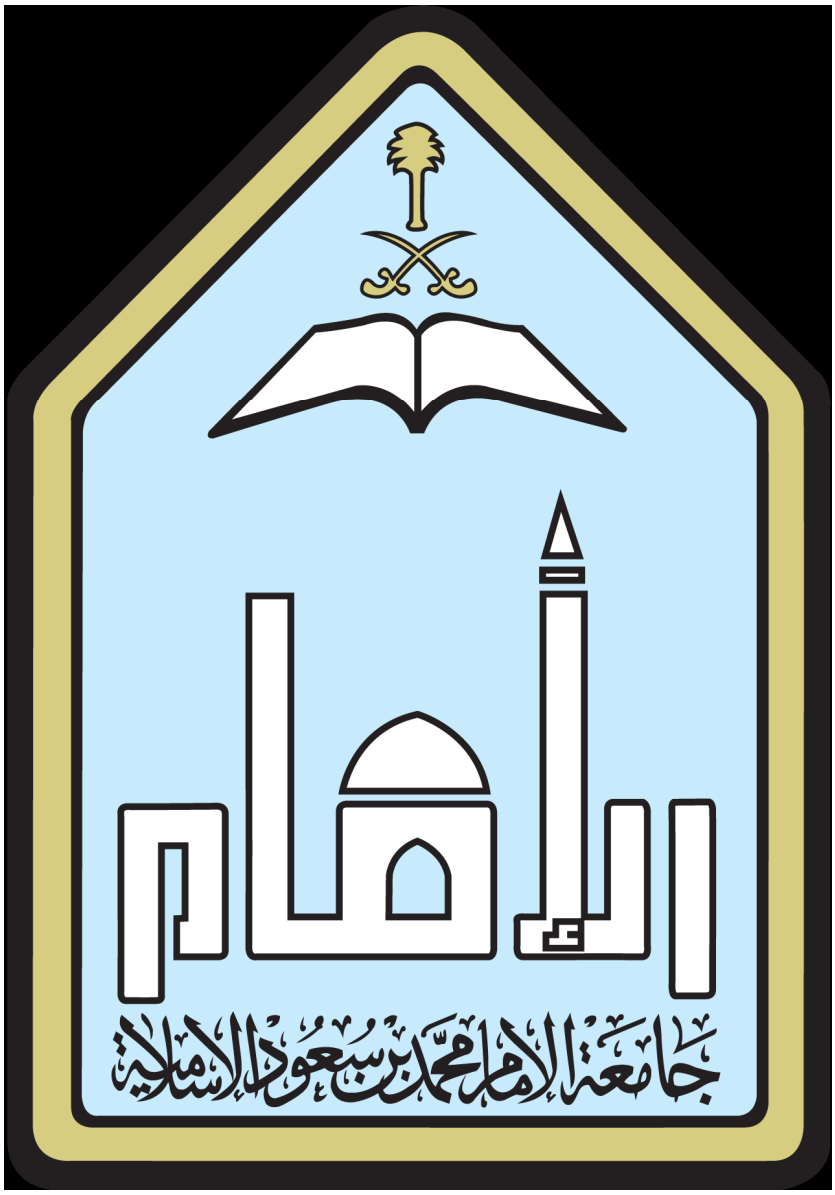
Al Imam Mohammad Ibn Saud Islamic University

College of Engineering

Department of Electrical Engineering



MATLAB

What Is MATLAB?

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
- 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 noninteractive 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, which together represent 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 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 particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

The MATLAB System

The MATLAB system consists of five main parts:

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 complete large and complex application programs.

The MATLAB Working Environment.

This is the set of tools and facilities that you work with as the MATLAB user or programmer. It includes facilities for managing the variables in your workspace and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling M-files, MATLAB's applications.

Handle Graphics.

This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete Graphical User Interfaces on your MATLAB applications.

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.

The MATLAB Application Program Interface (API).

This is a library that allows you to write C and Fortran programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

Syntax

The MATLAB application is built around the MATLAB scripting language. Common usage of the MATLAB application involves using the Command Window as an interactive mathematical shell or executing text files containing MATLAB code.

