Principi di Bioingegneria
A.A. 2023/24
Lezione 1

Introduzione al corso
Introduzione a MATLAB

Vincenzo Catrambone, PhD

vincenzo.catrambone@unipi.it

UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Research Center E. Piaggio
University of Pisa

Docenti del corso

- Vincenzo Catrambone

vincenzo.catrambone@unipi.it

- Gabriele Maria Fortunato

gabriele.fortunato@unipi.it

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Research Center E. Piaggio
University of Pisa

Ricevimento su appuntamento, in presenza o su piattaforma Microsoft Teams

Obiettivi del corso

Obiettivo del corso è mostrare, attraverso esempi e mediante l'uso di supporti informatici quali Matlab e Simulink, come gli strumenti metodologici della Bioingegneria possano essere impiegati per comprendere, valutare funzionalmente e riprodurre sistemi biologici o parti di essi.

Outline del corso

- Utilizzo del software Matlab
- Introduzione alla bioingegneria e ruolo dell'ingegnere biomedico
- Cenni di biochimica (biomolecole, cellule e tessuti)
- Proprietà dei materiali e dei tessuti biologici
  - Proprietà meccaniche, elettriche, ottiche, elettromagnetiche
- Principi di misure, errori di misura, unità di misura e analisi dimensionale
- Acquisizione e analisi di biosegnali
- Acquisizione e analisi di bioimmagini
- Modellazione di processi e strutture fisiologiche

Orario delle lezioni

- Giovedì 10:30-13:30
  - Aula A2.6
- Venerdì 8:30-10:30
  - Aula Si5

Modalità d’esame

- Prova scritta
  - Quesiti a risposta multipla + risposta aperta
- Prova orale
  - Consegna esercitazioni Matlab svolte durante il corso con domande sul software
  - Domande sul programma del corso

Materiale del corso

Slide + materiale aggiuntivo

https://www.centropiaggio.unipi.it/course/principi-di-bioingegneria

Sito del Centro di Ricerca Piaggio

https://unipiit.sharepoint.com/:u:/r/sites/a__td_58614/SitePages/ClassHome.aspx?csf=1&web=1&share=EStT8mUEi2JGmaWhR7FQj5EBmRYhg4ETwsrRjjRiHxfusQ&e=eqqFfb

Canale Teams

Cos’è MATLAB

Installazione e supporto

Guida all’ambiente MATLAB

Tutorial

Esercizi

Cos’è MATLAB

MATLAB (abbreviazione di MATrix LABoratory) è un ambiente per il calcolo e il relativo linguaggio di programmazione (di alto livello). E’ stato inventato intorno agli anni ’70 e successivamente è stata creata la società MathWorks che ne è proprietaria.

MATLAB è un software proprietario, e con licenza a pagamento. E’ uno dei più diffusi in ambito scientifico e ingegneristico, sicuramente il più diffuso nelle università. Esistono software open source compatibili:

Octave è sostanzialmente sempre compatibile, gli altri possono avere piccole incompatibilità

Cos’è MATLAB

Esistono anche alcune librerie che aggiungono funzionalità simili a quelle di MATLAB ad altri linguaggi esistenti, come:

- *NumPy*, *SciPy* e *Matplotlib* per *Python*
- *IT++* per *C++*
- *Numeric.js* per *JavaScript*
- *SciLua* e *Torch* per *Lua*
- *Perl Data Language* per *Perl*
- *ILNumerics* per *.NET*
- *SciRuby* per *Ruby*

Disclaimer: La parte ‘pratica’ di questo corso sarà incentrata su MATLAB, perché il più diffuso nei diversi ambiti disciplinari/scientifici, non perché sia il migliore o l’unico.

Perché MATLAB

MATLAB consente di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi, creare interfacce utente, e interfacciarsi con altri programmi.

MATLAB è usato da milioni di persone nell'industria e nelle università per via dei suoi numerosi strumenti a supporto dei più disparati campi di studio applicati e funziona su tutti i più diffusi sistemi operativi.

Contiene:

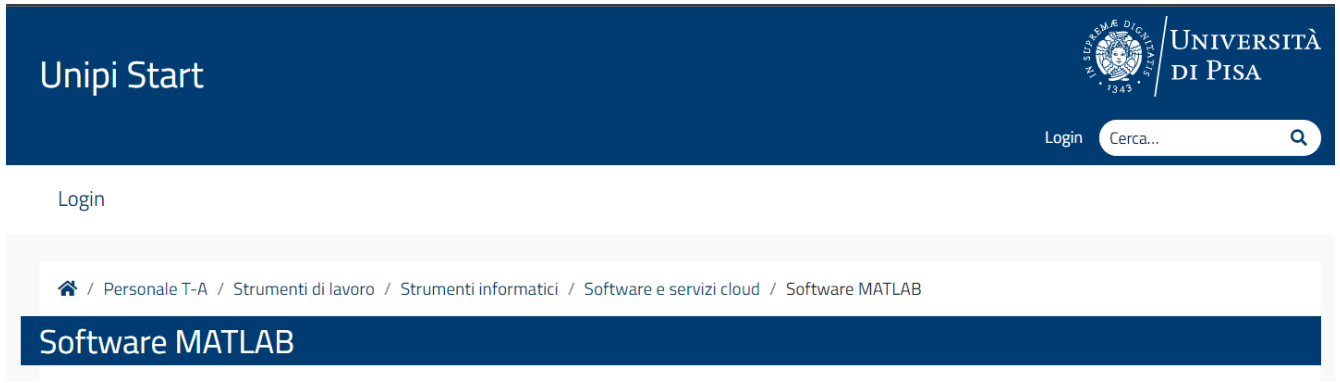
- Un vasto set di funzioni di base general purpose;
- Possibilità di definire funzioni hand-made;
- Estensioni native application oriented (apps e toolboxes di MATLAB stesso);
- Estensioni esterne (toolboxes di altri);
- Un'interfaccia grafica interattiva per la modellazione e la simulazione (Simulink);
- Una foltissima community online (MATLAB Central, GitHub, Stack Overflow, ecc.).

