Computer Lab in Economics
Master in International Economics
Introduction to MATLAB

Inmaculada Álvarez Ayuso

Office 314 (Módulo I) www.uam.es/inmaculada.alvarez

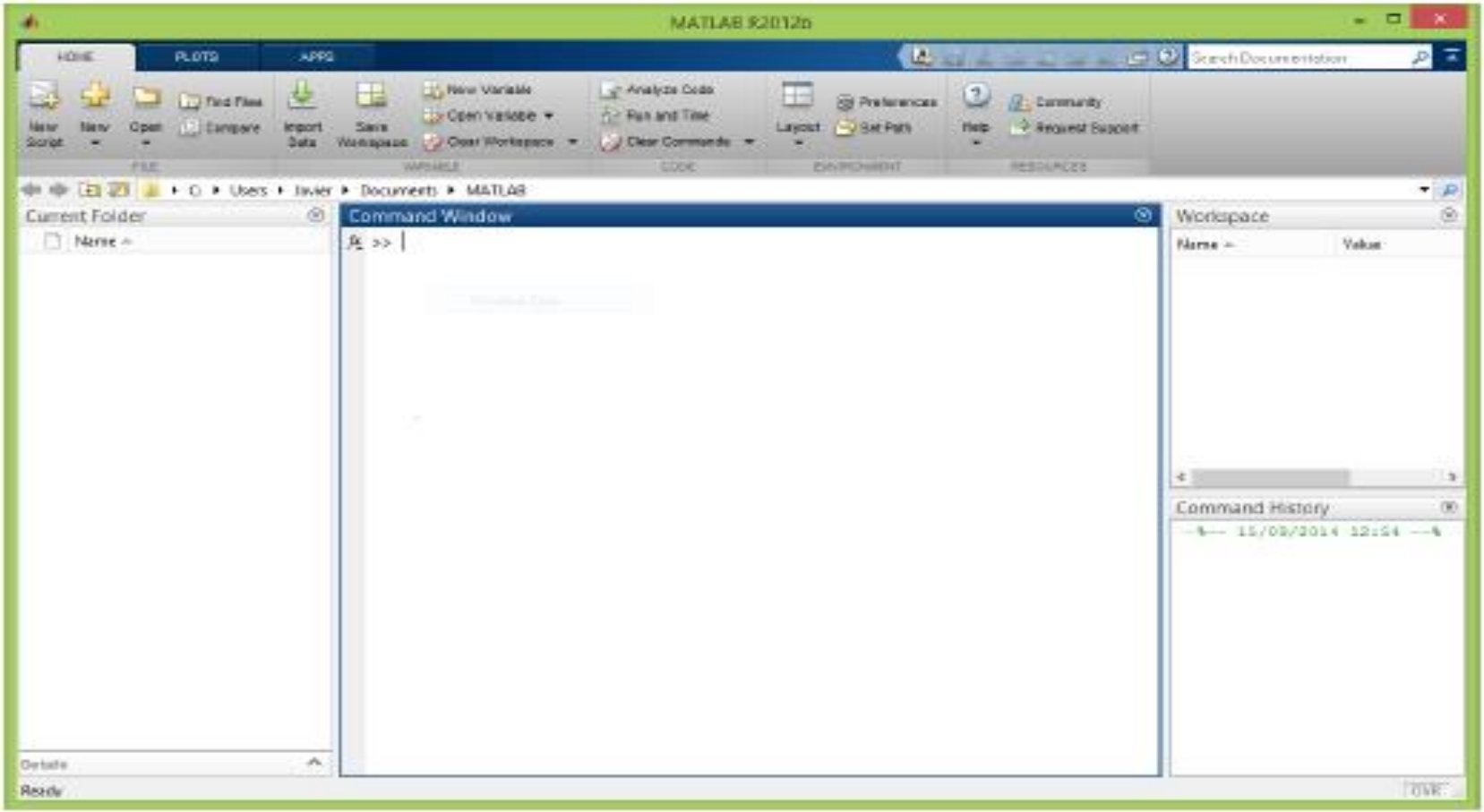
E-mail: inmaculada.alvarez@uam.es

Introduction to the use of MATLAB

- Matrix laboratory (MATLAB) is a software for numerical computing.
- The first version was released in 1984 and it was designed by Cleve Moler.
- The MathWorks, Inc. is the company that develops MATLAB since its initial release (`http://www.mathworks.com/`).
- Nowadays, two versions of MATLAB are realeased every year.
- MATLAB is very used in Engineering and Mathematics, but it is a very powerful tool to use also in Economics.

