Introduction to `MatLab`.

Florimond Guéniat

August 15, 2018

Chapter 1

Quick tour MatLab

I Legal stuff

MatLab is a registered trademark of MathWorks, Inc.

II What is the use of MatLab?

MatLab (for MATrix LABoratory) aims at delivering quickly some *relatively* inexpensive computations. It shines, as expected from the name, when it involves linear algebra, i.e., operations on matrices.

The main advantage of MatLab compared to language like C/C++ or Fortran are:

- No compilation
- The prompt
- Simplicity
- portability
- Built-in functions:
  - integration
  - visualization
  - tool-box

The negative points are mostly:

- sub performance
- not open source
- price

III Equivalent of MatLab

Octave and SciLab are almost identical to MatLab. A MatLab script would work on these two others open-source and free softwares.

Most of the tips can also be applied to python, especially when the packages scipy and numpy (for scientific and engineering computations) are used.

IV Philosophical idea of these teachings

The objective of these tutorial are to illustrate the ENG 4XXX courses as well at to help you to learn quickly how to numerically solve problems. It will hence give you the first concepts behind `MatLab`, coding and problem solving. The emphasis here is learning by doing. Therefore, try not to read these documents without a computer close-by.

V Hello World

V a) `MatLab` as a software

V a) i Start `MatLab`

Click on the icon, duh!

V a) ii Organization of the window

You can find a few important sections:

- the command windows
  This is the prompt
- current folder
  It lists the files
- the Workspace
  It gives details on the objects present in memory
- the editor
  this is where you can write a script
- the ribbon
  It gives access to properties, functions, etc. Similar in spirit to Words and Excel.

[[TODO ** image opening windows of matlab **]]

V b) How to print "hello world"

Click on the Command Window, and type "hello world". You will see:

```
>> 'Hello World'
ans =
Hello World
```

`MatLab` printed the 'Hello World', congratulations !
You can see also that the 'Hello World has been assignated to a variable named ans.

Definition 1

ans: `ans` is short for answer.
The results of the command is always stored in the variable ans, except if it is assigned to a given variable. Consequently, the command `1+1` will affect the variable `ans`, but `x=1+1` will not, and 2 will be assign to `x`.
`ans` can be re-used in the prompt: `x = ans+1`. However, a good practice is to assign the result to a user-defined variable.

Now, try without the quotes, and you will see:

```
>> Hello World
Undefined function or variable 'Hello'.
```

plus some help.

Hello World is understood by `MatLab` as a function/variable and then an option for this function. `MatLab` hence thinks that Hello is something that already exists, and it is not the case here. An error follow.

The main reason behind that is that we want to print a string.

Definition 2

String: A string is a chain of characters. It is *not* a number.
It has to be between quotes: 'some text' or double quotes: "some text". It should not be mixed with numbers.

Try now to add a semi colon at the end of the line:

```
>> 'Hello World';
>>
```

Nothing is printed in the prompt.

Definition 3

Prompt: The prompt is the >> sign.
Once enter is hit, `MatLab` will interpret the line, and send back any results.

One of the most important tip to remember: `MatLab` will always print the result of a line if it does not have a ";" at the end of the line.

ProTip 1

Do not forget the ";" at the end of lines !
It is not a big deal when dealing with small matrices and small vectors. But when an image is being manipulated, it means that MatLab is manipulated a matrix with dimension around 1000×1000. Forgetting the ";" sign means that MatLab will show around a *million* numbers every line of your script!

VI MatLab as a calculator

VI a) Algebra

You can use MatLab as a calculator. Click on the prompt:

```
>> 4+3
ans =
7
>> 4*3
ans =
12
```

As expected, MatLab respects the BODMAS (Brackets, Order, Division/Multiplication, Addition/Subtraction). Try a few operations !

```
>> (4+3)*2
ans =
14
>> 4+3*2
ans =
10
```

VI b) Variables

ans can be used to store a result, but it will be overwritten every time a command is executed:

```
>> 2+2
ans =
4
>> ans+2
ans =
6
>> ans+2
ans =
8
```

VI c) Creation and re-assignement

Variable can be easily created and assigned with the sign "=".

```
>> x = 2+2
x =
4
```

```
>> x+2
ans =
6
>> x*5
ans =
10
```

Definition 4

variable: A variable is essentially a name that is associated with a value. Values can be of several types:

- results, such as string, numbers or matrices: `x = 3.`
- functions, for instance `sin` is a built-in function
- complex objects, for instance, a plot

They are usually assigned with the sign "="

VI c) i Naming convention

A variable name can be anything, such as `goodnameforavariable` or `GoodNameForAVariable`, or `good_name_for_a_variable` . However:

- it cannot start with _
- it cannot start with a number
- a few names are protected

ProTip 2

Try to use clever name for variables, it will help to understand the code.
If all the results are named `result_1`,`result_2`,`result_3`, it is hard to know what they should contain.
On the contrary, if you have the variable `name_city`, you expect it to be a string and having a proper name.
In a similar way, if you have the variable `motor_freq`, you expect it to be a number.
Also, if you have the variable `price_pond`, the variable `price_dollar` and the variable `rate_dollar2pound`, the line `price_pound = rate_dollar2pond * price_dollar` is pretty explicit. If the variables where instead named `x`,`y`,`z`, then the line `x = y*z` is much more cryptic.