Istallazione e supporto

Come anticipato, MATLAB è un software proprietario e a pagamento, UNIPI come molte altre università ha delle convenzioni per cui MATLAB è liberamente accessibile a TUTTI gli utenti UNIPI (fino a 4 PC collegati alla stessa licenza)

Andare ai link:
https://start.unipi.it/personale-t-a/strumenti-di-lavoro/strumenti-informatici/software-e-servizi-cloud/software-matlab/

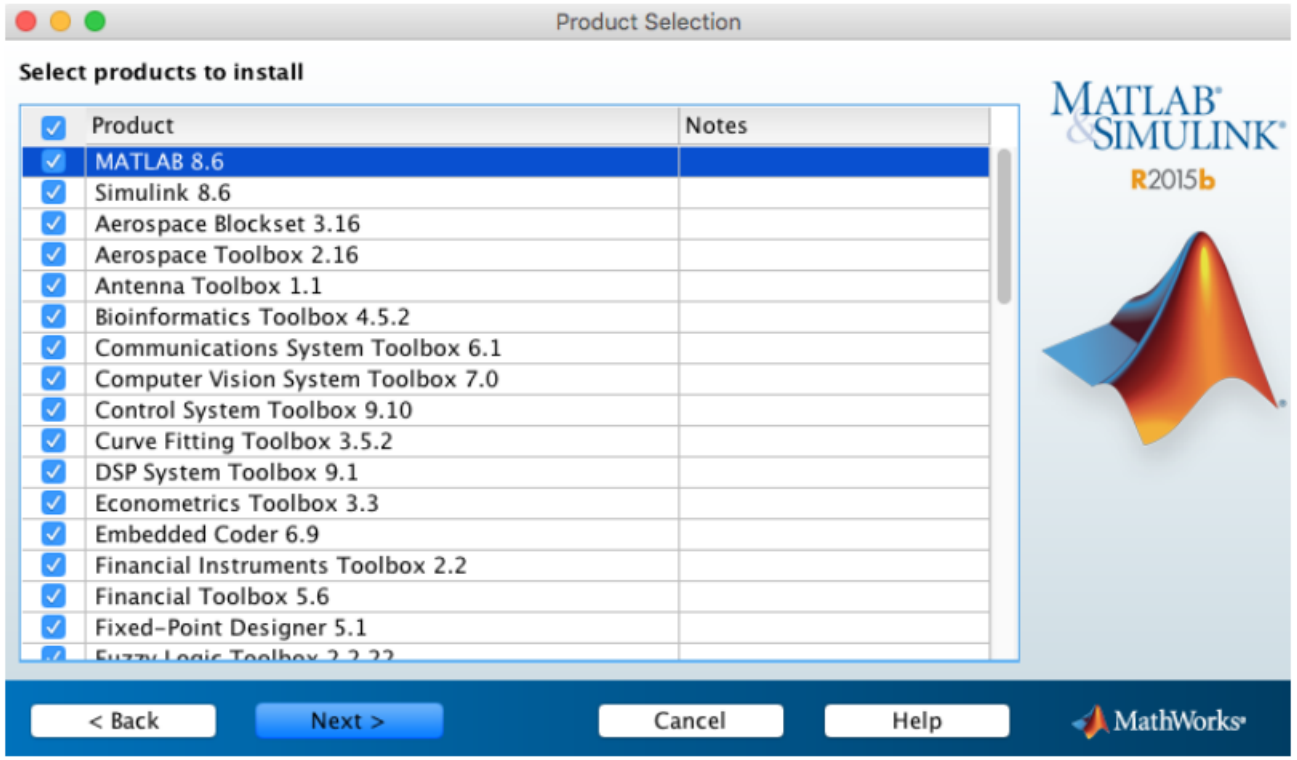
https://start.unipi.it/wp-content/uploads/2022/08/Istruzioni_Installazione_MatLab_Student_2017.pdf



Istallazione e supporto

A un certo punto (6. nella guida) sarà chiesto di selezionare i toolboxes da installare. Sono tutti gratuiti, ma potenzialmente pesanti in termini di memoria.

6. Selezionare i prodotti (toolbox) che si desiderano installare e premere "Next" per continuare (si consiglia di non installare i toolbox non utili per la propria attività di studio).



Al link (https://it.mathworks.com/products.html) la lista completa. Si consigliano, per le esercitazioni future di questo corso:

- Statistics and Machine Learning Toolbox
- Curve Fitting Toolbox
- Signal Processing Toolbox
- Image Processing Toolbox

Comunque possono essere aggiunti liberamente in ogni momento.

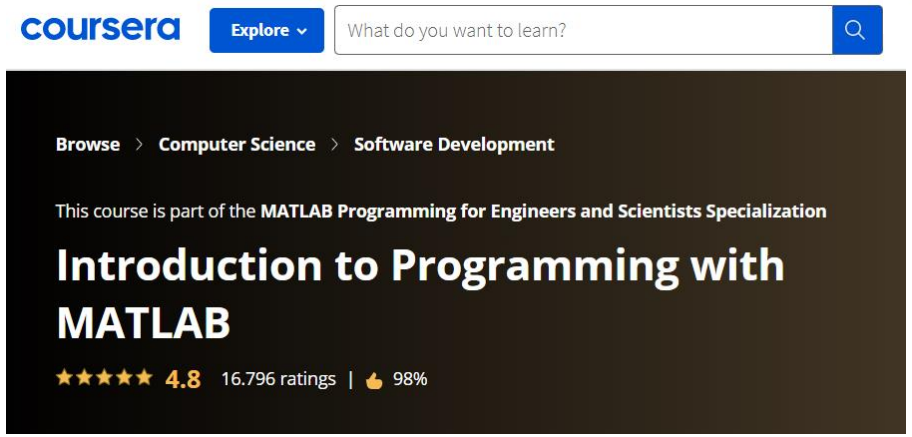
Oltre l'istallazione

Libro free*: https://www.sciencedirect.com/book/9780123850812/matlab

Sito Mathworks:
https://it.mathworks.com/academia/students.html

*Ci sono edizioni più aggiornate del libro, a pagamento. La struttura di MATLAB non è cambiata, info su singole funzioni che sono state aggiornate si trovano facilmente

