Introduction to `MatLab`.

Florimond Guéniat

August 16, 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 command 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

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**
>
> 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". For example, 'Lorem ipsum dolor sit amet' is a chain or characters.
> But it is not that easy. `s = '45'` is the chain of characters '4' and '5'. But `n = 45` is the number 45. `s` and `n` are different.

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

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

Nothing is printed in the command prompt.

> **Definition**
>
> COMMAND PROMPT: The command prompt is the >> sign.
> Command Prompt is a command line interpreter. It is used to execute entered commands. Once enter is hit, `MatLab` will interpret the command, 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.

> **Pro Tip**
>
> 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

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

Pro Tip

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.)

Pro Tip

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.

> **Pro Tip**
>
> Try to avoid multiple commands per line.
> Most of the time, it makes the code harder to read, especially if there is not a good reason to do so.
> Nevertheless, it can make sense write a few commands per line when assigning a 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 **]]

Pro Tip

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.

Pro Tip

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`

VII Let's solve a real problem

Problem Julie's car's odometer reading was 35201km when she last filled the fuel tank. Yesterday she checked it again and it read 35403km. Checking the tank, the car used 11 liters of gas to do so. If her car's gas tank holds 35 liters, how long can she drive before running out of gas?

Solution using `MatLab` as a calculator How much has she driven ?

```
>> 35403 - 35201
ans =
202
```

How much distance per liter of gas ?

```
>> 202/11
ans =
18.3636
```

How much gas is left in the tank ?

```
>> 35 - 11
ans =
24
```

Distance she can drive:

```
>> 24 * 18.3636
ans =
440.7264
```

Solution with variables We can script the same code. It means that we would just have to replace the value in the first three lines to update the whole code.

```
>> odormeter_before = 35201 ;
>> odormeter_after = 35403 ;
>> used_gas = 11 ;
>> distance = odormeter_after - odormeter_before ;
>> distance_per_liter = distance/used_gas ;
>> tank_capacity = 35;
>> estimated_distance = (tank_capacity - used_gas) *
    distance_per_liter
440.7264
```

VIII What to remember

`MatLab` is basically a powerful calculator.

VIII a) Key example

Let's summarize in a few examples everything we have seen.

```
>> year = 2018;
>> next_year = year + 1
next_year =
2019
>> angle = pi ;
>> ( 2.*cos(angle) + 1.) ^2
ans =
9.0
```

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 Basics with matrices and vectors

I a) Creation

Follow the procedure:

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

For creating a vector:

```
>> v = [1,2,3]
v =
1 2 3
>> w = [0;1]
w =
0
1
```

`v` is a row vector and `w` is a column vector.

For creating a matrix, 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.

I b) Manipulation

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)`

To add a constant to a vector, a vector to another vector (it works identically for matrices) :

```
>> v = [1 3];
>> v + 4
ans =
5 7
>> v*2.
ans =
2 6
>> w = [2 ,8];
>> v+w
ans =
3 11
>> w/2.
ans =
1 4
```

To access to an element of a vector (it works identically for matrices) :

```
>> v = [1 3 4 -2.5 8] ;
>> v(1)
ans =
1
>> v(3)
ans =
4
>> M = [[5 ,2];[9 , -1]] ;
>> M(1 ,1)
ans =
5
>> M(2 ,1)
ans =
9
```

To delete or *remove* the ith element, use the operator "[]":

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

It hence reduces the dimension of the vector. It works similarly for a matrix:

```
>> A = [ [1 3 4] ; [-2.5 8 9]] ;
>> A(:,1) = []
A =
3 4
8 9
```

The first column has been deleted.

II More details on vectors

II a) Creation of a vector

II a) i Row vector

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

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

It is also possible to replace the commas with space:

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

It works fine but makes the code harder to read.

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
```

II a) ii Column vector

Creating a column vector is similar to creating a row 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
```

II a) iii Transpose operator

It is possible to change a column vector to a row vector, and reciprocally, 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
```

II b) Access to the elements of a vector

II 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
```

II 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
```

Pro Tip

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. IV b) i.

II b) iii Deleting elements

In `MatLab`, using "[]" empty a variable. It is named the *empty vector operator*.

When used on a part of a vector or a matrix, it simply *deletes* this part. It is *gone*. As a consequence, it reduces the dimensions of the matrix.

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

II c) Basic operations on vectors

II 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.
```

Pro Tip

`MatLab` considers vectors as 1D matrices.
It is important when dealing with multiplication.
If `v=[1,2,3]`:

- `v*v'` is 14
  it is multiplying a 1×3 matrix with a 3×1 matrix.
- `v'*v` is `[[1,2,3];[2,4,6];[3,6,9]]`
  it is multiplying a 3×1 matrix with a 1×3 matrix.
- `v*v` will not work
  it is multiplying a 1×1 matrix with a 1×3 matrix; dimensions are not compatible.