The choice of a name is important, for you a, `x` is good for an unknown, `s` if you expect its value to be a string, `v` if it is a vector... More complex names can be used, such as `x_problem_1`. Try to be consistent thorough the piece of code !

A few tips:

- Use different names for different results
- Use a name that is meaningfull (e.g. `str_name` if the variable is assigned with a chain of character that is a name)
- Consequently, avoid unecessary use of index (e.g. `result_1`, `result_2` etc.)

ProTip 3

Many naming convention exist.
However, you can use the following name convention:

- `UpperCamelCase` for functions: `MyFunction`
- `CAPITALIZED_WITH_UNDERSCORES` for constants `Pi=3.14`
- `lowercase_separated_by_underscores` for other variables `name_of_univ = 'BCU'`.

VI c) ii Reassignement

Updating a variable is handy: you might want to change the variable `year` from 2017 to 2018.

You can easily update a variable, by reassigning a new value to it. It hence uses the sign "=". For instance:

```
>> x = 2+2
x =
4
>> x = x + 5
x =
9
>> x = 0
x =
0
```

VI c) iii Exercices

1. Create the variables `x,y,z` assigned with 1, 2 and 3.
2. Create the variable `sum_xyz` that is the sum of `x,y` and `z`.
3. Propose a name for a variable that is assigned as a value `'Birmingham'`
4. Propose a name for a variable that is assigned as a value `'BCU'`
5. Try to assign to the variable `year` the value `2017`, and then to `2018`!
6. Try to assign to the variable `girlfriend_name` the value `'Adilah'` (using the sign equal, pun totally intended), and to the variable `ex_girlfriend_name` the value `'Marie'`. Then, reassign to the variable `girlfriend_name` the value `'Kiara'`, and to the variable `ex_girlfriend_name` the value `'Adilah'`.

VI d) Workspace

When you have created a variable, it is available in the *workspace*. It is the area (usually) on the right. It allows to:

- show what variables are currently known to `MatLab`
- know what is present in the memory
- indicate what there is in the variables
- eventually modify the content of a variable

VI e) Entering multiple commands per line

It is possible to enter multiple commands per line. Use commas "," or semicolons ";" for that ; the commas will *not* suppress the outputs.

ProTip 4

Try to avoid multiple commands per line, most of the time, it makes the code harder to read. Nevertheless, it makes sense when assigning a few variables that are related.

```
>> x = 2 ; y = 3 ; z = 4 ;
>> x = 2 ; y = 3 , z = 4 ;
y =
3
>> x = 2 , y = 3 , z = 4 ,
x =
2
y =
3
z =
5
```

VI f) Basic arithmetic

Basic arithmetic operators are pretty classic:

Table 1.1: Arithmetic operators

operation	command	exemple
addition	+	3+4
soustraction	-	3-4
multiplication	*	3*4
division	/	3/4
power	^	2 ^ 4

VI g) functions

VI g) i How to find a function or a command ?

If you look for something, hit the help button. For instance, if you want to look for the sine function:

[[TODO ** image help sine **]]

ProTip 5

Use the help! It is *very* useful and you will mostly find any function/tool/infos that you need.
Usually, the help contains a few examples. Do not hesitate to read them carefully, and to try them. They will help understanding how to use the functions and properties of `MatLab`.
There is also a "See Also" section that can be useful when looking for a particular topic.

VI g) ii Using a function

Calling a function is relatively easy and intuitive. Let's take the sine function as an illustration.

```
>> sin(3.14)
ans =
0.0016
```

You ask `MatLab` to evaluate the function `sin` in $3.14 \approx \pi$. For that, you just put the argument in parenthesis.

ProTip 6

Trigonometric functions in `MatLab` are in radiant. `sin(360)` is hence different from 0 but rather close to 0.96.
In a similar spirit, `log` in `MatLab` is the natural logarithm, and not the $\log$ in base 10.

Typical functions are available with somewhat explicit names, see Tab. 1.2. Similarly, many useful constants for the engineer are implemented in `MatLab`, see Tab. 1.3.