Altri online tutorials

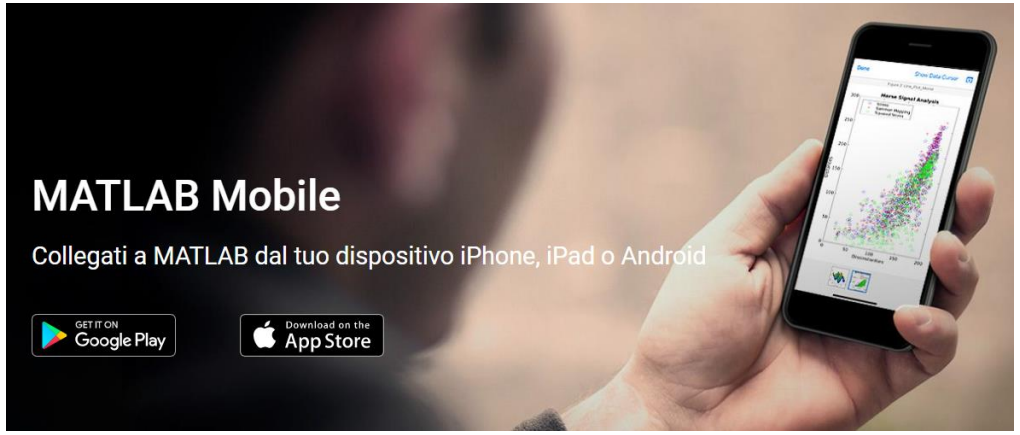


Supporto online

MATLAB® Central

MATLAB Answers | File Exchange | Cody | Blogs | Participate | Communities | More

stackoverflow



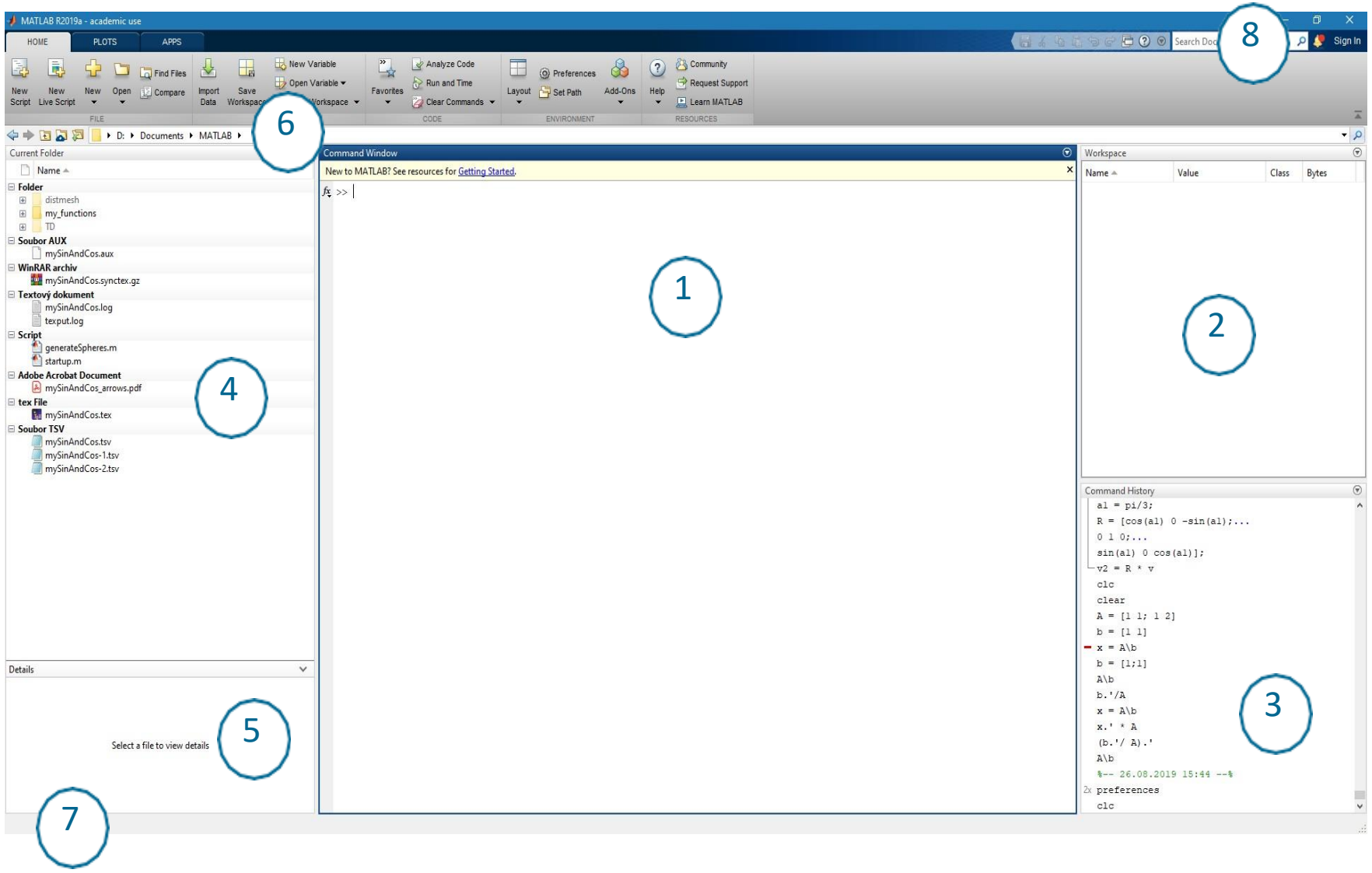
GitHub

Quora

DISCORD

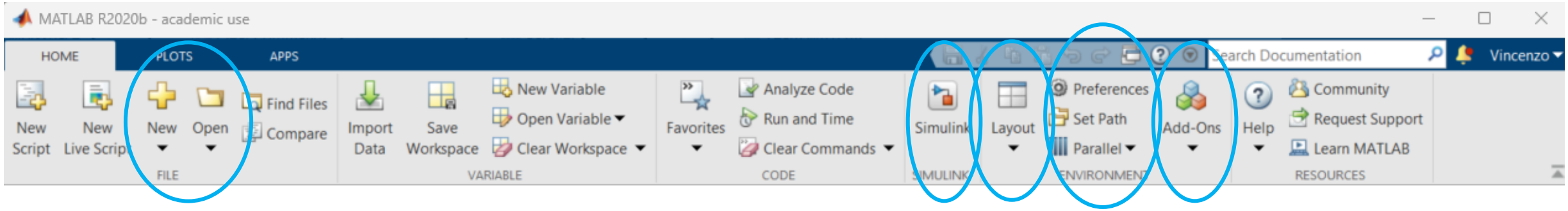
reddit

Guida all'ambiente MATLAB

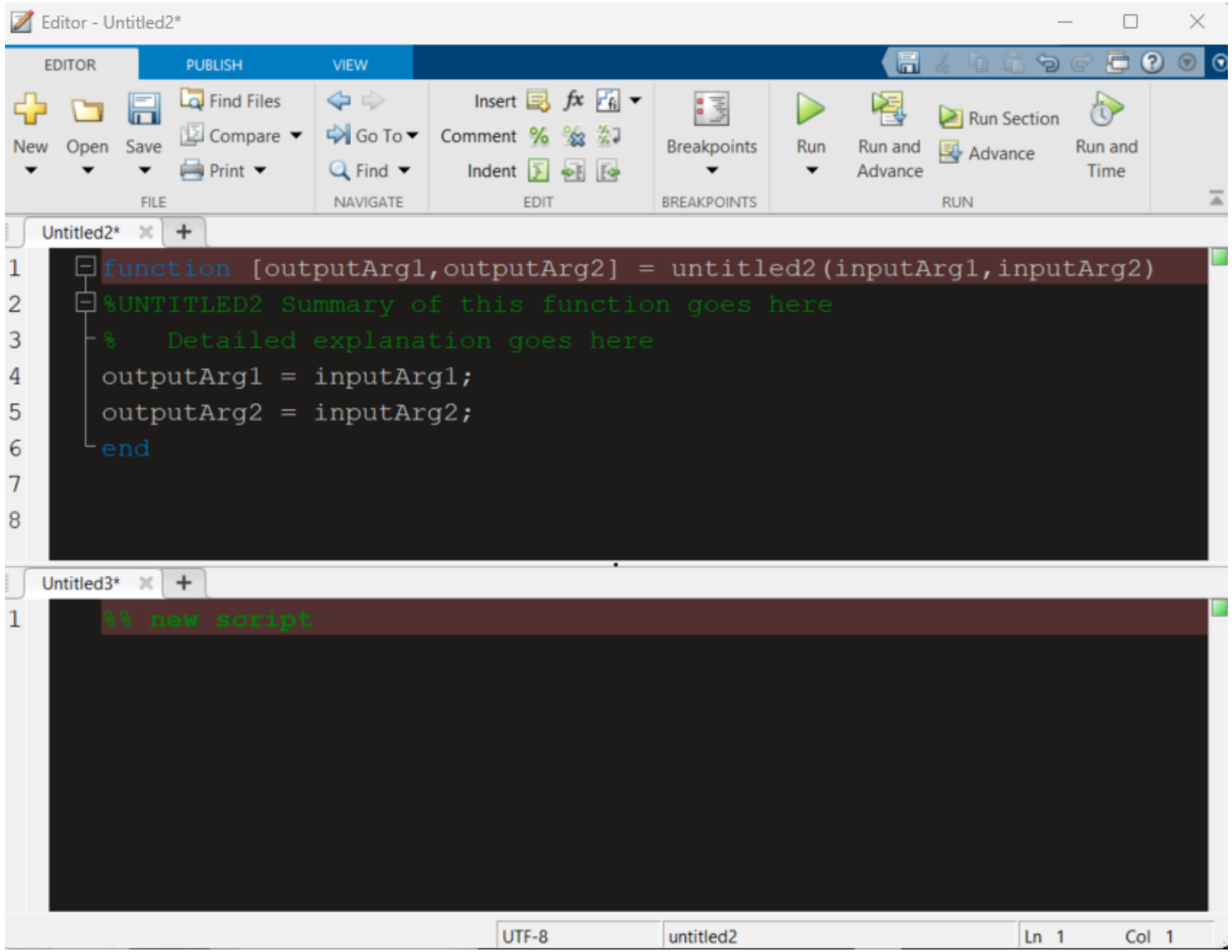


1. Command Window
2. Workspace
3. Command History – not activated, to activate →
4. Current Folder
5. Current Folder – Details
6. Current Working Directory
7. Status ("Ready" or "Busy")
8. Search in documentation

Guida all'ambiente MATLAB



Guida all'ambiente MATLAB



Guida all'ambiente MATLAB

- Command:

```
>>help %MATLAB help
```

- Command:

```
>>doc function_name
```

- Right click on function name, click on 'Open selection'

Tutorial

Commands	Purpose
help	lists topics on which help is available
helpwin	opens the interactive help window
helpdesk	Opens the web-browser-based help facility
demo	runs the demo program
doc	Online HTML documentation