Introduction to the use of MATLAB

MATLAB R2012b Interface



Introduction to the use of Matlab

MATLAB Interface

- The **Command Window** is used to enter commands into MATLAB and it is where output is displayed.
- The **Workspace** displays all the variables.
- The **Command History** shows the history of all the commands executed in the current and previous sessions.
- The **Current Folder** displays all the files in the current folder.

Introduction to the use of Matlab

Basic operations in MATLAB

- We can perform basic math operations directly in the Command Window:

```
>> 7*5
ans =
35
```

- The result of all operations executed are automatically stored in the **ans** variable.

Introduction to the use of Matlab

Basic operations in MATLAB: assigning variables

- We can store data in a variable using the assignment operator, =:

```
>> x = 6 * 4
x =
24
```

Introduction to the use of Matlab

Basic operations in MATLAB: suppressing auto-printing

- MATLAB automatically prints the output when a statement evaluated.
- To suppress auto-printing, you should end the statement with a semicolon, ;:

```
>> x = 6 * 4;
```

- Now, you can display the content of the variable by writing its name:

```
>> x
x =
24
```

- Or with the **disp** function:

```
>> disp(x)
24
```

Introduction to the use of Matlab

Basic operations in MATLAB: arithmetic operators

- The arithmetic operators in MATLAB are the following:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation

Introduction to the use of Matlab

Basic operations in MATLAB: information about variables

- To display the name of all existing variables use the command **who**:

```
>> who
Your variables are:
ans   x
```

- More information (size, bytes, class, attributes) can be displayed with the **whos** command:

```
>> whos
Name        Size                    Bytes  Class

ans         1x1                         8  double
x           1x1                         8  double
```

Introduction to the use of Matlab

Basic operations in MATLAB: deleting variables

- Variables can be deleted with the clear command:

```
>> clear x
```

- Multiple variables can be deleted at the same time:

```
>> x1 = 1; x2 = 2; x3 = 3;
>> clear x*
```

In this case, all variables whose names begin with *x* will be deleted.

Introduction to the use of Matlab

Basic operations in MATLAB: getting help

- Help of a function can be displayed in the command window with the **help** command:

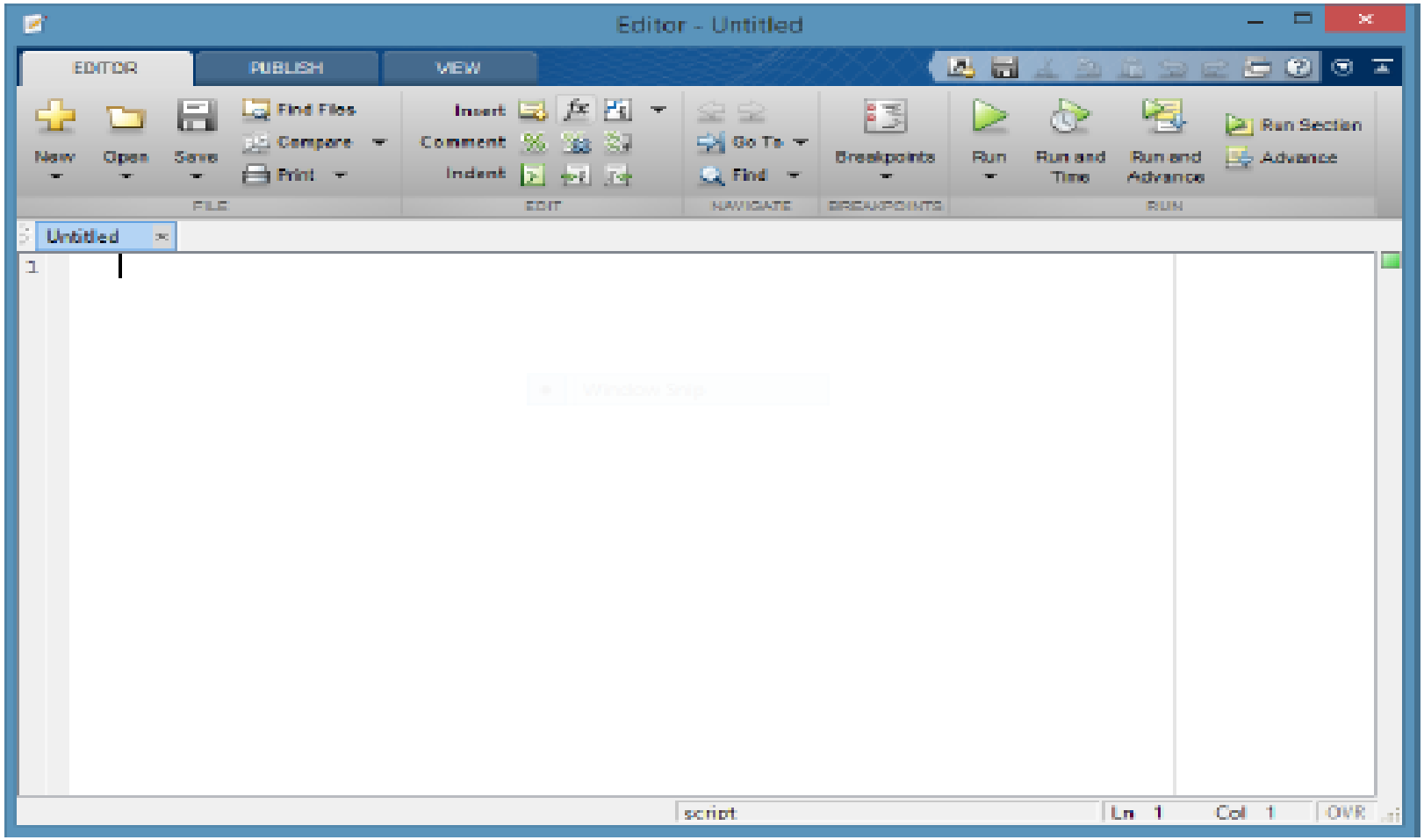
```
>> help clear
clear  Clear variables and functions from
   memory.
clear removes all variables from the workspace
   .
clear VARIABLES does the same thing.
clear GLOBAL removes all global variables.
clear FUNCTIONS removes all compiled MATLAB
   and MEX-functions.
```

- Extensive information can be displayed in the Help browser using the **doc** command:

```
>> doc clear
```

Introduction to the use of Matlab

Basic operations in MATLAB: script editor



Introduction to the use of Matlab

Basic operations in MATLAB: commenting the code

- Comments are text that are not executed (Not interpreted as code). They are used to explain what the code is doing.
- Single line comment

```
% This is a comment
```

- Multi line comment

```
%{
This is a
multi line
comment
%}
```

Introduction to the use of Matlab

Arrays and matrices

- Arrays can be created in MATLAB by writing all their elements inside square brackets:

```
>> x = [1 2 3 4 5]
x =
1     2     3     4     5
```

- Or using the **colon notation** to create a sequence of numbers:

```
>> x = 1:2:13
x =
1     3     5     7     9    11    13
```

Introduction to the use of Matlab

Arrays and matrices: linspace function

- The **linspace** function is used to create an array of *N* elements equally spaced between two numbers (*X*1 and *X*2).

```
linspace(X1, X2, N)
```

```
>> linspace(10,30,5)
ans =
10     15     20     25     30
```

Introduction to the use of Matlab

Arrays and matrices: array indexing

- Elements of an array can be accessed by subscripts:

```
>> x(4)
ans =
7
```

- Several elements can be retrieved at the same time using the colon operator:

```
>> x(3:5)
ans =
5     7     9
```