II c) ii Multiplication

II c) ii 1 Multiplication by a scalar 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
```

II c) ii 2 dot product 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
```

II c) ii 3 Element-wise multiplication Element-wise multiplication 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
```

Pro Tip

In `MatLab`, using "." in front of an operator means that this operator will be applied element-wise (to each element of the vector/matrix).
For instance, `v.^2` means that *all* the elements of `v` will be squared. If `v` is $[2, 4, 3]$, then `v.^2` is $[2^2, 4^2, 3^2]$, or `[4,16,9]`.

[[TODO exercices]]

III 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.

III a) Addition/subtraction

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

```
>> M = [[1  3];[2,4];
>> M + 4
 ans =
 5 7
 6 8
>> A = M - 2
A =
 3 5
 4 6
```

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

```
>> M = [[1  0];[2,3];  A = [[2, -1];[1,1]];
>> M + A
 ans =
 3 -1
 3  4
>> M - A
 ans =
 1 1
 1 2
```

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

```
>> M =[[1,0];[1,0];[1,0]  ;  A = [[2, -1];[1,1]];
>> x+y
Matrix dimensions must agree.
```

III a) i 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.

III a) i 1 dot product

The first definition is the dot product between two vectors.

> **Definition**
>
> 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
```

III a) i 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
```

[[TODO exercises]]

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

III a) ii Accessing and deleting elements

III a) ii 1 Accessing elements It works in a similar way as for vectors, except that the dimensions are separated with a comma. Let's work with `M`:

```
>> M = [ [1,2,3]; [4,5,6]; [7,8,9]]
M =
1 2 3
4 5 6
7 8 9
```

To extract the third column of `M`:

```
>> c = M(:,3)
c =
3
6
9
```

To extract the second row of `M`:

```
>> l = M(2,:)
l =
4 5 6
```

To extract the first *and* the second row of `M`:

```
>> A = M([1,2],:)
A =
1 2 3
4 5 6
```

Following this notation, we can exchange the first rows of `M`:

```
>> A = M([2,1],:)
A =
4 5 6
1 2 3
```

Pro Tip

Using lists (for instance `indices = [1,3,4]`) as arguments of matrices is a very powerful methods for altering a matrix or for creating a new one.

- `M(indices,:)` will select the rows of `M` in the order of the elements of `indices`. They can be copied, if there is repetition in `indices`, as `indices = [1,2,2]` They can be rearranged, if the elements in `indices` are not ranked, as `indices = [3,1,2]`
- `M(:,indices)` will select the rows of `M` in the order of the elements of indices
- `M(indices_row,indices_col)` will extract a sub-matrix of `M`

III a) ii 2 Deleting elements In `MatLab`, using "[]" empty a variable. It is named the *empty vector operator*.

When used on a part of a vector or a matrix, it simply *deletes* this part. It is *gone*. As a consequence, it reduces the dimensions of the matrix.

It hence reduces the dimension of the vector. It works similarly for a matrix:

```
>> A = [ [1 3 4] ; [-2.5 8 9]] ;
>> A(:,1) = []
A =
3 4
8 9
```

The first column has been deleted.

It is possible to delete one element from a vector

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

But trying to do it for a matrix will lead to an error message.

```
>> A = [ [1 3 4] ; [-2.5 8 9]] ;
>> A(1,1) = []
A null assignment can have only one non-colon index.
```

You can remove a block, a column or a row from a matrix. But you can not remove a single element, if not, there would be a "hole".

IV Useful built-in functions for vectors and matrices

IV a) Dimensions of a matrix

The command `size` send back the dimensions of a matrix.

```
>> A = [ [1 3 4] ; [-2.5 8 9]] ; v = [1 2 3 4];
>> size(A)
ans =
2 3
>> size(A(:))
ans =
6 1
>> size(A(1,:))
ans =
```

```
1 3
>> size(v)
ans =
1 4
>> size(v')
ans =
4 1
```

To reuse the size, you can store it in variables:

```
>> A = [ [1 3 4] ; [-2.5 8 9]] ;
>> size_a = size(A);
>> size_a(1)
ans =
3
>> [n_x,n_y] = size(A);
>> n_x
ans =
3
>> n_y
ans =
2
```

IV b) Generating a vector

IV b) 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.
```

Table 2.1: A few functions that generate vectors.

name	description	illustration
colon operator :	see Sec IV b) 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]])`

Table 2.2: A few functions that generate matrices.