info	Info about MATLAB
why	Give a philosophical device
home	Send cursor home
↑ ↓	Recall previous commands

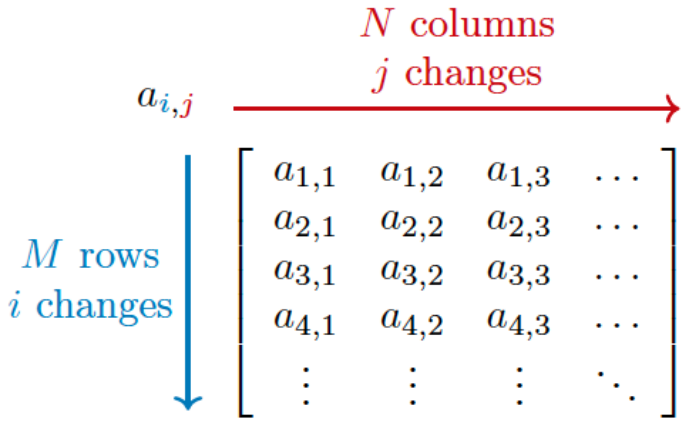
Useful commands

Commands	Purpose
who	lists variables currently in the workspace
whos	lists variables directly in the workspace with their size
clear var/clear all/clearvars	clear single var/ all variables/ the workspace
clc	clears command window, cursor shift to the top
close all/close fig	Close all open windows/ single fig
gcf	Get current figure
tic/toc	Start stopwatch timer/read stopwatch timer
clock/date	Wall clock time/ Date, month, year
disp	Display text or matrix
dir	lists contents of the current directory
cd	changes the current working directory
mkdir	creates a new directory
exit/quit	Quit MATLAB
addpath()	Add path to the current search path

Tutorial

- Matrix is a basic data structure in Matlab.
- There are following variables types depending on size:
  - single element: 1×1
  - vector: $M \times 1$ or $1 \times N$
  - matrix: $M \times N$
  - array (multidimensional matrices):