Introduction to the use of Matlab

Arrays and matrices: array indexing

- If we want to retrieve all the elements from one position to the end we can use the end keyboard:

```
>> x(4:end)
ans =
7     9     11     13
```

- Array indexing can also be used to modify elements of an array

```
>> x(5) = 0
x =
1     3     5     7     0     11     13
```

Introduction to the use of Matlab

Arrays and matrices: arrays and column vectors

- By default, all arrays in MATLAB are row vectors.
- We can create a column vector by writing all the elements of the vectors separated by semicolons:

```
>> x = [1; 2; 3; 4]
x =
1
2
3
4
```

- Alternatively, we can create a column vector by transposing a row vector:

```
>> x = ([1, 2, 3, 4])';
```

Introduction to the use of Matlab

Arrays and matrices: element-by element array multiplication and division

- Element-by-element array multiplication is performed using the `.*` operator:

```
>> x = 1:3; y = 4:6;
>> x .* y
ans =
4     10     18
```

- Element-by-element array division is performed using the `./` operator:

```
>> x ./ y
ans =
0.2500     0.4000     0.5000
```

Introduction to the use of Matlab

Arrays and matrices: element-by element array exponentiation

- Element-by-element array exponentiation is performed with the .^ operator:

```
>> x .^ 2
ans =
1     4     9

>> x .^ y
ans =
1    32   729
```

Introduction to the use of Matlab

Arrays and matrices: matrices

- Matrices are arrays with two dimensions (rows and columns).
- They are created concatenating arrays:

```
>> X = [1 2 3; 4 5 6]
X =
1       2       3
4       5       6
```

- Matrix multiplication, "division" and exponentiation are performed with the *, / and ^ operators, whereas **element-by-element** operations are performed with the corresponding .*, ./, and .^.

Introduction to the use of Matlab

Arrays and matrices: horizontal and vertical concatenation

- **Horizontal concatenation** is performed with:

```
>> x = 1:3; y = 4:6;
>> [x, y]
ans =
1       2       3       4       5       6
```

- **Vertical concatenation** is performed with:

```
>> [x; y]
ans =
1       2       3
4       5       6
```

Introduction to the use of Matlab

Arrays and matrices: standard matrices

- Matrix of ones:

```
>> ones(3)
ans =
1     1     1
1     1     1
1     1     1
```

- Matrix of zeros:

```
>> zeros(3)
ans =
0     0     0
0     0     0
0     0     0
```

Introduction to the use of Matlab

Arrays and matrices: standard matrices

- Identity matrix:

```
>> eye(3)
ans =
1     0     0
0     1     0
0     0     1
```

- Matrix of nan:

```
>> nan(3)
ans =
NaN   NaN   NaN
NaN   NaN   NaN
NaN   NaN   NaN
```

Introduction to the use of Matlab

Arrays and matrices: standard matrices

- If only one argument is specified, the previous commands generate a square matrix of the desired dimension.
- To create a **non-square matrix** or a **vector**, we have to specify two arguments with the two dimensions:

```
>> ones(3,4)
ans =
1     1     1     1
1     1     1     1
1     1     1     1

>> zeros(1,5)
ans =
0     0     0     0     0
```

Introduction to the use of Matlab

Arrays and matrices: size of a matrix

- The size of a matrix can be returned with the **size** command:

```
>> X = eye(4,3);
>> size(X)
ans =
4      3
```

- The function returns an array in which the first element is the number of rows and the second element is the number of columns.
- We can retrieve only the number of rows or the number of columns specifying 1 or 2 respectively as the second argument to the size command:

```
>> size(X,1); % Number of rows
>> size(X,2); % Number of columns
```

Introduction to the use of Matlab

Arrays and matrices: repmat function

- The **repmat** function is used to create and fill a matrix with replications of the desired number or vector.

```
>> repmat(2.5, 3,4)
ans =
2.5000    2.5000    2.5000    2.5000
2.5000    2.5000    2.5000    2.5000
2.5000    2.5000    2.5000    2.5000

>> repmat(1:3,2,2)
ans =
1     2     3     1     2     3
1     2     3     1     2     3
```

Introduction to the use of Matlab

Arrays and matrices: reshape function

- The **reshape** function is used to change the dimensions of a matrix:

```
>> X = rand(2,3)
X =
0.9572    0.8003    0.4218
0.4854    0.1419    0.9157
>> Y = reshape(X,3,2)
Y =
0.9572    0.1419
0.4854    0.4218
0.8003    0.9157
```

- Note that the new matrix must have dimensions that allow to contain all of the original elements.
- You can leave one argument to [] so MATLAB automatically computes the required dimension.