Table 1.2: A few function names in `MatLab`. Many others are already implemented in `MatLab`.

Trigonometry	name	Stats	name	Misc.	name
sine	sin	mean	mean	square root	sqrt
cosine	cos	maximum	max	absolute value	abs
exponential	exp	minimum	min	round up	ceil
natural logarithm	log	standard dev.	std	conjugate	conj

Table 1.3: A few useful constant names in `MatLab`.

$\pi \approx 3.14$	`pi`
$i = \sqrt{-1}$	`i`
$i = \sqrt{-1}$	`j`
∞	`Inf`
Not a Number	`NaN`

VI h)

Chapter 2

Basic Linear Algebra

Linear algebra is the fundations of `MatLab`, and what makes it popular.
In particular, `MatLab` makes the manipulation of matrices and vectors very easy.
This section will show how to do:

- operations involving vectors
- operations involving matrices
- operations involving both arrays and matrices

I Vectors

I a) Creation of a vector

I a) i Line vector

Creating a vector `v` is easy. You just put all the components between brackets.

```
>> v = [1 2 3]
v =
1 2 3
```

Let's create Cartesian vectors in dimension two. $e_x = (1, 0)$ and $e_y = (0, 1)$:

```
>> ex = [1 0]
ex =
1 0
>> ey = [0 ,1]
ey =
0 1
```

ProTip 7

When creating a line vector, commas between components are unnecessary, but will help reading the code !

I a) ii Column vector

Creating a column vector is similar to creating a line vector, except that the element are separated with a semi-colon ";".

```
>> v = [1;0;4]
v =
1
0
4
>> w = [0;1]
w =
0
1
```

I a) iii Transpose operator

It is possible to change a column vector to a line vector, and reciprocaly, by using the transpose operator "'".

```
>> ex = [1;0]
ex =
1
0
>> ex'
ans =
1  0
>> ey = [0,1]
ey =
0  1
>> ey'
ans =
0
1
```

I b) Access to the elements of a vector

I b) i Access to one element

The first element of a vector `v` is `v(1)`. The second element is `v(2)`, and so forth.

Accessing the element of a vector is just calling the vector with specifying the desired element:

```
>> x = [1  3  4  -2.5  8]
x =
1  3  4  -2.5  8
>> x(1)
ans =
1
>> x(3)
ans =
4
>> x(4)
ans =
-2.5
```

The last element of a vector can be called using the argument `end`: You can also call the ith item from the end using `end-i`.

```
>> x = [1  3  4  -2.5  8]
x =
 1.000  3.000  4.000  -2.5  8.000
>> x(end)
ans =
8
>> x(end-1)
ans =
 -2.5
>> x(end-2)
ans =
4
```

I b) ii Access to several elements

The operator ":" gives access to all the elements between the first and the last element (included), in a column vector:

```
>> v = [1  3  4  -2.5  8];
>> v(:)
ans =
1
3
4
 -2.5
8
```

Accessing to all the elements between the second and fifth element of `v` is `v(2:5)`:

```
>> v = [1  3  4  -2.5  8  12];
>> v(2:5)
ans =
3
4
 -2.5
8
```

To access to *all* the elements between the first and the last element (included), in a column vector, simply use the colon operator:

```
>> x = [1  3  4  -2.5  8];
>> x(:)
ans =
1
3
4
 -2.5
8
```

ProTip 8

The colon operator is pivotal in `MatLab`.
It is noticeably useful to

- generate lists: `x= 1:20`
- controlling loops (more in a following chapter): `for i=1:20`
- transform a matrix in a column vector `M(:)`
- extract sub-parts of vectors/matrices `M(2:4,1:2:8)`

More details in Sec. III a) i.

I c) Basic operations on vectors

I c) i Addition/subtraction

It is easy to add or subtract a given value to *all* the components of a vector, using the signs "+" and "-".

```
>> x = [1  3]
x =
1  3
>> x + 4
ans =
5  7
>> y = x - 2
y =
-1  1
>> z = [5  10  -1  8] + 3
z =
8  13  2  11
```

Vectors can be added, as long as their dimensions correspond:

```
>> ex = [1  0]; ey = [0,1] ;
>> ex + ey
ans =
1  1
>> ex - ey
ans =
1  -1
```

Of course, if their dimensions do not correspond, `MatLab` will send back and error:

```
>> x =[1,0] ; y = [1,0,0];
>> x+y
Matrix dimensions must agree.
```

ProTip 9

MatLab considers vectors as 1D matrices.

