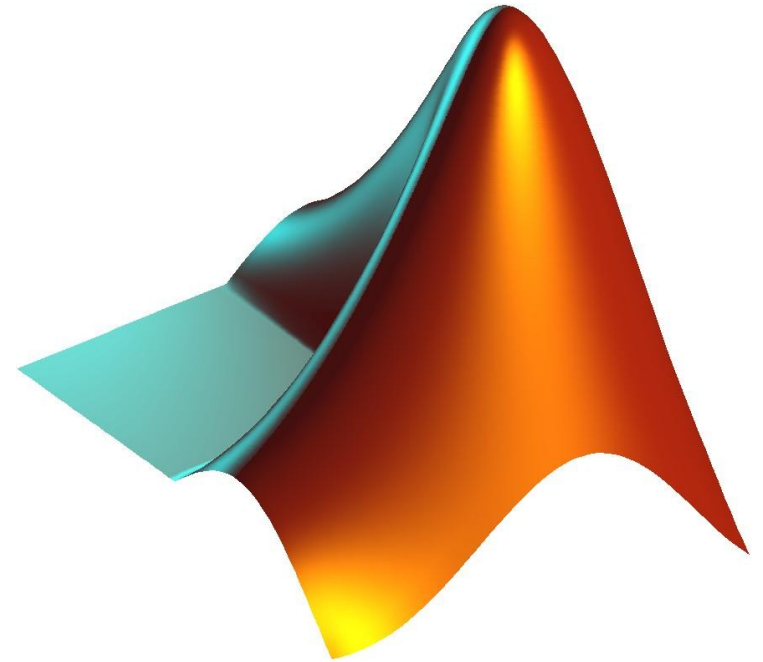


LAB WEEK 1



Two-Year Master in Mathematics
Numerical Methods - A.Y. 2024-25

MATLAB: (Practical) Review



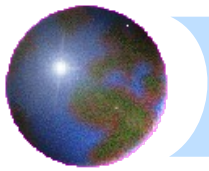
Notes adapted from Prof. Fabiana Zama's slides by Liwei Hu



Self-Introduction and Contact



<https://www.unibo.it/sitoweb/liwei.hu2/en>



Overview

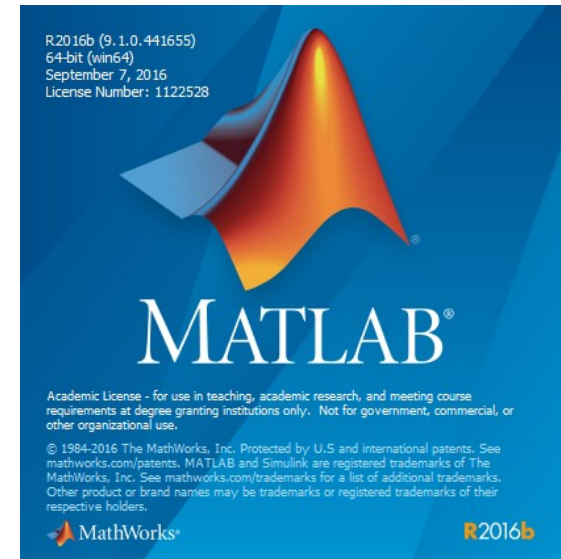


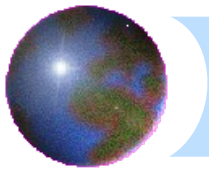
MATLAB allows to solve many mathematical and application problems, for example:

- Matrix computation
- Polynomials rootfinding
- Signal and image processing (toolbox)
- Graphics: x-y and polar, 3D graphics

Especially useful for:

- Development of algorithms
- Modeling, simulation and prototyping
- Data acquisition
- Data analysis and visualization
- Applications development, with graphical user interfaces



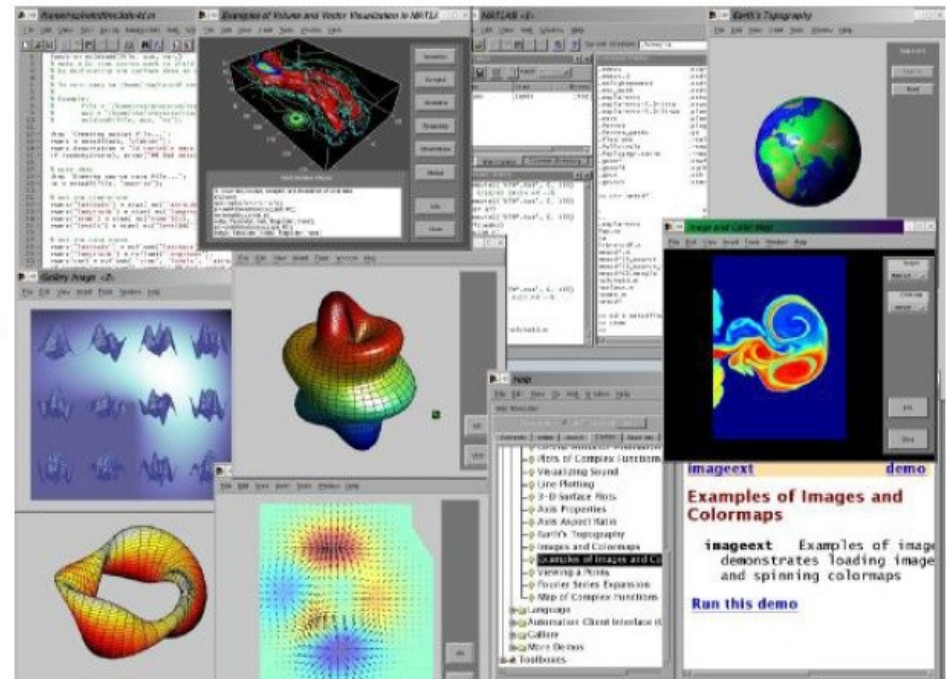


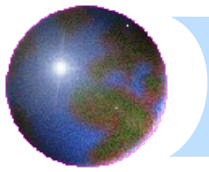
Overview



MATLAB is an **interpreted language**, errors are easy to find but it is slow in **execution**. (An interpreted language is a programming language whose instructions are translated into machine language only at the time of execution).

