POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

Introduction To MATLAB

Contacts

- Sara Mandelli: sara.mandelli@polimi.it

- Slides and scripts will be uploaded on http://marconlab.deib.polimi.it/courses/multimedia-signal-processing/mmsplaboratories

MATLAB

- MATLAB = MATtrix LABoratory
  - numerical computing environment and programming language
  - Useful for working with matrixes

Exam

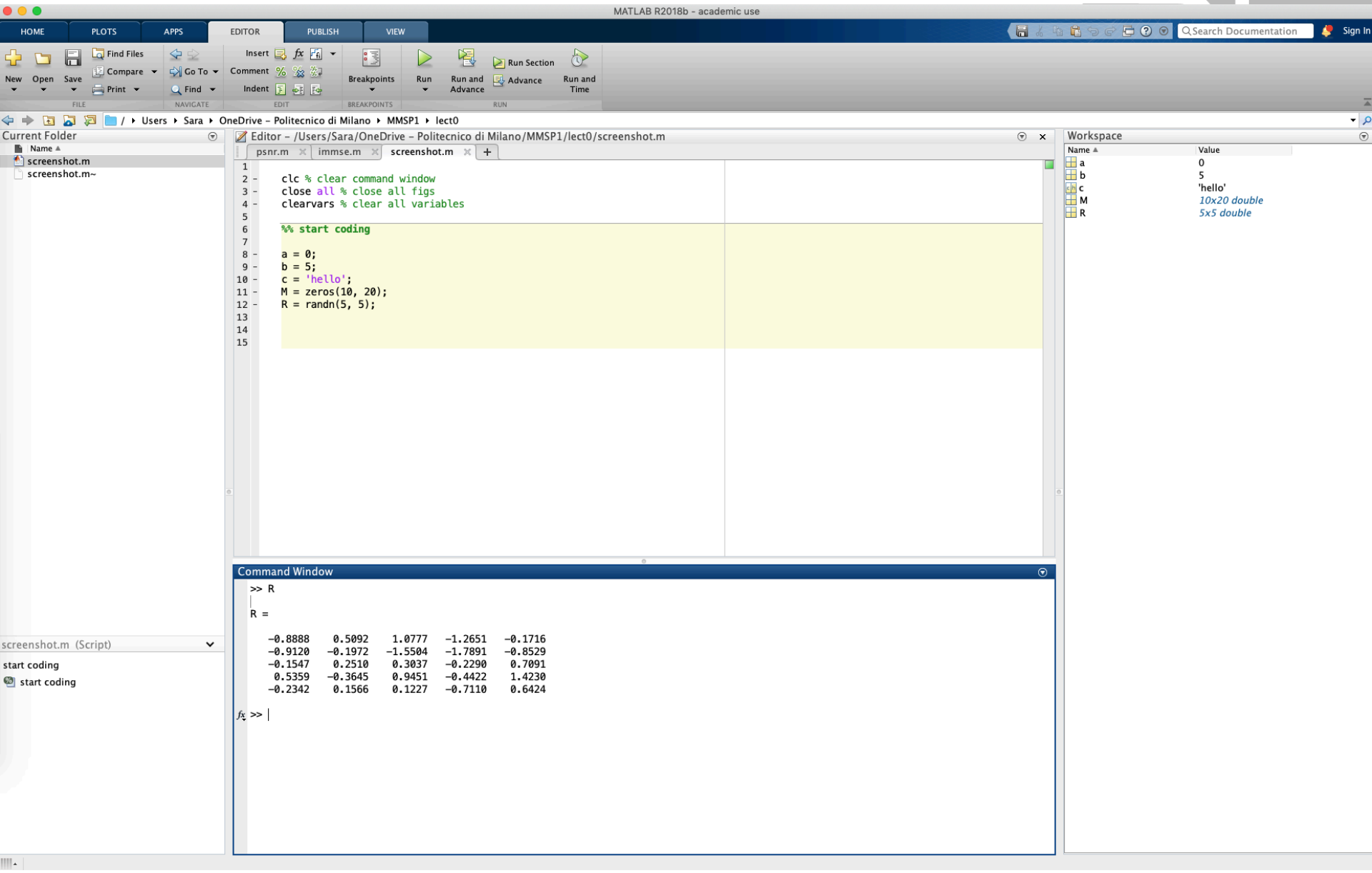
- 11 pt.
- MATLAB code written on paper

How to use MATLAB @polimi

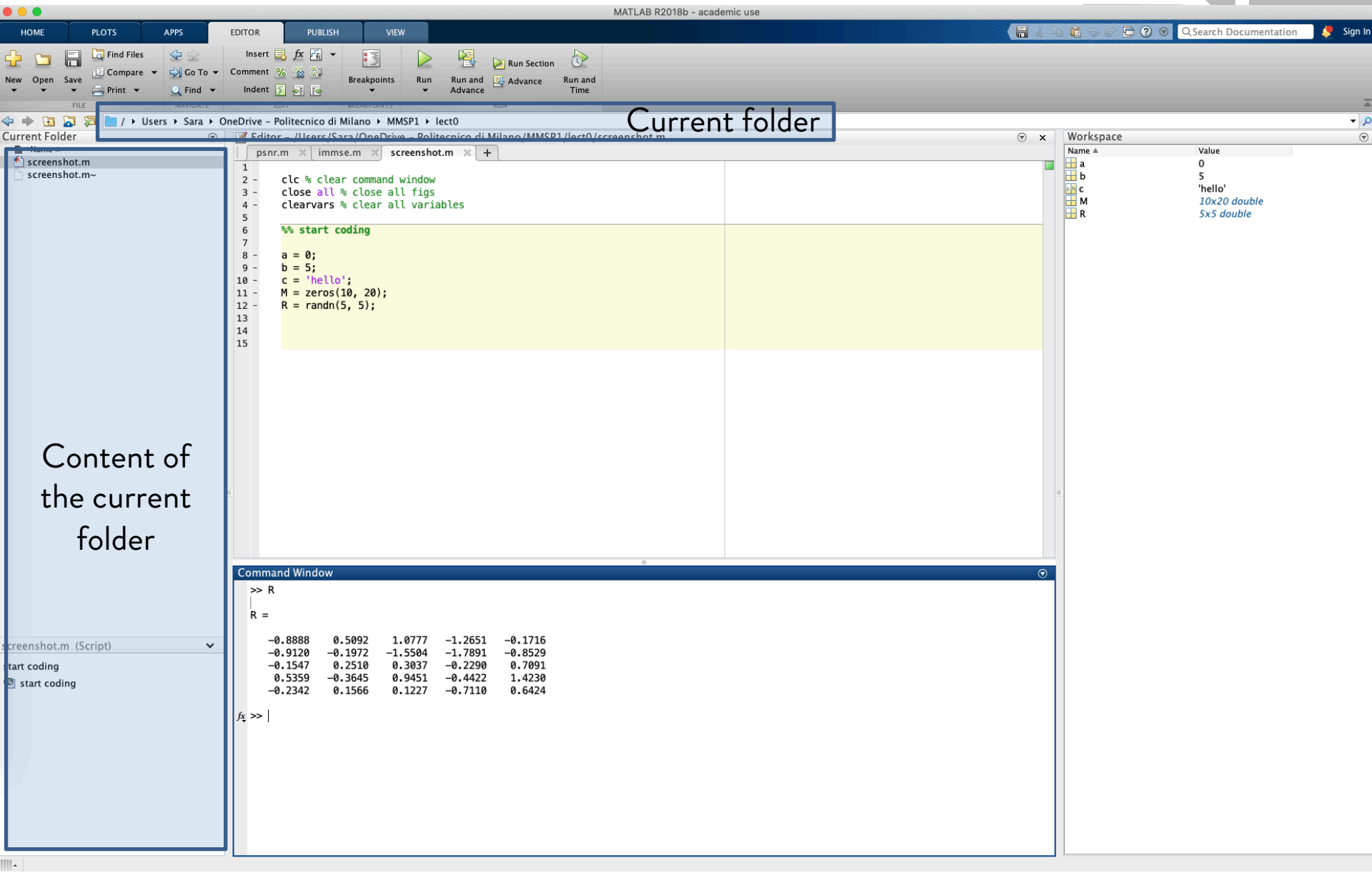
You can install it and use from your personal pc
https://www.software.polimi.it/en/software-download/students/matlab/