Variables

Variables are defined using the assignment operator, `=`. MATLAB is a weakly typed programming language because types are implicitly converted. It is an inferred typed language because variables can be assigned without declaring their type, except if they are to be treated as symbolic objects,[13] and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function. For example:

```
>>x=17
x =
 17

>>x='hat'
x =
hat

>>x=[3*4,pi/2]
x =
   12.0000    1.5708

>>y=3*sin(x)
y =
   -1.6097    3.0000
```

Vectors and matrices

A simple array is defined using the colon syntax: *initial*`:`*increment*`:`*terminator*. For instance:

```
>>array=1:2:9
array=
13579
```

defines a variable named `array` (or assigns a new value to an existing variable with the name `array`) which is an array consisting of the values 1, 3, 5, 7, and 9. That is, the array starts at 1 (the *initial* value), increments with each step from the previous

value by 2 (the *increment* value), and stops once it reaches (or to avoid exceeding) 9 (the *terminator* value).

```
>>array=1:3:9
array=
147
```

the *increment* value can actually be left out of this syntax (along with one of the colons), to use a default value of 1.

```
>>ari=1:5
ari=
12345
```

assigns to the variable named `ari` an array with the values 1, 2, 3, 4, and 5, since the default value of 1 is used as the incrementer.

Indexing is one-based,[14] which is the usual convention for matrices in mathematics, although not for some programming languages such as C, C++, and Java.

Matrices can be defined by separating the elements of a row with blank space or comma and using a semicolon to terminate each row. The list of elements should be surrounded by square brackets: []. Parentheses: () are used to access elements and subarrays (they are also used to denote a function argument list).

```
>>A=[163213;510118;96712;415141]
A=
163213
510118
96712
415141

>>A(2,3)
ans=
11
```

Sets of indices can be specified by expressions such as "2:4", which evaluates to [2, 3, 4]. For example, a submatrix taken from rows 2 through 4 and columns 3 through 4 can be written as:

```
>>A(2:4,3:4)
ans=
118
712
141
```

A square identity matrix of size *n* can be generated using the function *eye*, and matrices of any size with zeros or ones can be generated with the functions *zeros* and *ones*, respectively.

```
>>eye(3,3)
ans=
100
010
001

>>zeros(2,3)
ans=
000
000

>>ones(2,3)
ans=
111
111
```

Transposing a vector or a matrix is done either by the function *transpose* or by adding prime after a dot to the matrix. Without the dot MATLAB will perform conjugate transpose.

```
>>A=[1;2],B=A.',C=transpose(A)
A=
1
2
B=
12
C=
12

>>D=[03;15],D.'
D=
03
15
ans=
01
35
```

Most MATLAB functions can accept matrices and will apply themselves to each element. For example, `mod(2*J,n)` will multiply every element in "J" by 2, and then reduce each element modulo "n". MATLAB does include standard "for" and "while"

loops, but (as in other similar applications such as R), using the vectorized notation often produces code that is faster to execute. This code, excerpted from the function *magic.m*, creates a magic square *M* for odd values of *n* (MATLAB function `meshgrid` is used here to generate square matrices I and J containing 1:n).

```
[J,I]=meshgrid(1:n);
A=mod(I+J-(n+3)/2,n);
B=mod(I+2*J-2,n);
M=n*A+B+1;
```

Structures

MATLAB has structure data types. Since all variables in MATLAB are arrays, a more adequate name is "structure array", where each element of the array has the same field names. In addition, MATLAB supports dynamic field names(field look-ups by name, field manipulations, etc.). Unfortunately, MATLAB JIT does not support MATLAB structures, therefore just a simple bundling of various variables into a structure will come at a cost.

Functions

When creating a MATLAB function, the name of the file should match the name of the first function in the file. Valid function names begin with an alphabetic character, and can contain letters, numbers, or underscores. Functions are often case sensitive.