- 2-D and 3-D graphics for data visualization
- Tools to build convenient graphical user interfaces
- Functions to integrate MATLAB algorithms with external applications and languages, such as C, C ++, Fortran, Java, COM, and Microsoft Excel





Overview



- *Installation (and configuration)*
- *Matlab desktop*
- *Documentation*
- *Matlab programming*
 - *Variables*
 - *Basic commands and expressions*
 - *Common data types*
 - *Common operations and control structures*
 - *Matlab scripts*
 - *Matlab functions, function handle, and matlab toolbox*



Installation on your pc



Follow instructions from:

<https://www.mathworks.com/academia/tah-portal/alma-mater-studiorum-universita-di-bologna-1122528.html>

- Create a MathWorks Account with the university email address, i.e. the one ending with “@studio.unibo.it”;



(Optional) Configuration



For Windows users on LAB computers:

In order to perform copy-and-paste as in Windows, go to:

Preferences>Keyboard>Shortcuts

and switch from “Emacs Default Set” to “**Windows Default Set**”

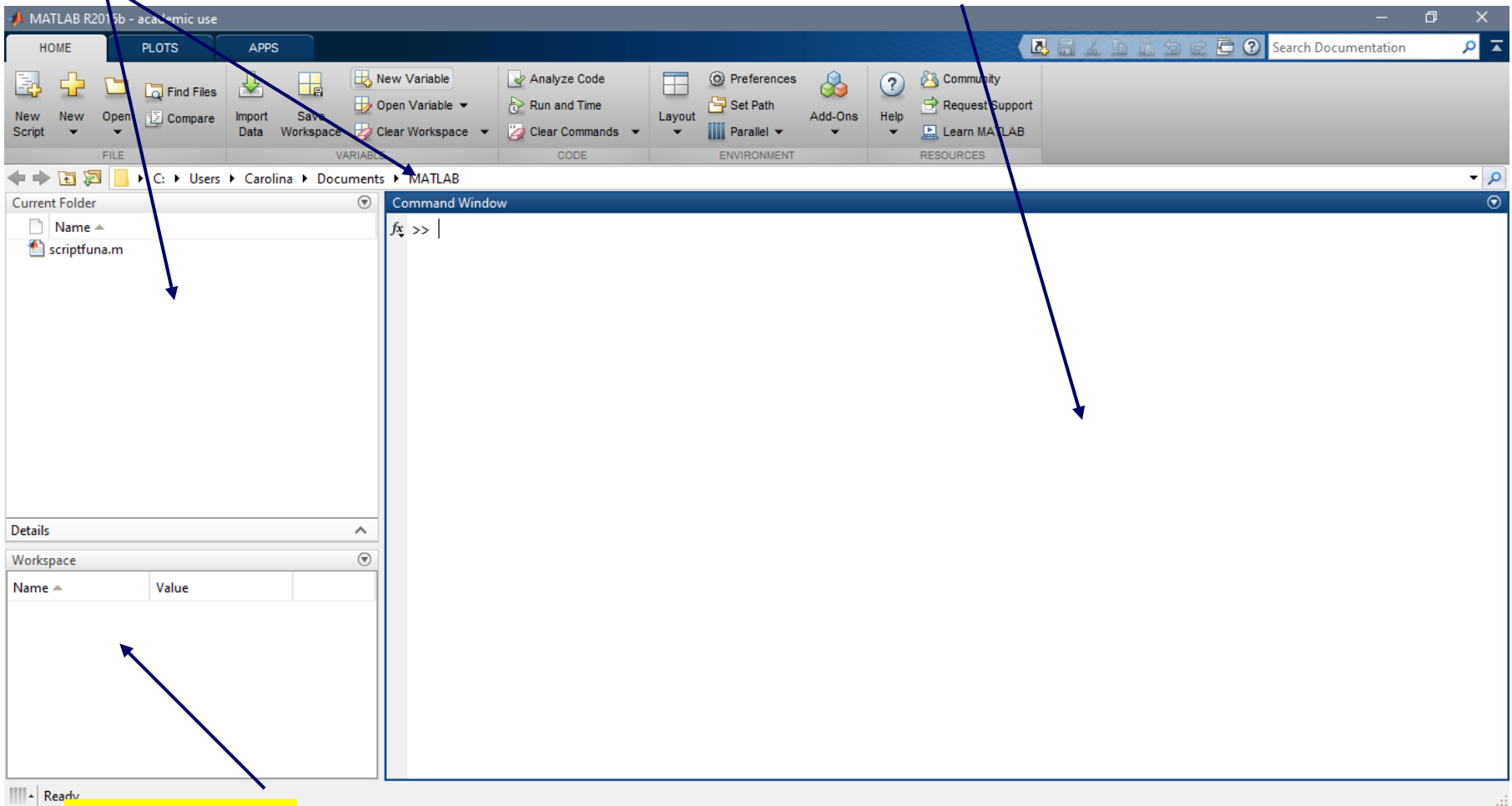


Matlab Desktop



Current Folder and Working Directory

Command Window



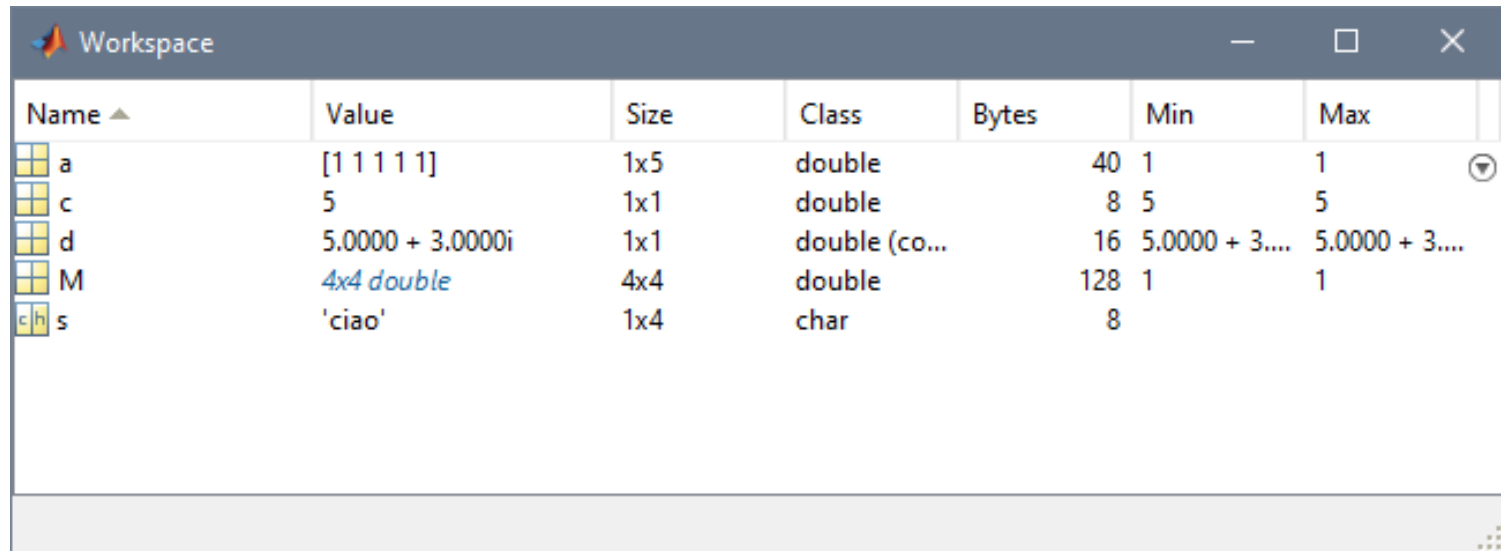
Workspace



Workspace

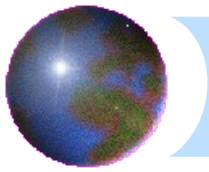


The Matlab Workspace displays all defined variables (Name and Value) together with their dimension (Size) and data type (Type);



Name ▲	Value	Size	Class	Bytes	Min	Max
a	[1 1 1 1 1]	1x5	double	40	1	1
c	5	1x1	double	8	5	5
d	5.0000 + 3.0000i	1x1	double (co...	16	5.0000 + 3....	5.0000 + 3....
