Introduction to MATLAB for Economics

Javier Barbero

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

October 24, 2014
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:

  ```matlab
  >> 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, =:

\[
\begin{align*}
\text{>> } & \quad x = 6 \times 4 \\
\text{x } & = \\
24
\end{align*}
\]
Supressing auto-printing

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

  \[
  >> x = 6 \times 4;
  \]

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

  \[
  >> x
  \]

  \[
  x = 24
  \]

- Or with the `disp` function:

  \[
  >> 
  \]

  \[
  disp(x)
  \]

  \[
  24
  \]
The arithmetic operators in MATLAB are the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
</tbody>
</table>
Information about variables

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

  ```matlab
  >> who
  Your variables are:
an   x
  ```

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

  ```matlab
  >> whos
  Name      Size       Bytes     Class
  ans       1x1         8 double
  x         1x1         8 double
  ```
Deliting variables

- Variables can be deleted with the `clear` command:
  
  ```matlab
  >> clear x
  ```

- Multiple variables can be deleted at the same time:
  
  ```matlab
  >> 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:

  ```matlab
  >> 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:

  ```matlab
  >> doc clear
  ```
The MATLAB Script Editor
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  
  %}
Arrays can be created in MATLAB by writing all their elements inside square brackets:

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5
\end{bmatrix}
\]

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

\[
x = \\
1 & 2 & 3 & 4 & 5
\]

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

\[
1:2:13
\]

\[
>> x = 1:2:13
\]

\[
x = \\
1 & 3 & 5 & 7 & 9 & 11 & 13
\]
The **linspace** function is used to create an array of \( N \) elements equally spaced between two numbers (\( X_1 \) and \( X_2 \)).

\[
\text{linspace}(X_1, X_2, N)
\]

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

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

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

```matlab
>> 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:

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

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

  ```matlab
  >> x = ([1, 2, 3, 4])';
  ```
Element-by-element array multiplication and division

- Element-by-element array multiplication is performed using the \( \.* \) operator:

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

- Element-by-element array division is performed using the \( \./ \) operator:

```matlab
>> x ./ y
ans =
 0.2500  0.4000  0.5000
```
Element-by-element array exponentiation

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

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

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

- Matrices are arrays with two dimensions (rows and columns).
- They are create concatenating arrays:
  
  \[
  \begin{bmatrix}
  1 & 2 & 3 \\
  4 & 5 & 6
  \end{bmatrix}
  \]

  \[
  X =
  \begin{bmatrix}
  1 & 2 & 3 \\
  4 & 5 & 6
  \end{bmatrix}
  \]

- 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:

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

- **Vertical concatenation** is performed with:

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

- **Matrix of ones:**
  
  ```matlab
  >> ones(3)
  ans =
  1  1  1
  1  1  1
  1  1  1
  ```

- **Matrix of zeros:**
  
  ```matlab
  >> 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
```
The size of a matrix can be returned with the `size` command:

```matlab
>> 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:

```matlab
>> size(X,1); % Number of rows
>> size(X,2); % Number of columns
```
The `repmat` function is used to create and fill a matrix with replications of the desired number or vector.

```matlab
>> repmat(2.5, 3, 4)
an =
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)
an =
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:

\[
\begin{align*}
\text{>> } & \quad X = \text{rand}(2,3) \\
& \quad X = \\
& \quad \begin{bmatrix}
0.9572 & 0.8003 & 0.4218 \\
0.4854 & 0.1419 & 0.9157
\end{bmatrix} \\
\text{>> } & \quad Y = \text{reshape}(X,3,2) \\
& \quad Y = \\
& \quad \begin{bmatrix}
0.9572 & 0.1419 \\
0.4854 & 0.4218 \\
0.8003 & 0.9157
\end{bmatrix}
\end{align*}
\]

- 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).

<table>
<thead>
<tr>
<th>Indexing</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>X(i,j)</td>
<td>Index the $i,j$ element of the matrix</td>
</tr>
<tr>
<td>X(i,:)</td>
<td>Index the $i$-th row and all columns</td>
</tr>
<tr>
<td>X(:,j)</td>
<td>Index the $j$-th column and all rows</td>
</tr>
</tbody>
</table>
# Matrix Arithmetic Operations

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Matrix Multiplication</td>
</tr>
<tr>
<td>.*</td>
<td>Element-by-element multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-by-element division</td>
</tr>
<tr>
<td>^</td>
<td>Matrix exponentiation</td>
</tr>
<tr>
<td>.^</td>
<td>Element-by-element exponentiation</td>
</tr>
</tbody>
</table>
Matrix Inverse

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

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

A faster and better way to solve the system is to use the matrix division operator \.