    $M \times N \times P \times Q \times R \times \ldots$
  - Matrices can be complex.
  - It can contains text as well (beware the length).
- M -by-N matrix:



This hold for every variable type

Data type format

- **Numbers**:
  - single (32bit floating point)
  - double (64bit floating point)
  - int8 (intero 8 bit)
  - int16 (intero 16 bit)
  - int32 (intero 32 bit)
  - int64 (intero 64 bit)
  - uint8 (intero senza segno 8 bit)
  - uint16 (intero senza segno 16 bit)
  - uint32 (intero senza segno 32 bit)
  - uint64 (intero senza segno 64 bit)
- **Logical** (1 bit – 1 true, 0 false)
- **Char** (2-byte Unicode)

Signed 8-bit integer	-2^7 to 2^7-1
Signed 16-bit integer	-2^{15} to $2^{15}-1$
Signed 32-bit integer	-2^{31} to $2^{31}-1$
Signed 64-bit integer	-2^{63} to $2^{63}-1$
Unsigned 8-bit integer	0 to 2^8-1
Unsigned 16-bit integer	0 to $2^{16}-1$
Unsigned 32-bit integer	0 to $2^{32}-1$
Unsigned 64-bit integer	0 to $2^{64}-1$

Operands, constants, and precedences

Symbol	Role	Corresponding function
+	Addition	plus
+	Unary plus	uplus
-	Subtraction	minus
-	Unary minus	uminus
.*	Element-wise multiplication	times
*	Matrix multiplication	mtimes
./	Element-wise right division	rdivide
.\	Element-wise left division	ldivide
/	Matrix right division	mrdivide
\	Matrix left division	mldivide
.^	Element-wise power	power
^	Matrix power	mpower
.'	Transpose	transpose
'	Complex conjugate transpose	ctranspose

pi	3.14159. . . .
i	$\sqrt{-1}$
j	$\sqrt{-1}$
inf	infinity ∞
NaN	stands for "not a number," such as the result of 0/0

( )	parentheses
^	exponentiation
*, /, \	all multiplication and division
+, −	addition and subtraction

```
>> 4 + 5 * 3
ans =
    19

>> (4 + 5) * 3
ans =
    27
```

scientific notation

```
>> 2 * 10^4
ans =
       20000

>> 2e4
ans =
       20000
```

Think about what the results would be for the following expressions, and then type them in to verify your answers:

4 ^ 2 - 1
4 ^ (2 - 1)
2 \ 3
4 * 2 - 9/3
5 - - 3

Variable definition and assignment

```
>> v = [1    2    3    4]
v =
   1   2   3   4

>> v = [1,2,3,4]
v =
   1   2   3   4

>> vec = 1:5
vec =
     1   2   3   4   5

>> nv = 1:2:9
nv =
     1   3   5   7   9
```

```
>> ls = linspace(3,15,5)
ls =
     3     6     9    12    15
```

concatenating

```
>> newvec = [nv ls]
newvec =
   1   3   5   7   9   3   6   9   12   15
```

```
>> newvec(5)
ans =
     9

>> b = newvec(4:6)
b =
     7     9     3
```

What happens if adding the step value would go beyond the range specified by the last?
Es: v = 1 : 2 : 6

How can you use the colon operator to generate the following vector v?
v = [9 7 5 3 1]

Variable definition and assignment

```
>> rv = [3  55  11]
rv =
     3     55     11

>> rv(4) = 2
rv =
     3     55     11      2

>> rv(6) = 13
rv =
     3     55     11      2      0     13
```

Think about what would be produced by the following sequence of statements and expressions, and then type them in to verify your answers:

```
pv = 2:2:8
pv(4) = 33
pv(6) = 11
prac = pv(3:5)
linspace(4,12,3)
prac(end)
```

Variable definition and assignment

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

```
>> r = 1:3;
>> c = r'
c =
     1
     2
     3
```

```
>> mat = [4  3  1;  2  5  6]
mat =
     4     3     1
     2     5     6
```

```
>> mat = [2:4; 3:5]
mat =
     2     3     4
     3     4     5

>> mat(2,3)
ans =
     5

>> mat(1,:)
ans =
     2     3     4
```

```
>> vec = 1:8
vec =
     1     2     3     4     5     6     7     8

>> vec(2:4) = []
vec =
     1     5     6     7     8
```

```
>> mat = [7 9 8; 4 6 5]
mat =
     7     9     8
     4     6     5

>> mat(1,2) = [];
??? Indexed empty matrix assignment is not allowed.

>> mat(:,2) = []
mat =
     7     8
     4     5
```

```
>> mat(4,:) = 2:2:8
mat =
     2    11     4     9
     5     6     7     2
     0     0     0     0
     2     4     6     8
```

Variable definition and assignment

Symbol	Role	Corresponding function
==	Equal to	eq
~=	Not equal to	ne
>	Greater than	gt
>=	Greater than or equal to	ge
<	Less than lt	
<=	Less than or equal to	le

```
A = [2 7 6; 9 0 5; 3 0.5 6];
B = [8 7 0; 3 2 5; 4 -1 7];
A == B
ans =
     0   1   0
     0   0   1
     0   0   0
```

```
A = [0 1 1 0 1];
B = [1 1 0 0 1];
```

Symbol	Role	Description
&	Logical AND It returns 1 for every element location that is true (nonzero) in both arrays and 0 for all other elements.	A & B = 01001
|	Logical OR It returns 1 for every element location that is true (nonzero) in either one or the other, or both arrays, and 0 for all other elements.	A | B = 11101
~	Logical NOT It complements each element of the input array, A.	~A = 10010
xor	It returns 1 for every element location that is true (nonzero) in only one array, and 0 for all other elements.	xor(A,B)=10100