M	4x4 double	4x4	double	128	1	1
s	'ciao'	1x4	char	8		

Alternative way to view these information for a variable from the Command Window: “*whos* ***VARIABLE_NAME***”

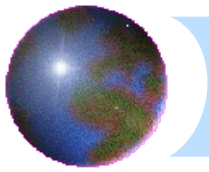


Command Window



The command window allows you to interact

- With MATLAB just like you type on a calculator
- *Copy and paste* of operations for easy repetition of commands
- Use ‘up-arrow’ key  to repeat commands (*listed in the command history*)



Documentation



- Matlab Help is an extremely powerful tool for learning.
- The Help not only contains theoretical information, but also shows examples for implementation.

```
Command Window

>> help sqrt
sqrt    Square root.

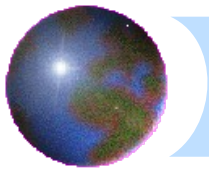
sqrt(X) is the square root of the elements of X. Complex
results are produced if X is not positive.

See also sqrtm, realsqrt, hypot.

Reference page for sqrt
Other functions named sqrt

fx >> |
```

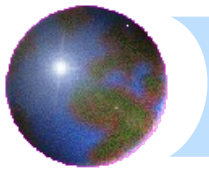
To visualize all information in a new window “*doc NAME*” instead of “*help NAME*”.



Variable Naming Rules I



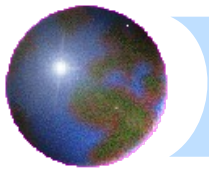
- MATLAB has some rules for variable names
 - Names of variables are **case sensitive**
 - ❖ BOB, Bob, bob, BoB are all different names.
 - Variable names can be at most 31 characters. Each additional character is ignored.
 - Variable names (and files!) **must start with a letter**, followed by other letters, digits or underscores (**no spaces**). Punctuation symbols are not allowed, because many of them have special meaning in MATLAB.



