A B]; E = [A;B];

Adding a row A(4,:) = [1 1 0]

Deleting a column B(:,2) = []

Matrix generation functions

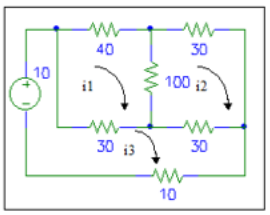
zeros(3,3) - 3x3 matrix of zeroes

ones(2,2) - 2x2 matrix of ones

eye(3) - identity matrix of size 3

rand(5) - 5x5 matrix of random numbers

✓ ***Find the loop currents of the circuit given in Fig. Below..***



Network equations are:

$$\begin{bmatrix} 170 & -100 & -30 \\ -100 & 160 & -30 \\ -30 & -30 & 70 \end{bmatrix} \begin{bmatrix} i1 \\ i2 \\ i3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 10 \end{bmatrix}$$

Sample Solution a:

(using command line editor)

>> Z = [170 -100 -30; -100 160 -30; -30 -30 70];

>> v = [0; 0; 10];

>>i = inv(Z) * v

✓ **Creating script files**

*Editing a file:***Home Menu New -> Script**, invokes MATLAB Editor/Debugger.

Save file with extension ***.m***

Sample Solution b: *(using ex_1b.m file)*

Edit the following commands in the Editor and save as ex_1b.m.

A line beginning with % sign is a comment line.

```
% ex_1b.m
clear; clc;
% Solution of Network equations
Z = [170 -100 -30; -100 160 -30; -30 -30 70];
v = [0; 0; 10];
disp('The mesh currents are : ')
i = inv(Z)*v
```

Typing ex_1b at the command prompt will run the script file, and all the 7 commands will be executed sequentially. The script file can also be executed from the **Run** icon in the MATLAB Editor.

Sample Solution c: *(interactive data input using ex_1c.m)*

```
% ex_1c.m
% Interactive data input and formatted output
clear; clc;
```

```
% Solution of Network equations
Z = input('Enter Z : ');
v = input('Enter v : ');
i = Z\v;      % Left division - computes inv(Z)*v
disp('The results are : ')
fprintf('i1 = %g A, i2 = %g A, i3 = %g A \n', i(1),i(2),i(3))
```

Results:

```
i = 0.1073
0.1114
0.2366
```

Function File

A *function file* is also an m-file, just like a script file, except it has a function definition line at the top that defines the input and output explicitly.

function<output_list> = fname <input_list>

Save the file as *fname.m* The filename will become the name of the new command for MATLAB. Variables inside a function are local. Use *global* declaration to share variables.

✓ Create a function file for rectangular to polar conversion

Sample solution:

```
function [r,theta] = r2p(x)
% r2p - function to convert from rectangular to polar
% Call syntax: [r,theta] = r2p(x)
% Input: x = complex number in the form a+jb
% Output: [r,theta] = the complex number in polar form
r = abs(x);
theta = angle(x)*180/pi;
```

Save the code in a file *r2p.m*.

Executing the function *r2p in the Command Window*

```
>> y = 3+4j;
>> [r,th] = r2p(y)
```

✓ Write a function factorial to compute the factorial for any integer n.

Programming Tips

Writing efficient MATLAB code requires a programming style that generates small functions that are vectorised. Loops should be avoided. The primary way to avoid loops is to use toolbox functions as much as possible.

MATLAB functions operate on arrays just as easily as they operate on scalars. For example, if **x** is an array, then cos(**x**) returns an array of the same size as **x** containing the cosine of each element of **x**.

Avoiding loops - Since MATLAB is an interpreted language, certain common programming habits are intrinsically inefficient. The primary one is the use of *for* loops to perform simple operations over an entire matrix or vector. Wherever possible, you should try to find a vector function that will accomplish the desired result, rather than writing loops.

For example, to sum all the elements of a matrix

By using *for* loops:

```
[Nrows, Ncols]=size(x);
xsum=0.0;
for m=1:Nrows
for n=1:Ncols
xsum=xsum+x(m,n);
end;
end;
```

Vectorised code:

```
xsum=sum(sum(x));
```

Repeating rows or columns - If the matrix has all same values use ones(M,N) and zeros(M,N).

To replicate a column vector x to create a matrix that has identical columns,

```
x=(12:-2:0)'; X= x*ones(1,11).
```

Experience has shown that MATLAB is easy to learn and use. You can learn a lot by exploring the Help & Demos provided and trying things out. One way to get started is by viewing and executing script files provided with the software. Don't try to learn everything at once. Just get going and as you need new capabilities find them, or, if necessary, make them yourself.