I c) ii Multiplication

It is easy to multiply or divide by a given value to *all* the components of a vector, using the signs "*" and "/"

```
>> x = [1  3]
x =
 1  3
>> x * 4
ans =
 4  12
>> y = x / 2.
y =
 0.5  1.5
>> z = 3 * [5  10  -1  8]
z =
 15  30  -3  24
```

What is multiplication for vectors ?
Two definitions can be proposed.

I c) ii 1 dot product The first definition is the dot product between two vectors.

Definition 5

dot product: The dot product (or inner product) of two vectors is the sum of the multiplication of their components. If $u = (u_i), v = (v_i)$ are n-dimensional vectors, $< u, v >= \sum_{i=1}^{n} u_i \times v_i$.

It can be done using the function `dot`.

```
>> x =[1,0] ; y = [1,0];
>> dot(x,y)
ans =
 1
>> x(1) * y(1) + x(2) * y(2)
ans =
 1
>> a =[0,0.5,2] ; b = [2,0,4];
>> dot(a,b)
ans =
 8
>> a(1) * b(1) + a(2) * b(2) + a(3) * b(3)
and =
 8
```

I c) ii 2 Element-wise multiplication Another definition could be the element wise multiplication. It means that each element of a vector is multiply by the corresponding element of the other vector. It is similar to the dot product *except* for the sum. The operator for that is ".*" (it is read "dot product", which is pretty stupid when you think about it!).

```
>> x =[1,0] ; y = [1,0];
>> x.*y
ans =
1 0
>> [x(1) * y(1), x(2) * y(2)]
ans =
1 0
>> a =[0,0.5,2] ; b = [2,0,4];
>> a.*b
ans =
0 0 8
>> [a(1) * b(1), a(2) * b(2), a(3) * b(3)]
and =
0 0 8
```

ProTip 10

In MatLab, using "." in front of an operator means that this operator will be applied element-wise (to each element of the vector/matrix).

[[TODO exercices]]

II Matrices

MatLab sees vectors a line matrices. Building a matrix is the equivalent of stacking lines. For that, MatLab uses the semi colon sign ";". The following lines are equivalent:

```
>> M =[1,0 ; 0,1];
M =
1 0
0 1
>> m =[1,0 ; 0 1];
M =
1 0
0 1
>> m =[ [1,0] ; [0 1]];
M =
1 0
0 1
```

The last line shows that a matrix is virtually stacked vectors.

[[TODO exercises]]

Try to create the vector $x = (1, 2, 3, 4)$.

III Useful built-in functions for vectors and matrices

III a) Generating a vector

III a) i The colon operator

One of the most powerful operator in `MatLab` is the colon operator ":". It allows in particular to generate lists.

We have seen already that `v(2:5)` gives the elements of `v` between the 2nd and 5th position. What it does is actually ask `MatLab` to send back elements of `v` in position $2, 3, 4, 5$. These positions are *generated* by the command `2:5`.

```
>> 2:5
ans =
2  3  4  5
```

The command `a:b` hence generates a vector, starting in a, each element being incremented by 1, until it would be greater than b.

```
>> 4:8
ans =
4  5  6  7  8
>> 4.5:8
ans =
4.5  5.5  6.5  7.5
```

In the second example, the last item is 7.5. 8.5 would be larger than 8 and is hence omitted.

It is possible to force the increment. The command `a:da:b` generates a vector, starting in a, each element being incremented by da, until it would be greater than b.

```
>> 0:.1:1
ans =
0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.
```

III a) ii Other functions

Table 2.1: A few functions that generate vectors.

name	description	illustration
colon operator :	see Sec III a) i	0:10:.5
`linspace(a,b,n)`	linearly spaced n-dimensional vector between a and b	`linspace(0,1,11)`
`diag(A)`	diagonal of matrix (A) between a and b	`diag([[1,2];[3,4]])`

III b) Generating a matrix

IV What to remember

IV a) Creating a matrix

Follow the procedure:

- start with a bracket [

Table 2.2: A few functions that generate vectors.

name	description	illustration
`zeros(m,n)`	m, n-dimensional matrix filled with 0	`zeros(2,4)`
`ones(m,n)`	m, n-dimensional matrix filled with 1	`ones(2,4)`
`rand(m,n)`	m, n-dimensional matrix filled with random numbers taken between 0 and 1	`rand(2,4)`
`eye(n)`	n-dimensional identify matrix	`eye(10)`
`diag(v)`	matrix filled with 0 with `v` as diagonal	`diag([1,2,3])`

- write each element of a column, separated with space or commas
- separate rows with a semi-colon
- end with the bracket]

IV b) Manipulating matrices

You can

- add, subtract, multiply matrices if their sizes are compatible
- access to the ith element of matrix `M` using `M(i)`
- access to the (ith,jth) element of matrix `M` using `M(i,j)`

Bibliography