Built-in functions

sin()
cos()
tan()
atan()
…

rem()
sign()
exp()
min()
max()
sqrt()
det()
diag()
…

abs()
angle()
real()
imag()
conj()
complex()
isreal()
isimag()
…

length()
size()
numel()
…

repmat()
reshape()
fliplr()
flipud()
rot90()
…

linspace()
zeros()
ones()
rand()
randi()
eye()
…

num2str()
str2mat()
int2str()
mat2str()
…

logical()
double()
single()
uint16()
string()
isdouble()
ischar()
…

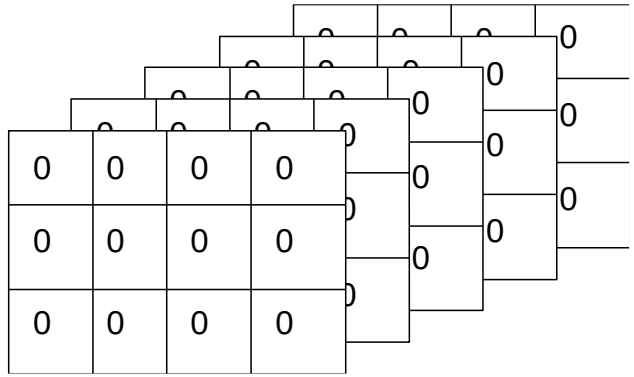
fix()
floor()
ceil()
round()
…

```
>> help sin
 sin     Sine of argument in radians.
    sin(X) is the sine of the elements of X.

    See also asin, sind, sinpi.

    Documentation for sin
    Other functions named sin
```

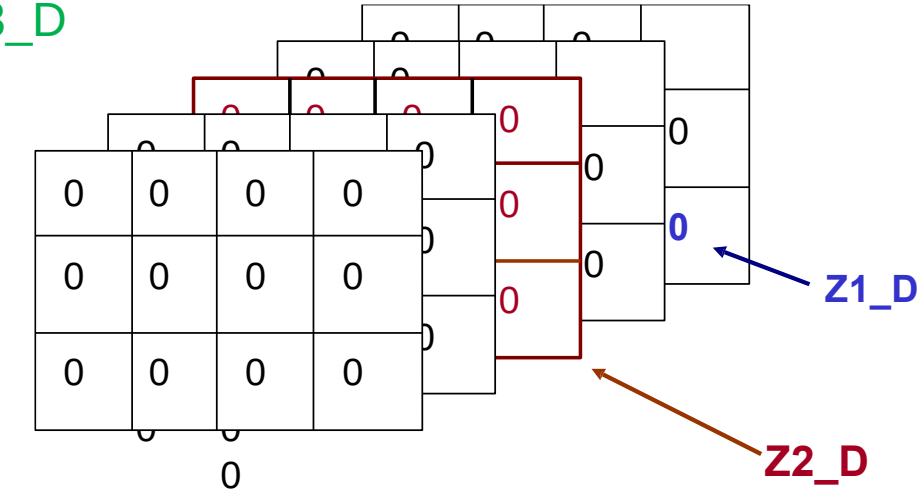
Variable definition and assignment

Come si diceva in precedenza, MATLAB è pensato per lavorare con array multidimensionali. Es:

```
>> Z3_D = zeros(3,4,5); % Z3_D è una matrice tri-dimensionale, 3x4x5  i cui elementi valgono 0
```



```
>> Z2_D = Z3_D(:,:,3); % Z2_D è una matrice bi-dimensionale 3x4 ottenuta da Z3_D
>> Z1_D = Z3_D(end,end,end); % Z1_D è uno scalare ottenuto da Z3_D
```



Working with strings

Strings in Matlab can be represented in two forms:

As a vector of characters which are represented as `char` data type.
It is created using apostrophes:

```
>> st1 = 'Hello world!';
```

As `string` data type.
It is created using double quotes:

```
>> st2 = "Hello world!";
```

```
 Name      Size                Bytes  Class     Attributes

  st1       1x12                  24  char
  st2       1x1                  166  string
```

If an apostrophe is required to be part of a string, it is to be typed as two quote characters:

```
>> st3 = 'That''s it!'
```

Distinguish between:
"string" in meaning of text and
"`string`" as data type.

In the case of more lines of characters, it must have same number of columns:

```
>> st4=['george';'pepi ']; size(st4)% [2, 6]
```

Unlike `char`, `string` does not treat numbers as ASCI or Unicode.

```
>>'a'+1
ans=
98
```

```
>>"a"+1
ans=
 "a1"
```

Working with strings

Conversion of number represented as a string (`char`) to number (`double`):
Conversion of multiple numbers (function `str2num`):

```
>> str2num('[1 2 3 pi]')
 ans =
   1.0000 2.0000 3.0000 3.1416
>> str2num('[1, 2; 3 4]')
 ans =
   1 2
   3 4
```

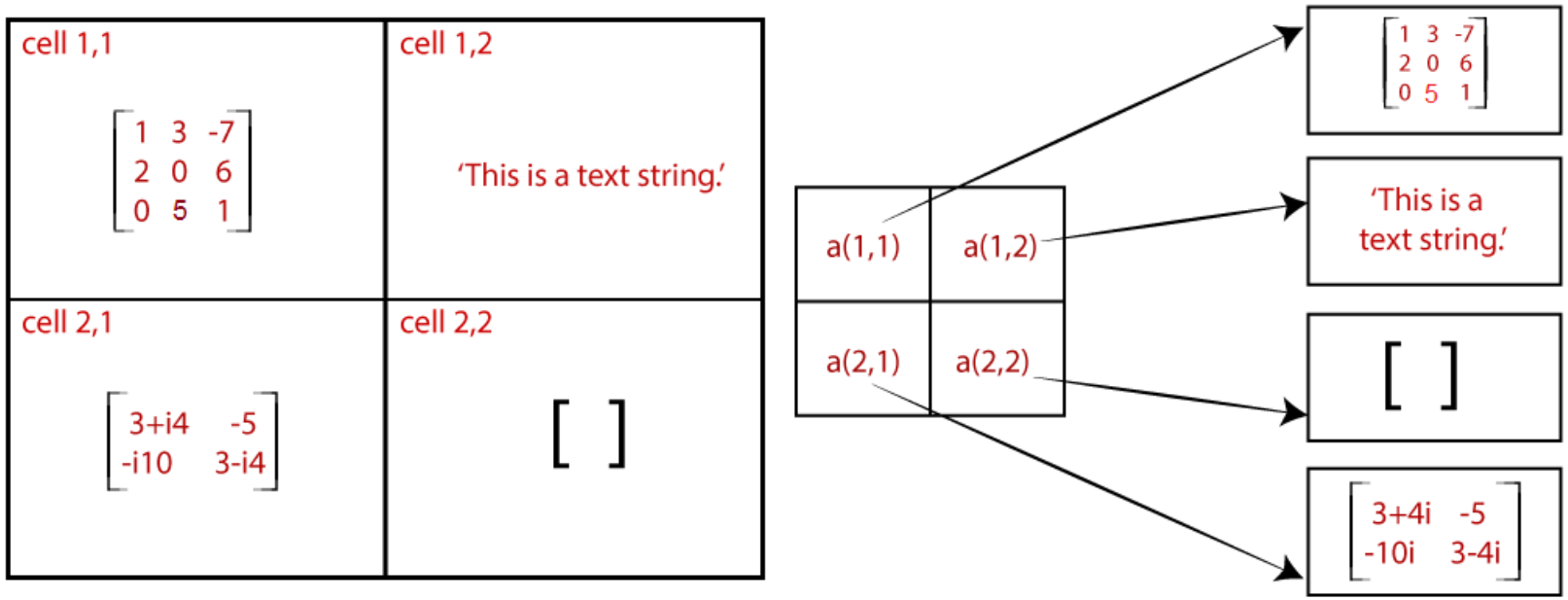
```
>> num2str(pi);% '3.1416'
>> num2str(pi,10);% '3.141592654'
>> string(pi);% "3.1416"
>> disp(['The value of pi is: 'num2str(pi,5)]);
>> st=sprintf('The value of pi is: %0.5f',pi);
>> contains(text,string)
```