Variable Naming Rules II



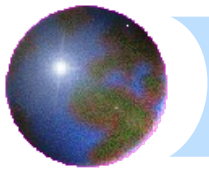
- Some words can not be used as names for variables: *for, end, if, while, function, return, elseif, case, otherwise, switch, continue, else, global, break*
- MATLAB has a number of special variables and constants



Special variables and constants



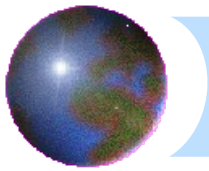
NAME	Description
ans	Default variable to store the result of a command
eps	Machine precision
i,j	Imaginary unit, i.e. $\sqrt{-1}$
Inf	∞
NaN	Undefined numerical result (Not a Number)
pi	π
realmax	Maximum real floating point number
realmin	Minimum real floating point number



Basic commands



Command	Description
clc	Empty the Command window
clear	Deletes all variables
clear var1 var2	Deletes listed variables
exist ('<i>name</i>')	Determines whether <i>name</i> (file or var) exists
quit	Quit MATLAB
who	List of current variables
whos	List of current variables with their size
:	Generates an array
,	Separates elements of an array
;	Suppress output; also newline in an array
...	Continuation of a line of code



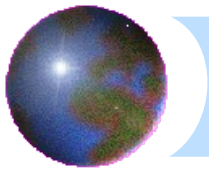
Basic syntax



- ❑ Assign a value to a variable
 - Syntax: **Variable = Expression**
 - Example: **$a = 1 + 3*b - 4*c^2$;**

- ❑ How to use the semicolon ;
 - **with semicolon - the output is suppressed**
 - without semicolons - output is displayed in the command window

- ❑ Adding comments
 - Add **%** at the beginning of a line
 - Example: **% Questa è una demo**



Common Data types



Common data types are

- scalar, character, logical
- vectors, char array
- matrices, tensors, etc..

a = sqrt(3);



Double

c = -2+i*8.2



Complex double

t = 1:10



Array of double

f = 1>0



Logical

q = 'k'

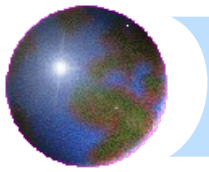


Char

var_txt = 'This string is assigned to the variable var_txt'



Char array



Row Vector (n by 1 Matrix)



Square brackets are used to define a matrix

```
» X=[2,3 7]
```

X =

2 3 7

Space or comma are used to separate elements in a single row

Feedback from MATLAB



Column Vector (1 by n matrix)



Square brackets are
used to define a matrix

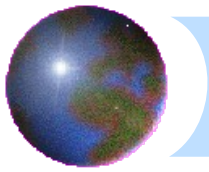
```
» X=[2;3 ; 7 ]
```

```
X =  
  2  
  3  
  7
```

the **semicolon** is used to end a
line.

Also ENTER can be used to end a
line

Feed-back from MATLAB



Automatic generation of vectors



>> x = 0 : 0.2 : 1 from 0 to 1, with increment 0.2

x =

0 0.20 0.40 0.60 0.80 1.00

>> x = linspace(0, 1, 6) 6 values from 0 to 1, with linear spacing

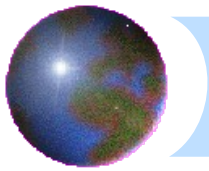
x =

0 0.20 0.40 0.60 0.80 1.00

>> x = logspace(0,1,6) 6 values from 10^0 to 10^1 with logarithmic
spacing ($\log(x(i)) - \log(x(i-1)) = \text{const.}$)

x =

1.0000 1.5849 2.5119 3.9811 6.3096 10.0000



Matrix generation



>> B = [1 2 ; 8 9] **matrix defined 'manually', element by element**

ans =
1 2
8 9

>> ones(2,2) **generates a 2 x 2 matrix with elements equal to 1**

ans =
1 1
1 1

>> zeros(2,3) **generates a 2 x 3 matrix with elements equal to 0**

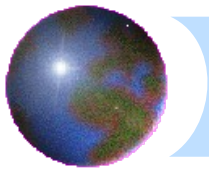
ans =
0 0 0
0 0 0

>> rand(3,3) **generates a 3 x 3 matrix of random numbers**

ans =
0.4447 0.9218 0.4057
0.6154 0.7382 0.9355
0.7919 0.1763 0.9169

>> eye(2) **generates the 2 x 2 identity matrix**

ans =
1 0
0 1



Common Operations: sub- matrix manipulation



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

1	2	3
4	5	6
7	8	9

```
>> x = A ( 1, 3 )
```

A(row index , column index)

x =

3

```
>> y = A ( 2 , : )
```

select the 2nd row

y =

4 5 6

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

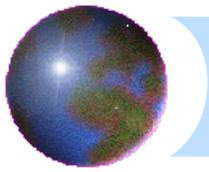
select a submatrix

z =

1	2	3
4	5	6

How to select the last
column?

