

Introduction to MATLAB for Economics

Introduction to MATLAB

Javier Barbero

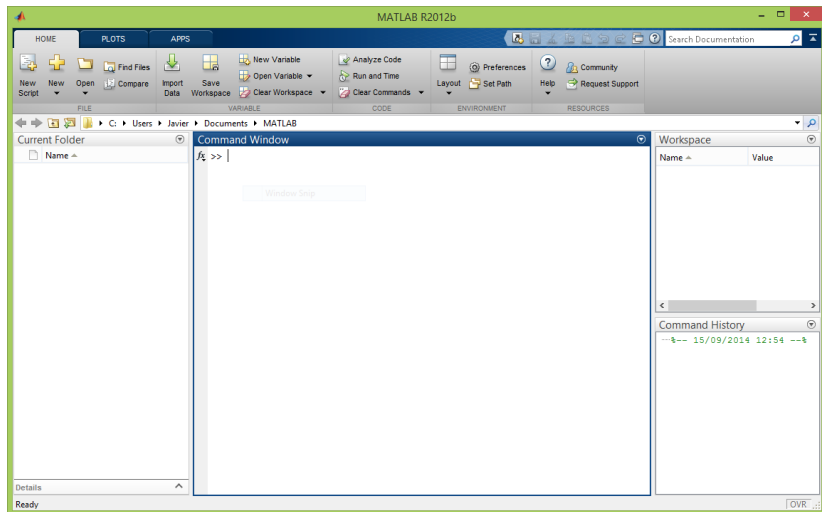
Universidad Autónoma de Madrid
<http://www.javierbarbero.net>

October 24, 2014

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 released every year.
- MATLAB is very used in Engineering and Mathematics, but it is a very powerful tool to use also in Economics.

MATLAB R2012b Interface



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.

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.

Assigning variables

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

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

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
```

Arithmetic Operators

- The arithmetic operators in MATLAB are the following:

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

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
```

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.

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.
```

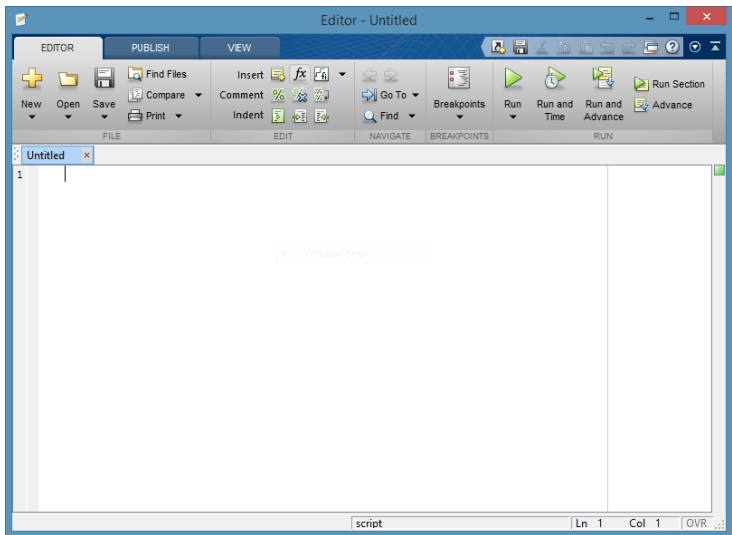
```
clear GLOBAL removes all global variables.
```

```
clear FUNCTIONS removes all compiled MATLAB  
and MEX-functions.
```

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

```
>> doc clear
```

The MATLAB Script Editor



Commentin 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  
%}
```

Arrays

- 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
```

The linspace function

- The **linspace** function is used to create an array of N elements equally spaced between two numbers ($X1$ and $X2$).

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

```
>> linspace(10,30,5)
```

```
ans =
```

```
10      15      20      25      30
```

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
```


Array Indexing Cont.

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

```
>> 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
```

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])';
```

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
```

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
```

Matrices

- Matrices are arrays with two dimensions (rows and columns).
- They are create 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 `.^`.

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
```

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
```

Standard matrices (Cont.)

- **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
```


Standard matrices (Non square)

- 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
```

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
```

The 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
```

The 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.

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

Matrix Arithmetic Operations

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

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.

Matrix Inverse Cont.

- When you need to compute the inverse to solve a system of linear equations, $Ax = b$, the solution is $x = \text{inv}(A)*b$.
- 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
```


Relational Operators

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

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
```

Logical Operators

Operator	Meaning
<code>&</code>	Element-wise AND
<code>&&</code>	Short-Circuit AND
<code> </code>	Element-wise OR
<code> </code>	Short-Circuit OR
<code>~</code>	NOT

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
```

Relational and Logical Functions

- The $\&$, $|$ and \sim 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 elements in a logical vector are True.

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
```

If-Else. Example

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

For Loops

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

```
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 .

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)
```

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)
```

While Loops

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

```
while condition
    % Commands
end
```

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)
```

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

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)
```

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.

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

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)
```

Plotting

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

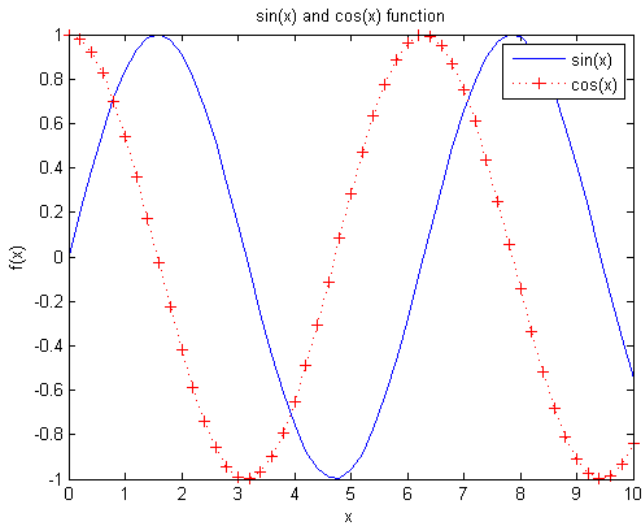
```
x = 0:0.1:10; y = sin(x);  
plot(x,y)
```

- 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.

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
```

Plotting. Example



Plotting. Color

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

Plotting. Linestyle

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

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

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.

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.