Function handles

MATLAB supports elements of lambda calculus by introducing function handles,[18] or function references, which are implemented either in .m files or anonymous[19]/nested functions.[20]

Classes and object-oriented programming

MATLAB supports object-oriented programming including classes, inheritance, virtual dispatch, packages, pass-by-value semantics, and pass-by-reference semantics.[21] However, the syntax and calling conventions are significantly different from other languages. MATLAB has value classes and reference classes, depending on whether the class has *handle* as a super-class (for reference classes) or not (for value classes).[22]

Method call behavior is different between value and reference classes. For example, a call to a method

```
object.method();
```

can alter any member of *object* only if *object* is an instance of a reference class.

An example of a simple class is provided below.

```
classdefhello
```

```
methods
functiongreet(this)
disp('Hello!')
end
end
end
```

When put into a file named hello.m, this can be executed with the following commands:

```
>>x=hello;
>>x.greet();
Hello!
```

Graphics and graphical user interface programming

MATLAB supports developing applications with graphical user interface (GUI) features. MATLAB includes GUIDE[23] (GUI development environment) for graphically designing GUIs.[24] It also has tightly integrated graph-plotting features. For example, the function *plot* can be used to produce a graph from two vectors *x* and *y*. The code:

```
x=0:pi/100:2*pi;
y=sin(x);
plot(x,y)
```

produces the following figure of the sine function:

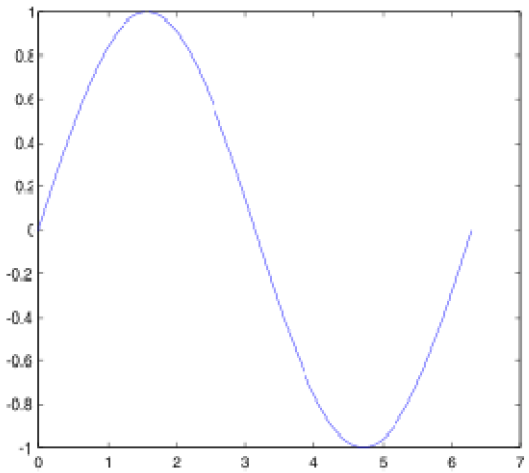
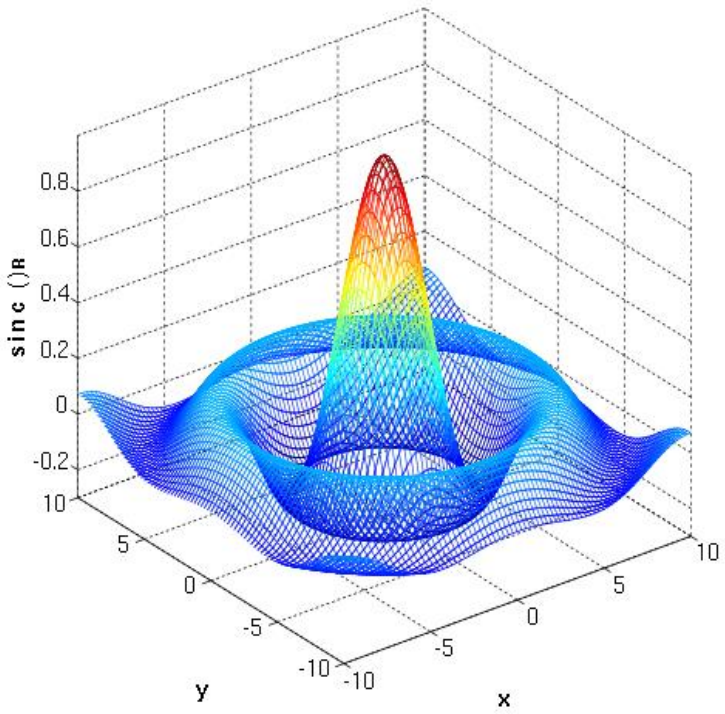


Figure.1 Graph Plotted by MATLAB

A MATLAB program can produce three-dimensional graphics using the functions *surf*, *plot3* or *mesh*.