A(:,end)



Concatenate, append



```
>> R = [ 1 2 3 ];  
>> S = [ 10 20 30 ];  
>> T = [ R S ]
```

T =

1 2 3 10 20 30

```
>> Q = [ R ; S ]
```

Q =

1 2 3
10 20 30

```
>> Q ( 3, 3 ) = 100
```

Q =

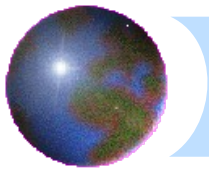
1 2 3
10 20 30
0 0 100

A = [A, [1;2;3]]

A =

2	0	1
0	1	2
3	3	3

When storing a value in an element outside the dimensions of the array, the dimensions automatically increase to accommodate the new element



Matrix Product, Element-wise operations

```
>> B = [ 1 2 ; 3 4]
```

```
B =
```

```
1 2  
3 4
```

```
>> C = B * B (equivalent to C = B
```

```
C =
```

```
7 10  
15 22
```

In summary...

$$B * B = B^2 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} * \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$B .* B = B.^2 = \begin{pmatrix} 1^2 & 2^2 \\ 3^2 & 4^2 \end{pmatrix}$$

```
>> D = B .* B (equivalent to D = B.^2) the dot denotes element-wise operations
```

```
D =
```

```
1 4  
9 16
```

What is the meaning of B / B ?
...and $B ./ B$?



Some useful functions



Some useful **mathematical functions**:

$\sin(x)$, $\cos(x)$, $\tan(x)$, $\text{atan}(x)$, $\exp(x)$, $\log(x)$, $\log_{10}(x)$, $\text{sqrt}(x)$

```
>> t = [ 0 : 0.01 : 10 ];
```

```
>> x = sin ( 2 * pi * t );
```

Compute **the size of matrix and vectors**:

```
>> size (A)
```

```
ans =
```

```
3    3
```

```
A = [ 1    2    3
```

```
4    5    6
```

```
7    8    9];
```

```
>> length ( t )
```

```
ans =
```

```
1001
```

BUT

```
>> size ( t )
```

```
ans
```

```
1    1001
```



Other operations and functions



For a vector x , $\text{sum}(x)$ is the sum of its elements. For a matrix A $\text{sum}(A)$ is a row vector containing the sum of each column.

```
>> sum ( A )  
ans =  
    12    15    18
```

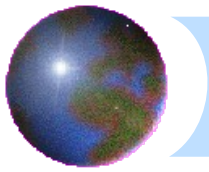
```
A =[1   2   3  
     4   5   6  
     7   8   9];
```

```
>> sum ( ans )    equivalent to sum(sum(A))  
ans =  
    45
```

```
>> A'    equivalent to transpose(A)  
ans =  
     1     4     7  
     2     5     8  
     3     6     9
```

```
>> diag(A)  
ans =  
     1  
     5  
     9
```

All Matlab functions work this way. Other examples: $\sin(x)$, $\cos(x)$, $\min(x)$, $\max(x)$, $\text{mean}(x)$,...



Output



- Function **fprintf**: useful to control output

`fprintf(format, args);` displays in the Command Window
`fprintf(fid, format, args);` writes on a file identified by `fid`

Examples:

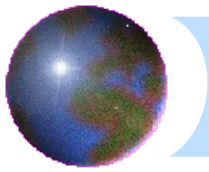
```
fprintf('%10.7f',var);  
fprintf('%10.7f %10.7f',var1,var2);  
fprintf('%d %20.16e %e \n',n,var1,var2);
```

Format legend:

`%d` integer format

`%f` fixed point format

`%e` scientific format (exponential notation)



Control structures

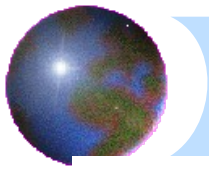


There are special language structures that allow for controlling the flow of execution (i.e. operations), that is, to perform a certain set of instructions even several times when a set of conditions occurs.

Control Structures:

- ☐ **if ... elseif ... else ... end**
- ☐ **switch ... case ... otherwise ... end**
- ☐ **for ... end**
- ☐ **while ... end**

- ☐ **break**
- ☐ **continue**



Selection structures



(a) Pseudocode

IF/THEN:

```
IF condition THEN
    True block
ENDIF
```

(b) MATLAB

```
if b ~= 0
    r1 = -c / b;
end
```

IF/THEN/ELSE:

```
IF condition THEN
    True block
ELSE
    False block
ENDIF
```

```
if a < 0
    b = sqrt(abs(a));
else
    b = sqrt(a);
end
```

The **if** statement evaluates a logical expression and executes a group of instructions when the expression is true. The **else** keyword executes a group of statements when the expression is false. The **end** keyword closes the **if**.



Multi-alternative structure



IF/THEN/ELSEIF:

IF condition₁ THEN

Block₁

ELSEIF condition₂

Block₂

ELSEIF condition₃

Block₃

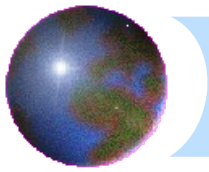
ELSE

Block₄

ENDIF

```
if class == 1
    x = x + 8;
elseif class < 1
    x = x - 8;
elseif class < 10
    x = x - 32;
else
    x = x - 64;
end
```

The **if** statement evaluates a logical expression and executes a group of instructions when the expression is true. The **elseif** keyword executes a group of statements when the successive expression is true.