You can use MATLAB from virtual desktop
https://virtualdesktop.polimi.it

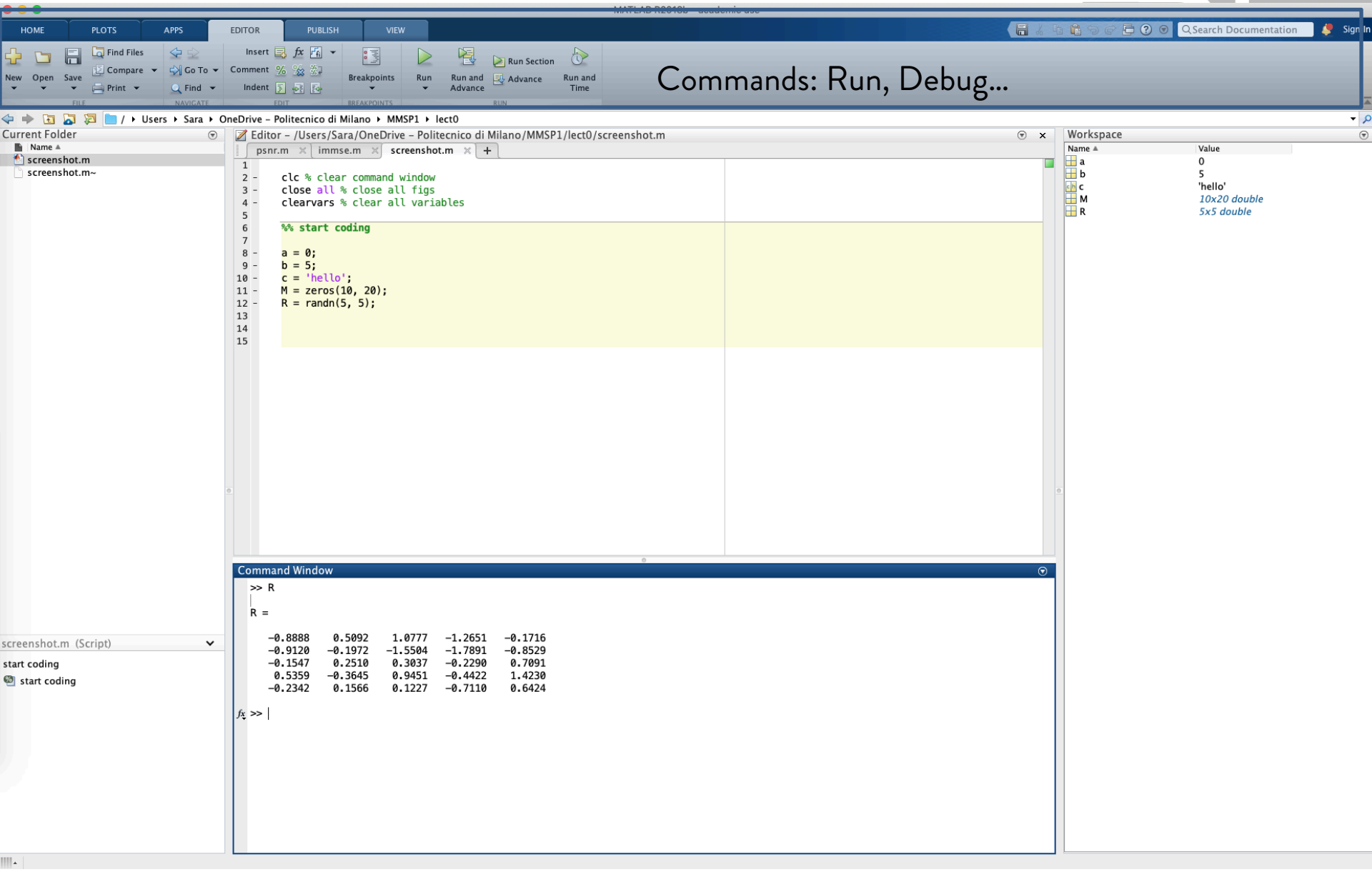
MATLAB screenshot



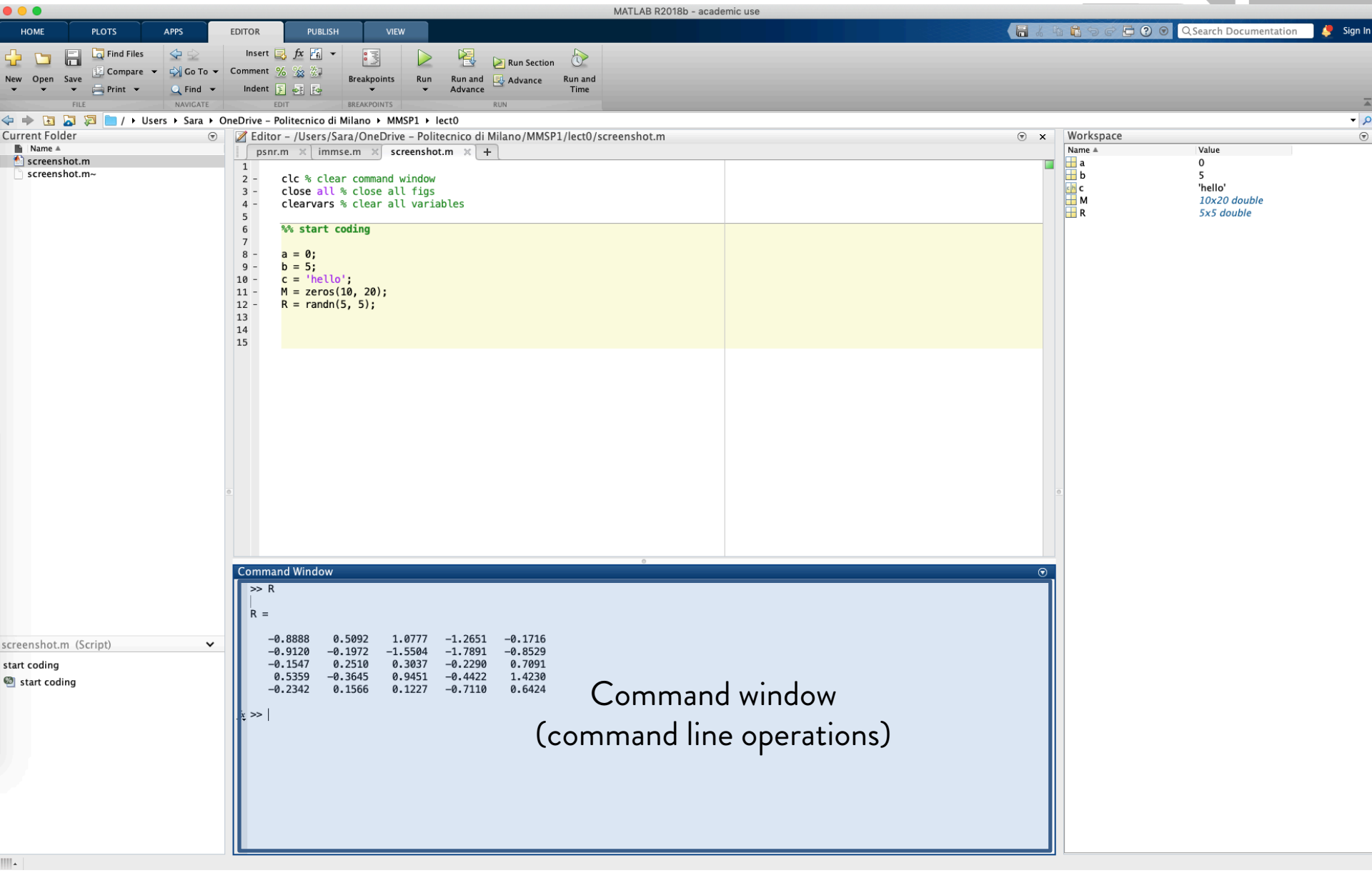
MATLAB screenshot



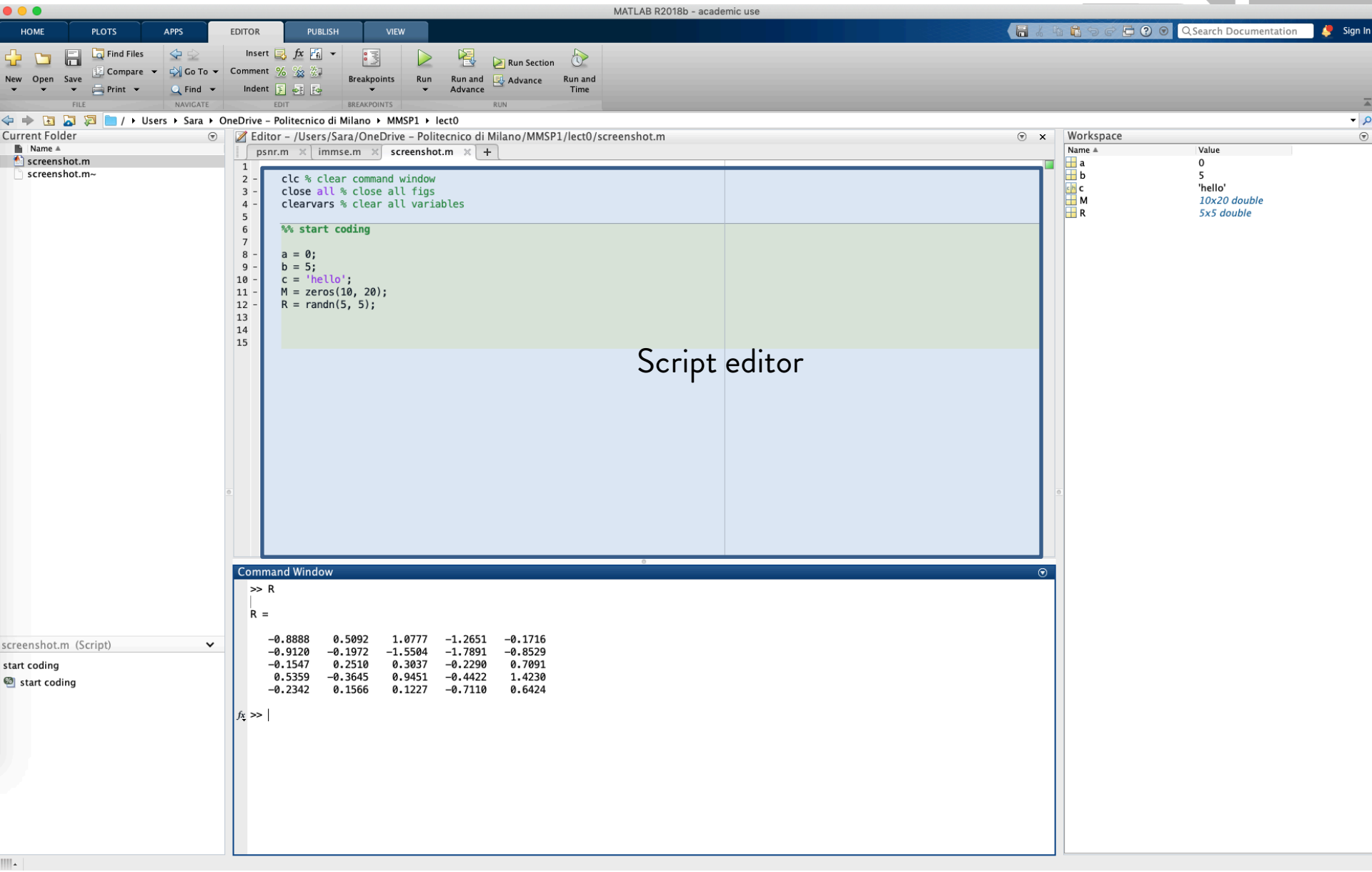
MATLAB screenshot



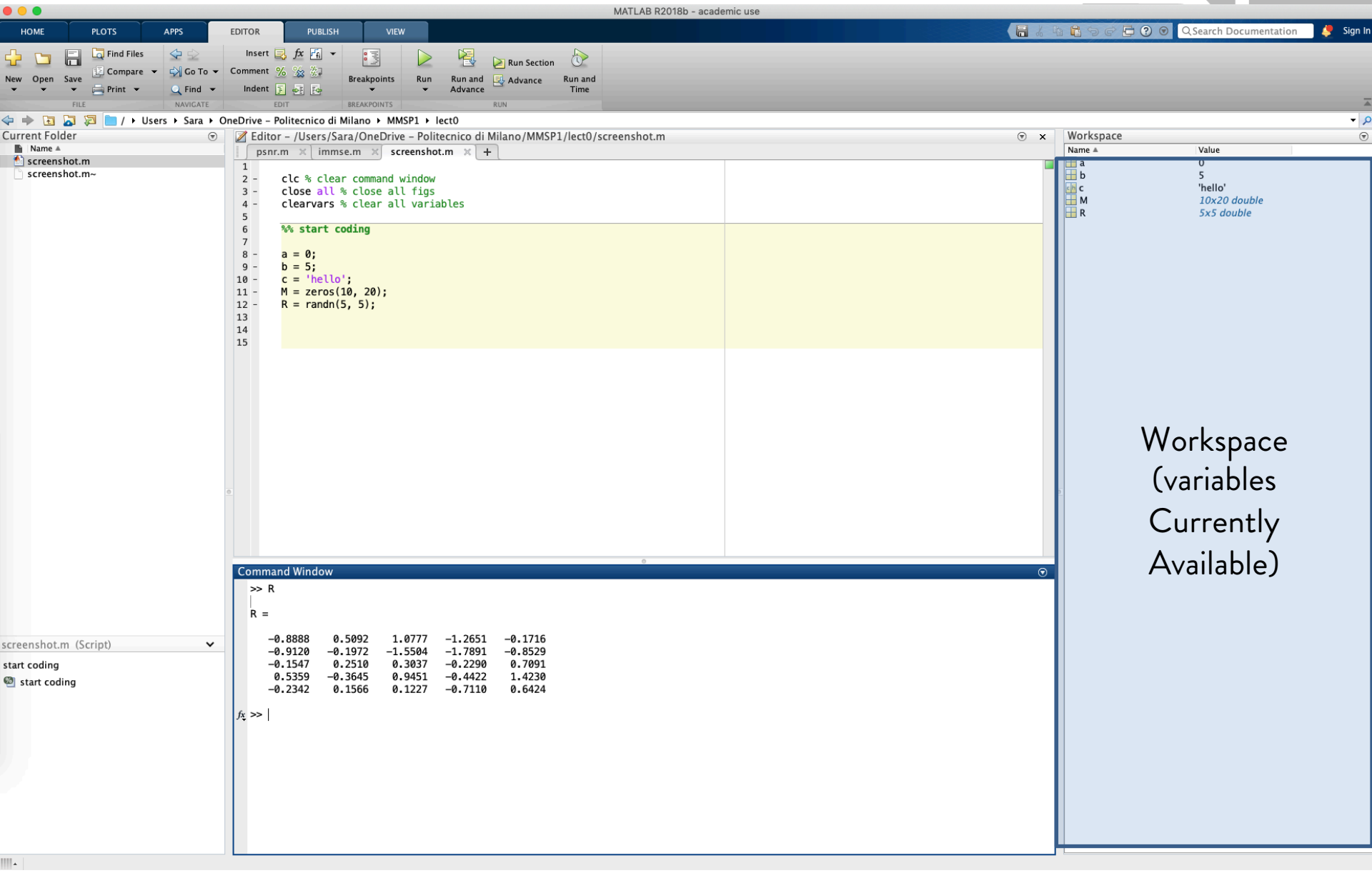
MATLAB screenshot



MATLAB screenshot



MATLAB screenshot



MATLAB fundamentals

Matlab fundamentals

- '>>' indicates a command in command window

```
>> a = 0;
```

- Type '%' to comment code (comments are in green)

```
% this is a comment
```

- Insert ';' at the end of line
  otherwise the output is shown in command window

```
>> A = 0;
>> A = 0

A =

     0

>> |
```

Matlab variables

- You don't need to declare variables before to assign them

```
a = 5;
```

- If you do not assign the output of a statement to a variable, MATLAB assigns the result to the reserved word 'ans'.

```
>> 3+5

ans =

     8
```

- By default, MATLAB stores all numeric values as double-precision floating point (64 bits)

- **Every numerical variable is an array (1D, 2D, 3D…)**

How to define arrays

- Array elements are contained in square brackets
- Row vector: each element is separated

  either by comma or blank space

```
row = [1, 3, 4, 6];  row = [2 3 5 7];
```

- Column vector: each element is separated by semicolon

```
column = [3; 4; 5];
```

- Matrix N x M: N rows by M columns

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

matrix =

     1     3     4
     5     5     6
     7     8     9
```

How to define arrays

- Dimensions must be consistent!

  When creating matrixes:

  - Blank space or comma defines a new column
  - Semicolon defines a new row
  - Be careful in concatenating rows and columns!