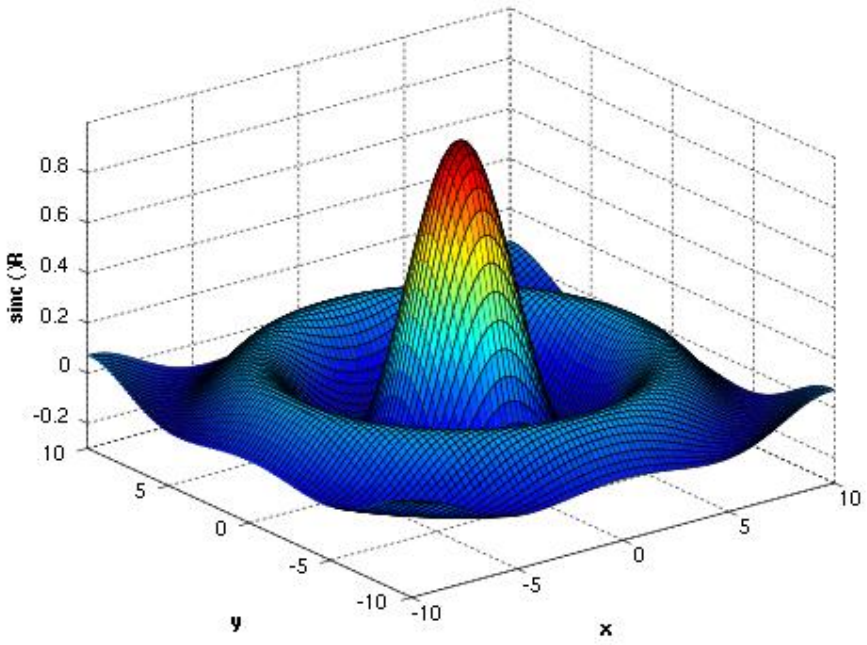
```
[X,Y]=meshgrid(-10:0.25:10,-10:0.25:10);
f=sinc(sqrt((X/pi).^2+(Y/pi).^2));
mesh(X,Y,f);
axis([-1010-1010-0.31])
xlabel('{\bfx}')
ylabel('{\bfy}')
zlabel('{\bfsinc} ({\bfR})')
hiddenoff
```

This code produces a **wireframe** 3D plot of the two-dimensional unnormalized sinc function:



```
[X,Y]=meshgrid(-10:0.25:10,-10:0.25:10);
f=sinc(sqrt((X/pi).^2+(Y/pi).^2));
surf(X,Y,f);
axis([-1010-1010-0.31])
xlabel('{\bfx}')
ylabel('{\bfy}')
zlabel('{\bfsinc} ({\bfR})')
```

This code produces a **surface** 3D plot of the two-dimensional unnormalized sinc function:



In MATLAB, graphical user interfaces can be programmed with the GUI design environment (GUIDE) tool.

Interfacing with other languages

MATLAB can call functions and subroutines written in the programming languages C or Fortran. A wrapper function is created allowing MATLAB data types to be passed and returned. MEX files (MATLAB executables) are the dynamically loadable object files created by compiling such functions. Since 2014 increasing two-way interfacing with Python was being added.

Libraries written in Perl, Java, ActiveX or .NET can be directly called from MATLAB, and many MATLAB libraries (for example XML or SQL support) are implemented as wrappers around Java or ActiveX libraries. Calling MATLAB from Java is more complicated, but can be done with a MATLAB toolbox[33] which is sold separately by MathWorks, or using an undocumented mechanism called JMI (Java-to-MATLAB Interface), (which should not be confused with the unrelated Java Metadata Interface that is also called JMI). Official MATLAB API for Java was added in 2016.[36]

As alternatives to the MuPAD based Symbolic Math Toolbox available from MathWorks, MATLAB can be connected to Maple or Mathematica.Libraries also exist to import and export MathML.

Reference: Wikipedia

Prepared By: Mohammad Obaidullah Khan,

Lecturer in Electrical Engineering, IMSIU, Riyadh, K.S.A.