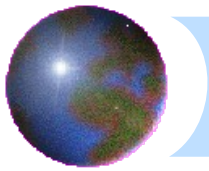
Switch and case



The **switch** statement executes groups of statements based on the value of a variable or expression. The **case** and **otherwise** keywords delimit the groups. There must always be an **end** to close the initial switch.

```
switch switch_expr
case case_expr
    instructions
case {case_expr1,
      case_expr2, ....}
    instructions
otherwise
    instructions
end
```

```
n = input('Enter a number: ');
switch n
    case -1
        disp('negative one')
    case 0
        disp('zero')
    case 1
        disp('positive one')
    otherwise
        disp('other value')
end
```



Switch-Case structure



SELECT CASE Test Expression

CASE Value₁

Block₁

CASE Value₂

Block₂

CASE Value₃

Block₃

CASE ELSE

Block₄

END SELECT

switch a + b

case 1

x = -25;

case 2

x = -5 - (a + b) / 10

case 3

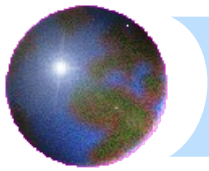
x = (a + b) / 10;

otherwise

x = 5;

end

Write the equivalent to if-elseif structure:



Count controlled loop (for-cycle)



COUNT-CONTROLLED LOOP:

DOFOR i = start, finish, step

Block

ENDDO

for i = start : step : finish

Block

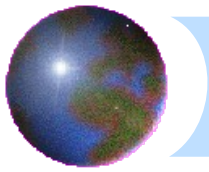
end

The **for** loop repeats a group of instructions for a predetermined number of times. An **end** (compulsory!) closes the instructions block.

Example:

Compute the sum of the
positive odd numbers <10

```
s=0;  
for i=1:2:10  
    s=s+i;  
end
```



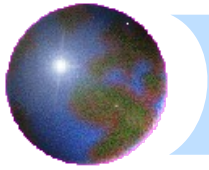
While ... end



The **while** loop repeats a set of statements an indefinite number of times under the control of a logical condition. A trailing **end** (compulsory!) delimits the loop's instructions.

General form

```
while condition  
    instructions  
end
```



Example



Determine the smallest integer n such that $n!$ (factorial) is a number greater than 100.

```
n=1;  
fatt=1;  
while fatt<=100  
    n=n+1;  
    fatt = fatt*n;  
end  
disp(n)
```



M-files (.py file equivalent)



- ❑ M-files are macros of MATLAB commands, they are stored as ordinary text files with the extension ".m", i.e. ***filename.m***
- ❑ There are two types of MATLAB m-files

script file

% script file

p=[1 3 2];

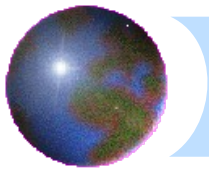
roots(p);

function file

% function file

function [y]=fun(x)

y=x^2+3*x^2+2;



Script vs function

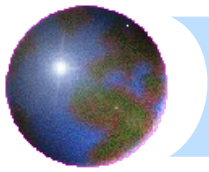


Script file

- List of MATLAB instructions
- Variables are global
- Can be run by typing the file name

Function file

- Start with the word function
- List of MATLAB instructions
- The variables are local
- Run by typing the file name and giving the parameters



Script file

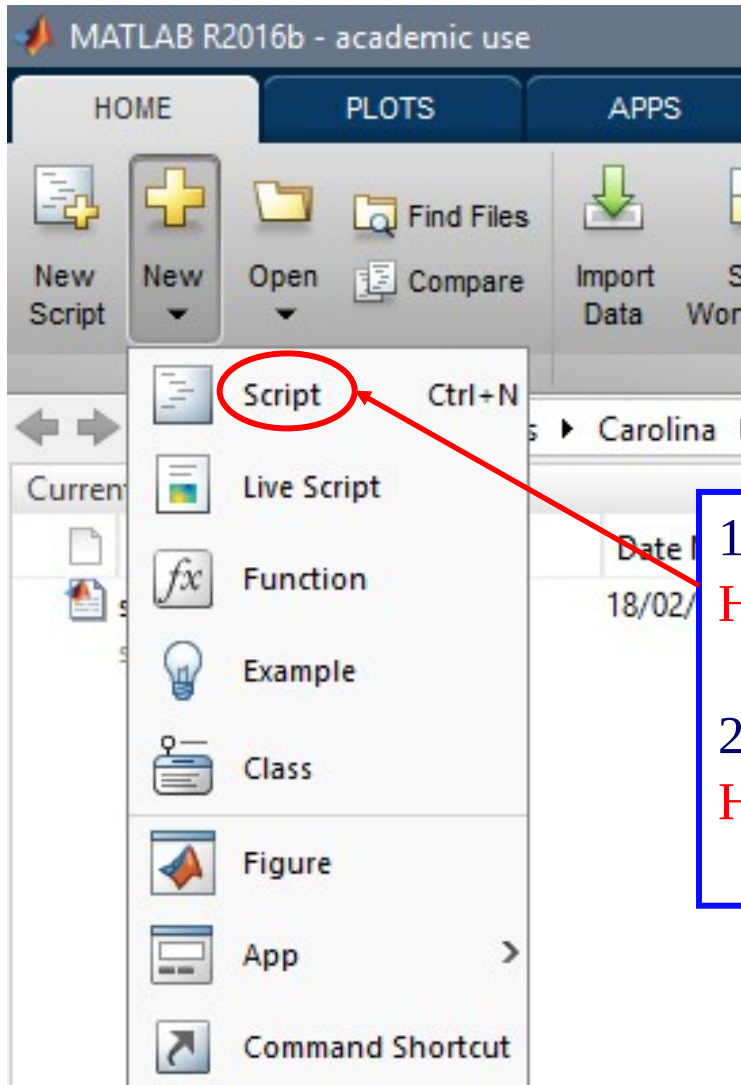


- ❑ Use a file script when you need a long sequence of instructions to solve a problem
- ❑ To run the program
 - type the name in the command window
 - from the Run menu in the Editor window