```
matrix = [1, 2; 1, 3, 4; 5];
```

$$\begin{bmatrix} 1 & 2 & ? \\ 1 & 3 & 4 \\ 5 & ? & ? \end{bmatrix}$$

This is NOT a matrix!

- If you run this code, MATLAB reports the error (in red)

```
>> matrix = [1, 2; 1, 3, 4; 5];
Dimensions of arrays being concatenated are not consistent.
```

How to define arrays

- To check array dimensions, type ‘size(your_array)’
  It returns an array with
  (#elements 1° dim, # 2° dim, # 3° dim ...)

- With 1D arrays, use ‘length(your_array)’
  It returns the # of array elements

How to define arrays

- You can create a matrix full of zeros specifying the dimensions (#elements 1° dim, #elements 2° dim, etc..)

```
>> zero_matrix = zeros(3, 4);
>> zero_matrix

zero_matrix =

     0     0     0     0
     0     0     0     0
     0     0     0     0
```

- You can create a matrix full of ones ('ones(matrix size)'), etc..

How to define arrays

- Define a range of values (1° method)

  i_value(included) : step_size : f_value(included)

```
>> values = 0:0.1:1

values =

         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000    0.8000    0.9000    1.0000
```

- Define a range of values (2° method)

  linspace(i_value(included), f_value(included), #elements)

```
>> values = linspace(0, 1, 11)

values =

         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000    0.8000    0.9000    1.0000
```

Both are row vectors
But methods are slightly different

Indexing 1D arrays

- Select an array element → **MATLAB starts from 1!**
  Include in round brackets the index you look for

```
>> a = [0, 1, 2, 4];
>> first_a = a(1)

first_a =

     0
```

- Select the last element → 'end' means the last element

```
>> a(end)

ans =

     4
```

- Select multiple elements

```
>> a(1:2:end)

ans =

     0     2
```

Indexing ND arrays

- Include in round brackets (1° dim coordinates, 2° dim coordinates, etc…)

```
>> A = [1, 2, 3; ...
        4, 5, 6; ...
        7, 8, 9];
>> A(2, 3)

ans =

     6
```

'…' is used to continue on next row

Indexing ND arrays

- Select some rows / columns

```
>> A(2:3, 1:2)

ans =

     4     5
     7     8
```

- Select all rows and last column

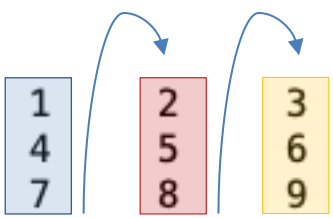
```
>> A(:, end)

ans =

     3
     6
     9
```

':' is used to select all elements in one dimension

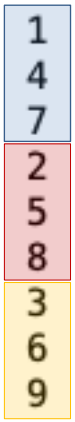
Linear indexing of ND arrays

- Include ONLY one subscript in round brackets
- MATLAB treats the array as a long column vector, by going down the columns consecutively. To visualize it:

```
A =
```




```
>> A(:)

ans =
```




```
>> A(2, 3)

ans =

     6
```

```
>> A(8)

ans =

     6
```

One single subscript == linear indexing

Logical indexing of ND arrays

- Use a logical array for the matrix subscript.
- MATLAB extracts the elements in column-order, and returns a column vector

```
A =

     1     2     3
     4     5     6
     7     8     9
```

```
>> A(A <= 5)

ans =

     1
     4
     2
     5
     3
```

Operations on arrays

- Symbols '+, -' are used for addition and subtraction (always element-wise)

```
>> A = [1, 2, 3;
     4, 5, 6;
     7, 8, 9];
B = [2, 4, 6;
     3, 1, 5;
     2, 4, 5];
C = B + A

C =

     3     6     9
     7     6    11
     9    12    14
```

```
>> A = [1, 2, 3;
     4, 5, 6;
     7, 8, 9];
B = [2, 4, 5];
C = B + A

C =

     3     6     8
     6     9    11
     9    12    14
```

```
>> A = [1, 2, 3;
     4, 5, 6;
     7, 8, 9];
B = [2; 4; 5];
C = B + A

C =

     3     4     5
     8     9    10
    12    13    14
```

- Dimensions must be consistent!

```
>> A = [1, 2, 3;
     4, 5, 6;
     7, 8, 9];
B = [2; 4];
C = B + A
Matrix dimensions must agree.
```

Operations on arrays

- Symbol '*' is used for product by a scalar and matrix product

```
a = 2;
b = [2, 3, 4];
c = a * b;
```

```
c =

     4     6     8
```

```
A = [1, 2, 3; ...
     4, 5, 6; ...
     7, 8, 9];
B = [2, 4, 5];
C = B * A;
```

```
C =

    53    64    75
```

Dimensions must be consistent!!!

```
A = [1, 2, 3; ...
     4, 5, 6; ...
     7, 8, 9];
B = [2; 4; 5];
C = B * A;
```

```
Error using *
Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix matches the number of rows in
the second matrix. To perform elementwise multiplication, use '.*'.
```

Operations on arrays

- Symbol '*' is used for product by a scalar and matrix product

```
a = 2;
b = [2, 3, 4];
c = a * b;
```

```
c =

     4     6     8
```

```
A = [1, 2, 3; ...
     4, 5, 6; ...
     7, 8, 9];
B = [2, 4, 5];
C = B * A;
```

```
C =

    53    64    75
```

Dimensions must be consistent!!!