Possible errors:

```
>> str2num('1a')
 ans =
  []
>> str2double('[1 2 3 pi]')
 ans =
  NaN
>>str2num('1+1j')
 ans =
  1.0000+1.0000i
>> str2num('1 +1j')
 ans =
  1.0000+0.0000i   0.0000+1.0000i
```

Cells and cell array

A cell is the functional data object in MATLAB. It can contain any data, an array of numbers, strings, structures, or cells. An array of cells is called a **cell array**.
For example, one cell of a cell array contains an array of real numbers, another an array of strings, and yet another a vector of complex numbers.



Cells and cell array

```
>> cellmat = {23 'a'; 1:2:9 'hello'}
cellmat =
    [      23]      'a'
    [1 x 5 double]    'hello'


>> cellmat(2,1)
ans =
   [1 x 5 double]

>> cellmat{2,1}
ans =
    1     3     5     7     9

>> cellmat{2,1}(4)
ans =
  7
```

```
>> mycellmat = cell(2,2)
mycellmat =
     []    []
     []    []

>> mycellmat{1,1} = 23
mycellmat =
    [23]     []
     []      []
```

```
>> names = {'Sue', 'Cathy', 'Xavier'}
names =

    'Sue'      'Cathy'      'Xavier'
```

```
>> iscellstr( cellmat)
ans =
     0
```

```
>> iscellstr(names)
ans =
     1
```

Structures

A structure is a data type in which each individual element has a name.
The individual elements of a structure are known as **fields**, and each field in a structure may have a different type.
The individual fields are addressed by combining the name of the structure with the name of the field, separated by a period.
Data in a field is accessed using dot notation.

```
>> x.name = 'data';
>> x.role = 'valuation';
>> x.strength = 10

x =

  struct with fields:

        name: 'data'
        role: 'valuation'
    strength: 10

>> whos x
  Name      Size            Bytes  Class     Attributes

  x         1x1               562  struct
```

```
>> x(2).name = 'datafield';
>> x(2).role = 'userdefined';
>> x(2).strength = 7

x =

  1×2 struct array with fields:

    name
    role
    strength
```

One thing to remember here is that all the field's names should be the same while adding more elements.

Structures

```
>> packages(1) = struct('item_no',123,'cost',19.99,...
     'price',39.95,'code','g');
>> packages(2) = struct('item_no',456,'cost', 5.99,...
     'price',49.99,'code','l');
>> packages(3) = struct('item_no',587,'cost',11.11,...
     'price',33.33,'code','w');
```

```
>> packages = repmat(struct('item_no',123,'cost',19.99,...
     'price',39.95,'code','g'),1,3);
>> packages(2) = struct('item_no',456,'cost', 5.99,...
     'price',49.99,'code','l');
>> packages(3) = struct('item_no',587,'cost',11.11,...
     'price',33.33,'code','w');
```

```
>> packages
packages =
1 x 3 struct array with fields:
    item_no
    cost
    price
    code
```

```
>> isstruct(package)
ans =
     1
```

```
>> isfield(package,'cost')
ans =
     1
```

```
>> pack_fields = fieldnames(package)
pack_fields =
    'item_no'
    'cost'
    'price'
    'code'
```

Tra le altre, ci sono anche le funzioni:
getfield(), setfield(), rmfield()

Nested variables

Si possono creare, naturalmente, variabili concatenate, es: dei campi di strutture che sono altre strutture, o cell array, e viceversa, dei cell array con all'interno strutture innestate

```
>> cyls(3) = struct('code', 'c', 'dimensions',...
     struct('rad', 3, 'height', 6), 'weight', 9);
>> cyls(1) = struct('code', 'x', 'dimensions',...
     struct('rad', 3, 'height', 6), 'weight', 7);
>> cyls(2) = struct('code', 'a', 'dimensions',...
     struct('rad', 4, 'height', 2), 'weight', 5);
```

- `cyls` is the entire data structure, which is a vector of structs
- `cyls(1)` is an individual element from the vector, which is a struct
- `cyls(2).code` is the *code* field from the struct *cyls(2)*; it is a character
- `cyls(3).dimensions` is the *dimensions* field from the struct *cyls(3)*; it is a struct itself
- `cyls(1).dimensions.rad` is the *rad* field that is from the struct *cyls(1).dimensions*; it is a **double** number

Esercizi

1. Create a variable to store the atomic weight of silicon (28.085).
2. Create a variable *myage* and store your age in it. Subtract one from the value of the variable. Add 2 to the value of the variable.
3. Explore the format command in more detail. Use help format to find options. Experiment with format bank to display dollar values.
4. Find a format option that would result in the following output format:

```
>> 5/16 + 2/7
ans =
  67/112
```

```
25 / 4* 4
3 + 4 ^ 2
4 \ 12 + 4
3 ^ 2
(5 - 2) * 3
```

5. Think about what the results would be for the following expressions, and then type them in to verify your answers.

$$R_T = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$

6. The combined resistance RT of three resistors R1, R2, and R3 in parallel is given by
   Create variables for the three resistors and store values in each, and then calculate the combined resistance.
7. Create a variable pounds to store weight in pounds. Convert this to kilograms and assign the result to a variable kilos. The conversion factor is 1 kilogram = 2.2 pounds.
8. Create a variable ftemp to store a temperature in degrees Fahrenheit (F). Convert this to degrees Celsius (C) and store the result in a variable ctemp. The conversion factor is C = (F – 32) * 5/9.
9. The function sin calculates and returns the sine of an angle in radians. Use help elfun to find the name of the function that returns the sine of an angle in degrees. Verify that calling this function and passing 90 degrees to it results in 1.