Create or open a script



1) Create a script

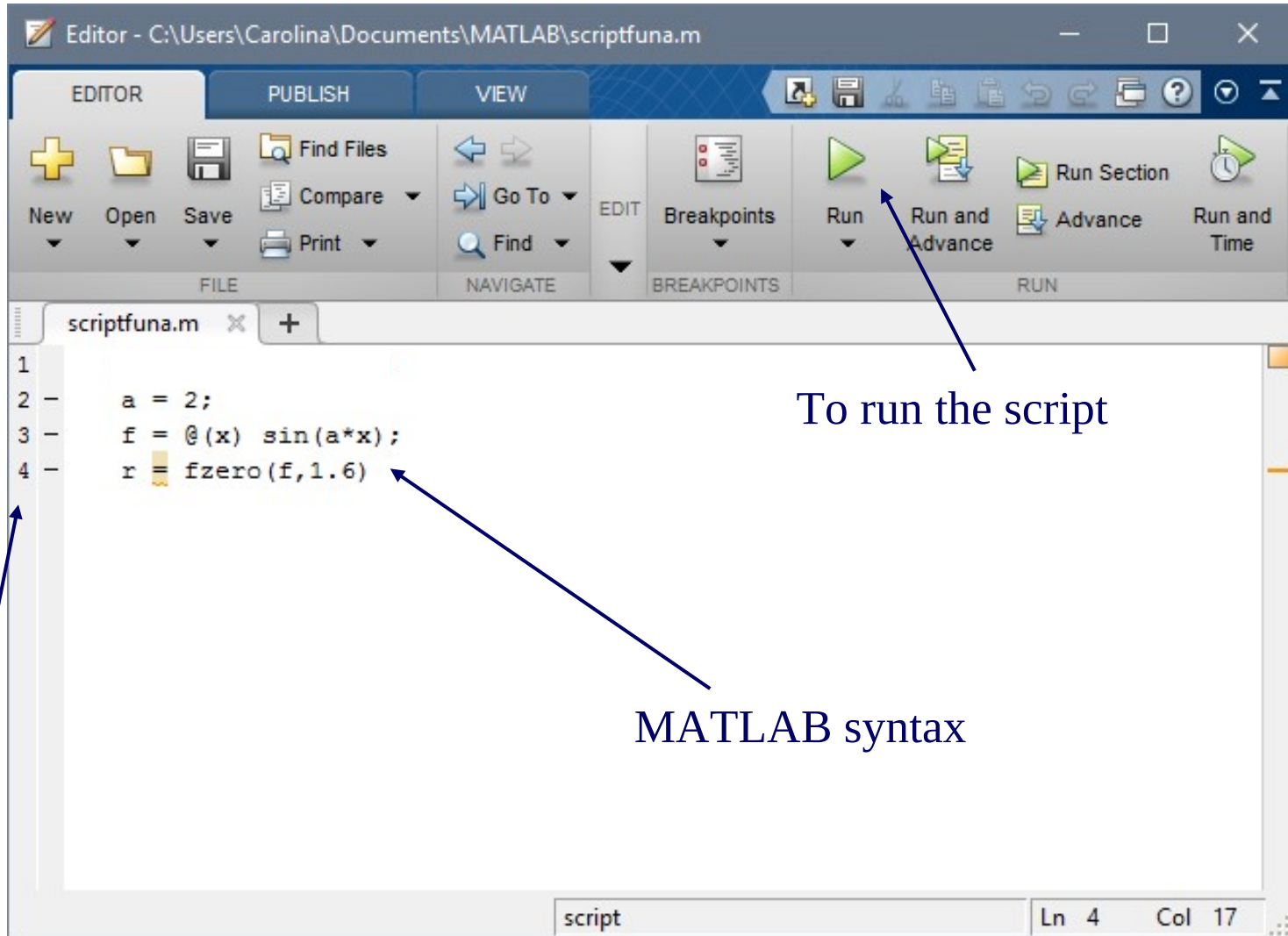
HOME → New → Script

2) Open an existing script

HOME → Open → select a script
This opens Matlab Editor



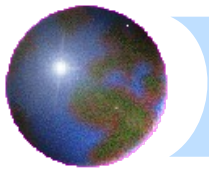
Editor and MATLAB code



Line numbers

To run the script

MATLAB syntax



Functions



□ Primary functions

- **Primary functions give name to the file**
- A number of secondary functions can be defined in the same file
- Primary functions can be called from any point (of a script or other function) while secondary functions can only be called by the corresponding primary function.



Example with secondary functions

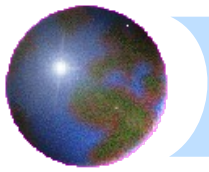


```
function [a,p] = polyGeom(s,n)
% polyGeom Computes area and perimeter of a regular polygon
% Synopsis: [a,p] = polyGeom(s,n)
%
% Input: s = length of one edge of the polygon
%        n = number of edges of the polygon
%
% Output: a = area of the polygon
%         p = perimeter of the polygon

r = s/(2*tan(pi/n)); % "radius" of the polygon
a = area(r,n);
p = perimeter(r,n);

% ===== secondary function "area"
function a = area(r,n)
% area computes the area of a polygon with n edges and radius r
a = n*r^2*sin(pi/n);

% ===== secondary function "perimeter"
function p = perimeter(r,n)
% perimeter computes the perimeter of a polygon with n edges and radius r
p = n*2*r*tan(pi/n);
```



Functions

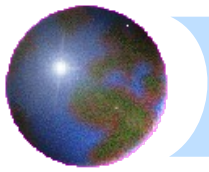


□ Syntax

`function [out1, out2, ...] = funname(in1, in2, ...)`

□ Description

- `function [out1, out2, ...] = funname(in1, in2, ...)`
defines function ***funname*** which takes as input variables ***in1***, ***in2***, ...etc. and returns as output variables ***out1***, ***out2***,... etc.