```
A = [1, 2, 3; ...
     4, 5, 6; ...
     7, 8, 9];
B = [2; 4; 5];
C = B * A;
```

TRANSPOSE B TO MAKE IT WORKS
(transpose(B) = B')

```
Error using *
Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix matches the number of rows in the second matrix. To perform elementwise multiplication, use '.*'.
```

Operations on arrays

- Symbol '.*' is used for element-wise product between two arrays

```
>> A = [1, 2, 3;
     4, 5, 6;
     7, 8, 9];
B = [2, 4, 6;
     3, 1, 5;
     2, 4, 5];
C = B.*A

C =

     2     8    18
    12     5    30
    14    32    45
```

```
>> A = [1, 2, 3;
     4, 5, 6;
     7, 8, 9];
B = [2, 4, 5];
C = B.*A

C =

     2     8    15
     8    20    30
    14    32    45
```

```
>> A = [1, 2, 3;
     4, 5, 6;
     7, 8, 9];
B = [2; 4; 5];
C = B.*A

C =

     2     4     6
    16    20    24
    35    40    45
```

- Dimensions must be consistent!!!

```
>> A = [1, 2, 3;
     4, 5, 6;
     7, 8, 9];
B = [2; 4];
C = B.*A
Matrix dimensions must agree.
```

Operations on arrays

- Symbol '\' is used for solving linear systems b = A*x
- The inverse of A can be computed as
  - 'A^{-1}' , 'inv(A)', 'pinv(A)' → inverse or pseudo-inverse
- x can be directly computed as x = A\b

  calculation is quicker and has less residual error.

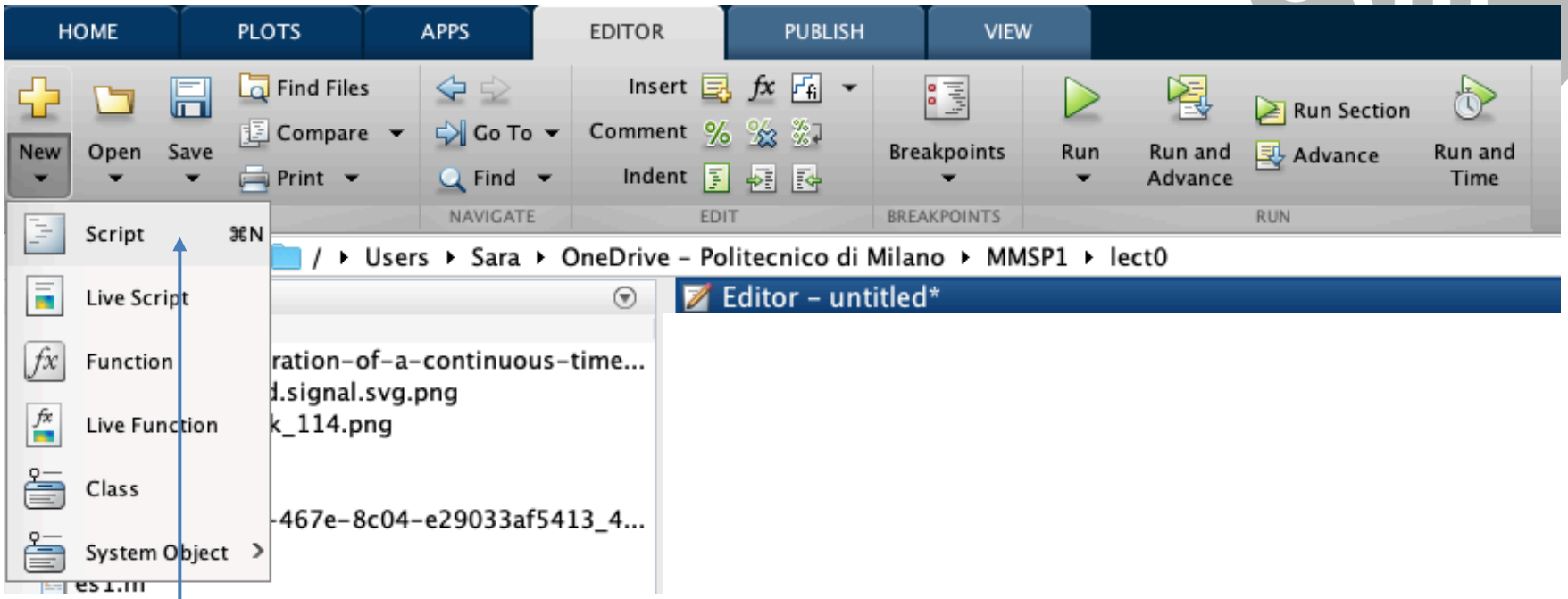
More complex tools

Scripts and functions

- Script:
  - *.m file
  - Used to write a program that performs complex tasks
  - Can call functions

- Function:
  - *.m file
  - Used to encapsulate an algorithm
  - Receives inputs (*parameters*) and returns outputs (*result*)

Writing a script



Select New → Script

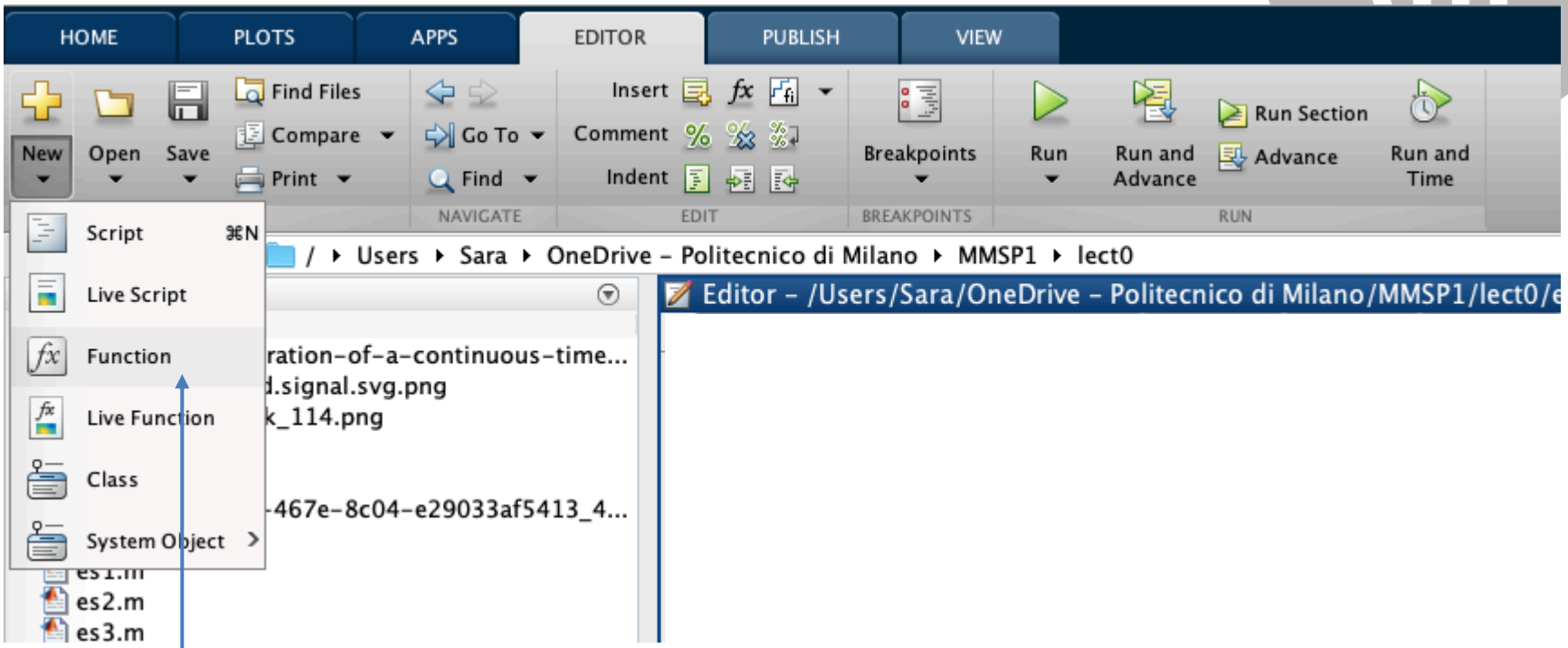
Remember to always start the script with