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])`

Table 2.3: A few functions that are useful.

name	description	illustration
`size(M)`	number of elements in M	`size([1,2,3])`
`shape(M)`	number of elements in each direction of M	`shape([2,3,5];[1,2,3]])`

IV b) ii Other functions

IV c) Generating a matrix

V Linear Algebra

VI What to remember

VI a) Creating a matrix

Follow the procedure:

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

VI b) Manipulating matrices

You can

- add (with "+"), subtract (with "-"), multiply (with "*") 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)`

VI c) Key example

Let's summarize in a few examples everything we have seen.

```
>> M = [[1 ,2];[3 ,4]]
M =
 1  2
 3  4
```

Chapter 3

Plotting data

One of the usefulness of `MatLab` is being able to easily plot, manipulate and modify data.
With only a few commands, you can have publication-ready figures.

I Creating the first plot

First one needs data.

```
>> x = [1 2 3 4 5 6 7 8 9 10];
>> y = [1 4 9 16 25 36 64 81 100];
```

To plot the data, the main command in `MatLab` is `plot`:

```
>> plot(x,y)
```

[[TODO ** image plot 1 **]]

Pro Tip

When only one data is provided, `MatLab` assumes that the abscisse are the indexes of the elements of the data.
What will be plotted is hence the coordinates $(1, y(1)), (2, y(2), \ldots, (n, y(n))$.
Compare `plot(x,y)` and `plot(y)`!
When trying to understand some results, never hesitate to plot some data `y` by typing `plot(y)` in the command prompt. Quick and dirty, but useful.

II A beautiful plot

II) **1 axis** It is *really important* to always have labels for the axis.

```
>> x = [1 2 3 4 5 6 7 8 9 10];
>> y = [1 4 9 16 25 36 64 81 100];
>> plot(x,y)
>> xlabel('data along x')
>> ylabel('data along y')
```

`xlabel` and `ylabel` are the `MatLab` functions that handle the labels.
You can see that the argument for these functions are between quotes "". A label is some text: it means that `MatLab` will need a string.

```
>> x = 0:pi/10:2.*pi;
>> y = cos(x);
>> plot(x,y)
>> str_xlabel = 'x';
>> xlabel(str_xlabel)
>> str_ylabel = 'cos(x)';
>> ylabel(str_ylabel)
```

II) **2 Colours and line styles** The default plot color in `MatLab` is blue, and the line default style is plain.

To change the style, an extra argument has to be sent to `MatLab`.

For instance,

- `plot(x,y,'r')` will change the color of the plot to red.
- `plot(x,y,'--')` will change the line style of the plot to dashed.

Note that the argument is between quotes. Arguments are generally passed as a string

Table 3.1: A few colours in `MatLab`.

colours	code	illustration
black	k	`plot(x,y,'k')`
green	g	`plot(x,y,'g')`
red	r	`plot(x,y,'r')`
blue	b	`plot(x,y,'b')`
yellow	y	`plot(x,y,'y')`

Table 3.2: A few line styles in `MatLab`.

style	code	illustration
solid	-	`plot(x,y,'-')`
dashed	–	`plot(x,y,'--')`
dotted	:	`plot(x,y,':')`
dash-dot	-.	`plot(x,y,'-.')`

Table 3.3: A few marker styles in `MatLab`.

marker	code	illustration
plus	+	`plot(x,y,'+')`
cross	x	`plot(x,y,'x')`
circle	o	`plot(x,y,'o')`
triangle up	ˆ	`plot(x,y,'^')`
triangle down	v	`plot(x,y,'v')`
triangle left	<	`plot(x,y,'<')`
triangle right	>	`plot(x,y,'>')`

Pro Tip

A good way of not messing with the style is to follow this order:

1. open the quotes: `'`
2. add the color, e.g. k: `'k`
3. add the style of the line, e.g. –: `'k--`
4. add the style of the markers, e.g. x: `'k--x`
5. end the quotes: `'k--x'`

you can now plot with the options:
`plot(x,y,'k--x')`

Pro Tip

A good plot has always:

- labels on the x-axis and y-axis
- different colors or patterns for different curves
  If possible, these differences should persist when printing the figure in black and white.

If the figure is in a report :

- instead of a title, the figure can be explained in the text under the figure.
- instead of a legend, the difference between curves can be explained in the text under the figure.

II a) Title

The command `figure` is used here for opening a new figure. If not, `MatLab` will draw any new plot in the previous figure, erasing the old figure.

III Edit a plot in the window

A way to edit the plot properties is to click on the arrow - Edit Plot - on the plot window.

[[TODO ** image fig plot with arrow **]]

Then, double clicking on the curve will open an extra window where you can play with the properties.

In particular there is two interesting properties:

1. `Line`, where you can edit the style of line (plain, dashed, etc.), the width of the line, and its color
2. `Marker`, where you can edit the type of markers (indication where each data is plotted, for instance with circles), the size of the markers, the color of the inside of the marker as well as the color of their edges

Bibliography