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>~=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Element-wise AND</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Short-Circuit AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>NOT</td>
</tr>
</tbody>
</table>
Logical Operators Example

```matlab
>> 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
```
The &, | and \sim can be performed with the \texttt{and(x,y)}, \texttt{or(x,y)} and \texttt{not(x)} functions.

The Exclusive OR operation can be performed with the \texttt{xor} function.

The function \texttt{any} returns True if any element in a logical vector is True.

The function \texttt{all} returns True if all elements in a logical vector are True.
An If construction allows to execute a group of commands only if a certain condition is True.

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

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

```matlab
if condition
    % Commands if True
else
    % Commands if False
end
```
If-Else. Example

```matlab
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.

```matlab
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 \( x \).
% 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

\[
x = 1:2:10;
y = \text{nan}(\text{size}(x));
\%
\text{Position counter}
pos = 1;
\%
\text{For Loops}
\textbf{for } i=x
\%
\text{Commands}
y(pos) = i.^2 + \sin(i);
\%
\text{Increase position counter}
pos = pos + 1;
\textbf{end}
\%
\text{Display results}
disp(y)
\]
A While Loops repeats a group of commands while a condition is True.

```matlab
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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Random numbers generated are</th>
</tr>
</thead>
<tbody>
<tr>
<td>rand</td>
<td>Uniformly distributed</td>
</tr>
<tr>
<td>randn</td>
<td>Normally distributed</td>
</tr>
<tr>
<td>randi</td>
<td>Uniformly distributed integers</td>
</tr>
</tbody>
</table>
Generating random numbers. Example

% Generates a 2x2 matrix of uniformly distributed
da = 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 assume, 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Computes</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>Mean (Average value)</td>
</tr>
<tr>
<td>median</td>
<td>Median</td>
</tr>
<tr>
<td>min</td>
<td>Minimum value</td>
</tr>
<tr>
<td>max</td>
<td>Maximum value</td>
</tr>
<tr>
<td>mode</td>
<td>Mode (Most repeated value)</td>
</tr>
<tr>
<td>var</td>
<td>Variance</td>
</tr>
<tr>
<td>std</td>
<td>Standard deviation</td>
</tr>
<tr>
<td>cov</td>
<td>Covariance matrix</td>
</tr>
<tr>
<td>corrcoef</td>
<td>Correlation matrix</td>
</tr>
</tbody>
</table>
Data Analysis. Example

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

% Mean of the variables
mean(X)

% Mean by rows
mean(X, 2)

% Maximum value
max(X)

% Correlation matrix
corrcoef(X)
Basic plotting is performed with the `plot` function.

```matlab
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.
% 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

\[ \sin(x) \text{ and } \cos(x) \text{ function} \]

\[ f(x) = \sin(x) \]

\[ f(x) = \cos(x) \]

\[ x \text{ values range from 0 to 10} \]
# Plotting. Color

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Yellow</td>
</tr>
<tr>
<td>m</td>
<td>Magenta</td>
</tr>
<tr>
<td>c</td>
<td>Cyan</td>
</tr>
<tr>
<td>r</td>
<td>Red</td>
</tr>
<tr>
<td>g</td>
<td>Green</td>
</tr>
<tr>
<td>b</td>
<td>Blue</td>
</tr>
<tr>
<td>w</td>
<td>White</td>
</tr>
<tr>
<td>k</td>
<td>Black</td>
</tr>
</tbody>
</table>
## Plotting. Linestyle

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Linestyle</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Solid line</td>
</tr>
<tr>
<td>–</td>
<td>Dashed line</td>
</tr>
<tr>
<td>:</td>
<td>Dotted line</td>
</tr>
<tr>
<td>-.</td>
<td>Dash-dot line</td>
</tr>
</tbody>
</table>
**Plotting. Marker**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Plus</td>
</tr>
<tr>
<td>o</td>
<td>Circle</td>
</tr>
<tr>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td>.</td>
<td>Point</td>
</tr>
<tr>
<td>x</td>
<td>Cross</td>
</tr>
<tr>
<td>s</td>
<td>Square</td>
</tr>
<tr>
<td>d</td>
<td>Diamond</td>
</tr>
<tr>
<td>^</td>
<td>Up triangle</td>
</tr>
<tr>
<td>v</td>
<td>Down triangle</td>
</tr>
<tr>
<td>&gt;</td>
<td>Right triangle</td>
</tr>
<tr>
<td>&lt;</td>
<td>Left triangle</td>
</tr>
<tr>
<td>p</td>
<td>Pentagram</td>
</tr>
<tr>
<td>h</td>
<td>Hexagram</td>
</tr>
</tbody>
</table>
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