Function template



Comments are used to add explanations

% function example

output

Function name

input

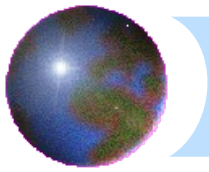
The first line of the file must start with 'function'

```
function [fc]=my_factorial (n)  
fc=factorial(n);
```

built-in function

EXTREMELY IMPORTANT:

The function name must be the same as the file name



Function Handle I



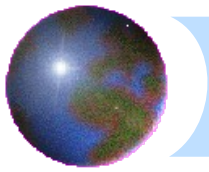
A **function handle** is a variable whose value is a function.

handle = @name where **name** is the name of an existing function

Once the handle is declared, one can invoke it as a function:

```
handle_sum = @sum;
```

```
handle_sum([3 2]);    %Computes the sum of the elements of vector [3 2]
```



Function Handle II



A function handle can also be declared "on the fly", that is, for a function that does not exist

```
handle = @(x)x*3
```

```
handle(10)
```

It is a handle of an "**anonymous**" function which accepts an input formal parameter x and performs with it a multiplication.

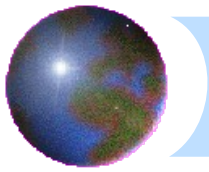
Note that $X * 3$ is the body of the

Being variables, handles can be passed to other functions:

```
handle_es = @(vect, func) func(vect)
```

```
handle_es([3 2], @sum)
```

```
handle_es([3 2], handle_sum)
```

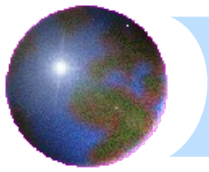


Example I



```
function s = fsum(fun,a,b,n)
% fsum  Calculates the sum of the values f(x) over n points in  a <= x <= b
%
% Synopsis:  s = fsum(fun,a,b,n)
%
% Input:  fun = handle to the function f(x) to be evaluated
%         a,b = endpoints of the interval
%         n   = number of points in the interval
%
% Output:  s = sum of f(x) for n points in the interval

x = linspace(a,b,n);    % generates n points in the interval
y = fun(x);             % evaluates the function at the given points
s = sum(y);              % computes the sum
```



Example II

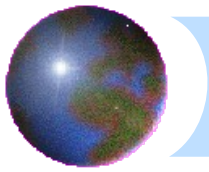


```
>> s=fsum(@sin,0,pi,15)
s =
    8.8752
```

```
>> s=fsum(@cos,0,pi,15)
s =
    6.6613e-016
```

```
function y=sincos(x)
y=sin(x).*cos(x);
```

```
>> s=fsum(@sincos,0,pi,15)
s =
    5.7142e-016
```



MATLAB Toolbox (Add-Ons)



- **Mapping Toolbox** provides tools and utilities for analyzing geographical data and creating maps.
- **MATLAB Compiler** allows to automatically convert MATLAB programs into standalone applications and software components to share them with end users. Applications and components created with the Compiler do not need MATLAB to run.
- **Partial Differential Equation (PDE) Toolbox** contains tools for the study and solution of partial differential equations (PDE) in two spatial dimensions (2-D) and over time. A set of command functions and a graphical user interface allows for preprocessing, solving, and post processing PDEs for a wide range of technical and scientific applications.
- **Symbolic/Extended Math Toolbox** provides tools for solving and manipulating mathematical expressions.
- **Image Processing Toolbox** provides a complete set of standard algorithms and graphical tools for image processing, analysis, visualization and development of new algorithms. You can improve image quality with noise or degradation, extract features, analyze shapes and textures, and perform image registration.



Tutorials and References I



MATHWORKS web site:

<http://www.mathworks.com/help/matlab/index.html>

The screenshot shows the MathWorks website documentation page for MATLAB R2017b. The page has a blue header with the MathWorks logo and navigation links: Products, Solutions, Academia, Support (highlighted), Community, and Events. Below the header is a search bar labeled 'Search R2017b Documentation' and a 'Documentation' dropdown menu. A left sidebar contains a 'CONTENTS' menu with links to 'Documentation Home' and 'MATLAB'. The main content area is titled 'MATLAB' and 'The Language of Technical Computing'. It includes a paragraph about MATLAB's use by engineers and scientists, a section for 'Getting Started' (Learn the basics of MATLAB), and a section for 'Language Fundamentals' (Syntax, operators, data types, array indexing and manipulation). On the right side of the main content, there are links for 'Examples', 'Functions', 'Release Notes', and 'PDF Documentation'. The top right corner of the page has links for 'Contact Us', 'How to Buy', and 'Log In'.

Tutorial: MATLAB primer

<http://math.ucsd.edu/~driver/21d-s99/matlab-primer.html>



Tutorials and References II



Book:

**Attaway Stormy – MATLAB: A Practical
Introduction to Programming and Problem Solving**