Introduction to the use of Matlab

Arrays and matrices: matrix indexing

- Matrix indexing works the same as vector-array indexing.
- You can specify two indices (row and column).

Indexing	Action
`X(i,j)`	Index the i,j element of the matrix
`X(i,:)`	Index the i-th row and all columns
`X(:,j)`	Index the j-th column and all rows

Introduction to the use of Matlab

Arrays and matrices: matrix arithmetic operations

Operator	Meaning
+	Addition
-	Subtraction
*	Matrix Multiplication
.*	Element-by-element multiplication
./	Element-by-element division
^	Matrix exponentiation
.^	Element-by-element exponentiation

Introduction to the use of Matlab

Arrays and matrices: matrix inverse

- The inverse of a matrix is computed using the **inv** command:

```
>> A = magic(3);
>> inv(A)
ans =
0.1472    -0.1444     0.0639
-0.0611     0.0222     0.1056
-0.0194     0.1889    -0.1028
```

- Computing the inverse of a matrix is a very time-consuming operation and should be avoided.

Introduction to the use of Matlab

Arrays and matrices: matrix inverse

- When you need to compute the inverse to solve a system of linear equations, $Ax = b$, the solution is `x = inv(A)*b`.
- A faster and better way to solve the system is to use the matrix division operator \.

```
>> A = magic(3);
>> b = [1, 2, 3]'
>> x = A\b
x =
0.0500
0.3000
0.0500
```

Introduction to the use of Matlab

Relational and logical operators: relational operators

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

Introduction to the use of Matlab

Relational and logical operators: relational operators (example)

```
>> A = magic(3)
A =
8     1     6
3     5     7
4     9     2
>> A > 5
ans =
1     0     1
0     0     1
0     1     0
>> A <= 4
ans =
0     1     0
1     0     0
1     0     1
```

Introduction to the use of Matlab

Relational and logical operators: logical operators

Operator	Meaning
&	Element-wise AND
&&	Short-Circuit AND
|	Element-wise OR
||	Short-Circuit OR
~	NOT

Introduction to the use of Matlab

Relational and logical operators: logical operators (example)

```
>> A = magic(3)
A =
8     1     6
3     5     7
4     9     2
>> A > 6 | A < 4
ans =
1     1     0
1     0     1
0     1     1
>> ~(A > 6 | A < 4)
ans =
0     0     1
0     1     0
1     0     0
```

Introduction to the use of Matlab

Relational and logical functions

- The &, | and ~ can be performed with the `and(x,y)`, `or(x,y)` and `not(x)` functions.
- The Exclusive OR operation can be performed with the **xor** function.
- The function **any** returns True if any element in a logical vector is True.
- The function **all** returns True if all elemetns in a logical vector are True.

Introduction to the use of Matlab

Control flow: If - Else

- An If construction allows to execute a group of commands only if a certain condition is True.

```
if condition
    % Commands if True
end
```

- An Else clause can be added to execute a group of commands if the condition is Not True.

```
if condition
    % Commands if True
else
    % Commands if False
end
```

Introduction to the use of Matlab

Control flow: If – Else (Example)

```
x = 5;
if (x > 3)
    disp('x is greater than 3')
else
    disp('x is lower than 3')
end
```

Introduction to the use of Matlab

Control flow: For loops

- For Loops are used to repeat a group of commands a fixed number if tiemes.

```
for i = x
        % Commands
end
```

- The code inside the For Loop will be executed once for every column in the array `x`. For every time it is executed, the variable `i` will contain a different column, in order, of `x`.