```
% Begin always with these three lines:
close all % close figures
clearvars % clear workspace
clc % clear command window
```

Writing a script

- Save the script as ‘my_script_name.m’
- Inside the script you can call functions

Writing a function



Select New → Function

```
function [outputArg1,outputArg2] = untitled(inputArg1,inputArg2)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
outputArg1 = inputArg1;
outputArg2 = inputArg2;
end
```

Writing a function

- Save the function as ‘my_function_name.m’
- The function can be called as

  ‘outputs = my_function(parameters)’

Loops

- Loops allow to repeat the execution of a part of your code for a certain number of iterations
- 'for' loop

```
x = ones(1,10);
for n = 2:2:10

    x(n) = 2 * x(n - 1);

end
```

You can write as many loops as you want…
But it is not recommended!

- 'while' loop

```
x = ones(1, 10);
n = 1;
while n < 10

    x(n) = 2 * x(n + 1);
    n = n + 1;

end
```

Loops

- Loops allow to repeat the execution of a part of your code for a certain number of iterations
- 'for' loop

```
x = ones(1,10);
for n = 2:2:10

    x(n) = 2 * x(n - 1);

end
```

- 'while' loop

```
x = ones(1, 10);
n = 1;
while n < 10

    x(n) = 2 * x(n + 1);
    n = n + 1;

end
```

You can write as many loops as you want…
But it is not recommended!

REMEMBER TO WRITE 'END'

Conditional execution

- The 'if' statement allows to execute part of the code only if a condition is satisfied.

```
a = 0.1;

if a <= .1
    c = 10;
elseif a >.1 && a <=.3
    c = 7.5;
else
    c = 5;
end
```

'elseif is placed after 'if'

'else' is placed after 'if' and 'elseif'

- Possible conditions:
  - A number (0→ false, non-zero → true)
  - A comparison (>, <, =, etc…)
  - A combination of conditions (& → and, | → or, ~ → not)

These were just examples...

For any information, click here