Introduction to the use of Matlab

Control flow: For loops (example)

```
% Array for iteration
x = 1:5;
% Empty array to store results
y = nan(size(x));
% For Loops
for i=x
    % Commands
    y(i) = i.^2 + sin(i);
end
% Display results
disp(y)
```

Introduction to the use of Matlab

Control flow: For loops (example 2)

```
x = 1:2:10;
y = nan(size(x));
% Position counter
pos = 1;
% For Loops
for i=x
    % Commands
    y(pos) = i.^2 + sin(i);
    % Increase position counter
    pos = pos + 1;
end
% Display results
disp(y)
```

Introduction to the use of Matlab

Control flow: while loops

- A While Loops repeats a group of commands while a condition is True.

```
while condition
    % Commands
end
```

Introduction to the use of Matlab

Control flow: while loops (example)

```
% Set a to 0
a = 0;
% While Loop
while a < 5
    a = a + 1;
end
% Display the value of a
disp(a)
```

Introduction to the use of Matlab

Generating random numbers

- Random numbers can be generated using several **rand*** functions.

Function	Random numbers generated are
rand	Uniformly distributed
randn	Normally distributed
randi	Uniformly distributed integers

Introduction to the use of Matlab

Generating random numbers (example)

```
% Generates a 2x2 matrix of uniformly distributed
a = rand(2)

% Generates a 1x3 matrix of normally distributed
b = randn(1,3)

% Generates a 5x5 vector of uniformly distributed
% integers between 0 and 10
c = randi(10,5)

% Generates a 6x1 vector of uniformly distributed
% integers between 12 and 15
d = randi([12, 15], 6, 1)
```

Introduction to the use of Matlab

Data analysis

- **Basic data analysis** can be performed in MATLAB.
- All data analysis functions assumes, **by default**, that **each column of the data matrix is a variable** and each row is an observation.
- However, this behavior can be changed specifying the dimension parameters of the functions.
- Extended data and statistical analysis can be performed with the **Statistical Toolbox**.
- **Here** we present functions that come with the **basic MATLAB** only.

Introduction to the use of Matlab

Data analysis: functions

Function	Computes
mean	Mean (Average value)
median	Median
min	Minimum value
max	Maximum value
mode	Mode (Most repeated value)
var	Variance
std	Standard deviation
cov	Covariance matrix
corrcoef	Correlation matrix

Introduction to the use of Matlab

Data analysis: example

```
% Generate random numbers
X = randn(10,5);

% Mean of the variables
mean(X)

% Mean by rows
mean(X, 2)

% Maximun value
max(X)

% Correlation matrix
corrcoef(X)
```

Introduction to the use of Matlab

Plotting

- Basic plotting is performed with the **plot** function.

```
x = 0:0.1:10; y = sin(x);
plot(x,y)
```

- Color, marker and linestyle can be modified.
- We can plot several data arrays in the same plot.
- Title, axis label and legend can be set to make the plot better.

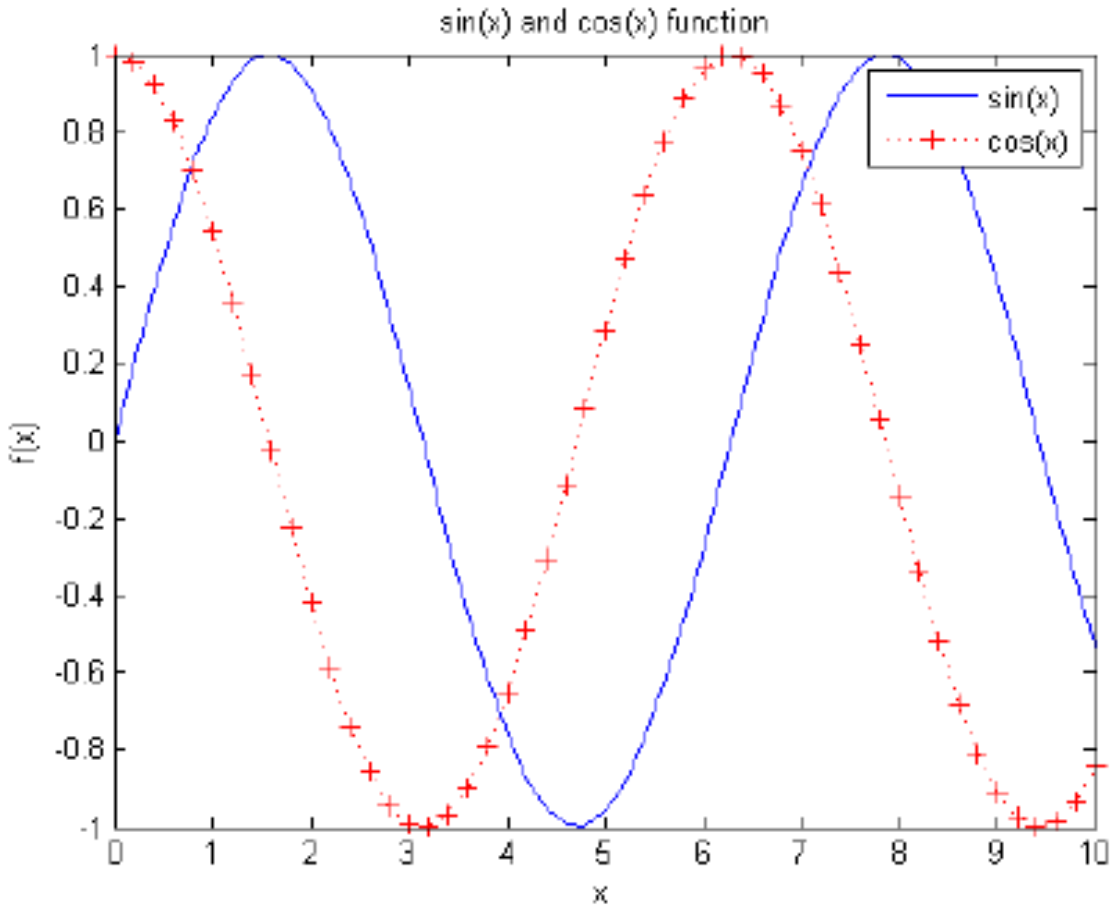
Introduction to the use of Matlab

Plotting: example

```
% Set the data
x = 0:0.2:10; y = sin(x); z = cos(x);
% Plot y=sin(x)
plot(x,y)
hold on % We are going to add more data
% Plot z=cos(x) in red, dotted and with + marker
plot(x,z,'r:+')
% Set title, axis labels and legend
title('sin(x) and cos(x) function')
xlabel('x')
ylabel('f(x)')
legend('sin(x)','cos(x)')
hold off % No more data in this plot
```

Introduction to the use of Matlab

Plotting: example



Introduction to the use of Matlab

Plotting: color

Symbol	Color
y	Yellow
m	Magenta
c	Cyan
r	Red
g	Green
b	Blue
w	White
k	Black

Introduction to the use of Matlab

Plotting: linestyle

Symbol	Linestyle
-	Solid line
--	Dashed line
:	Dotted line
-.	Dash-dot line

Introduction to the use of Matlab

Plotting: marker

Symbol	Marker
+	Plus
o	Circle
*	Asterisk
.	Point
x	Cross
s	Square
d	Diamond
^	Up triangle
v	Down triangle
>	Right triangle
<	Left triangle
p	Pentagram
h	Hexagram

Introduction to the use of Matlab

Plotting: advanced plotting

- The **plot** command is very versatile and it is used as the base to build advanced custom plots.
- Advanced plotting features include:
  - Common plots (bar, area, pie, histogram, scatter).
  - Subplots (Multiple plots with different axis in the same figure).
  - 3D plots.
  - Animations (Videos).
  - Exporting to different file formats.
- New graphics system (HG2, Handle Graphics version 2) in MATLAB R2014b.

Introduction to the use of Matlab

References

Hanselman, D. and Littlefield, B. (2012).
Mastering MATLAB.
Prentice Hall.

The MathWorks, Inc. (2012).
MATLAB — The Language of Technical Computing, Version R2012b (8.0).
Natick, Massachusetts.