Esercizi

10. A vector can be represented by its rectangular coordinates x and y or by its polar coordinates r and y. The relationship between them is given by the following equations. Assign values for the polar coordinates to variables r and theta. Then, using these values, assign the corresponding rectangular coordinates to variables x and y.

$$x = r * \cos(\theta)$$
$$y = r * \sin(\theta)$$

11. Wind often makes the air feel even colder than it is. The wind chill factor (WCF) measures how cold it feels with a given air temperature T (in degrees Fahrenheit) and wind speed (V, in miles per hour). One formula for the WCF is:

$$\mathrm{WCF} = 35.7 + 0.6\ \mathrm{T} - 35.7(\mathrm{V}^{0.16}) + 0.43\ \mathrm{T}(\mathrm{V}^{0.16})$$

Create variables for the temperature T and wind speed V, and then using this formula calculate the WCF.

12. Experiment to answer the following questions:
    - Is fix(3.5) the same as floor(3.5)?
    - Is fix(3.4) the same as fix(3.4)?
    - Is fix(3.2) the same as floor(3.2)?
    - Is fix(–3.2) the same as floor(–3.2)?
    - Is fix(–3.2) the same as ceil(–3.2)?

Esercizi

13. Find MATLAB expressions for the following: $\sqrt{19}$ $3^{1.2}$ `tan(π)`

14. Use intmin and intmax to determine the range of values that can be stored in the types uint32 and uint64.
15. Are there equivalents to intmin and intmax for real number types? Use help to find out.
16. Store a number with a decimal place in a double variable (the default). Convert the variable to the type int32 and store the result in a new variable.
17. Generate a random:
    - real number in the range from 0 to 1
    - real number in the range from 0 to 20
    - real number in the range from 20 to 50
    - integer in the range from 0 to 10
    - integer in the range from 0 to 11
    - integer in the range from 50 to 100
18. Open a new Command Window, and type *rand* to get a random real number. Make a note of the number. Then, exit MATLAB and repeat this, again making a note of the random number; it should be the same as before. Finally, exit MATLAB and again open a new Command Window. This time, change the seed before generating a random number; it should be different.
19. In the ASCII character encoding, the letters of the alphabet are in order; for example, ‘a’ comes before ‘b’ and also ‘A’ comes before ‘B’. However, which comes first—lowercase or uppercase letters?

Esercizi

20. Using the colon operator, create the following row vectors:

```
3    4    5    6

1.0000  1.5000  2.0000  2.5000  3.0000

5    4    3    2
```

```
     4    6    8

−3   −6   −9  −12  −15

     9    7    5
```

21. Using the linspace function, create the following row vectors:

```
1    2    3    4    5    6    7    8    9    10

2    7    12
```

22. Create the following vectors twice, using linspace and using the colon operator:

23. Create a variable myend that stores a random integer in the range from 8 to 12. Using the colon operator, create a vector that iterates from 1 to myend in steps of 3.
24. Using the colon operator and the transpose operator, create a column vector that has the values –1 to 1 in steps of 0.2.
25. Write an expression that refers to only the odd-numbered elements in a vector, regardless of the vector length. Test your expression on vectors that have both an odd and even number of elements.
26. Create a vector variable *vec*; it can have any length. Then, write assignment statements that would store the first half of the vector in one variable and the second half in another. Make sure that your assignment statements are general, and work whether *vec* has an even or odd number of elements. (Hint: Use a rounding function such as fix.)
27. Generate a 2x3 matrix of random
    - real numbers, each in the range from 0 to 1
    - real numbers, each in the range from 0 to 10
    - integers, each in the range from 5 to 20

Esercizi

28. Find an efficient way to generate the following matrix:

```
mat =
     7     8     9    10
    12    10     8     6
```

    Then, give expressions that will, for the matrix *mat*:
    - refer to the element in the first row, third column
    - refer to the entire second row
    - refer to the first two columns

28. Create a 2x3 matrix variable *mymat*. Pass this matrix variable to each of the following functions and make sure you understand the result: *fliplr*, *flipud*, and *rot90*. In how many different ways can you reshape it?
29. Create a 4x2 matrix of all zeros and store it in a variable. Then, replace the second row in the matrix with a vector consisting of a 3 and a 6.
30. Create a vector *x* that consists of 20 equally spaced points in the range from –π to + π. Create a *y* vector that is *sin(x)*.
33. Create a 3x5 matrix of random integers, each in the range from –5 to 5. Get the sign of every element.
34. Create a 4x6 matrix of real numbers, each in the range from –5 to 5; store it in a variable. Create another matrix that stores for each element the absolute value of the corresponding element in the original matrix.
35. Create a vector variable *vec*. Find as many expressions as you can that would refer to the last element in the vector, without assuming that you know how many elements it has (i.e., make your expressions general).
36. The built-in function clock returns a vector that contains six elements: the first three are the current date (year, month, day) and the last three represent the current time in hours, minutes, and seconds. The seconds is a real number, but all others are integers. Store the result from clock in a variable called *myc*. Then, store the first three elements from this variable in a variable *today* and the last three elements in a variable *now*. Use the *fix* function on the vector variable *now* to get just the integer part of the current time (hint: >> datestr(now, 'dd/mm/yy-HH:MM')).

Esercizi

37. Create a vector of structures experiments that stores information on subjects used in an experiment. Each struct has four fields: num, name, weights, and height. The field num is an integer, name is a string, weights is a vector with two values (both of which are double values), and height is a struct with the fields feet and inches (both of which are integers). The following is an example of what the format might look like.

experiments

			weights		height	
	num	name	1	2	feet	inches
1	33	Joe	200.34	202.45	5	6
2	11	Sally	111.45	111.11	7	2

38. Create a data structure to store information about the elements in the periodic table of elements. For every element, store the name, atomic number, chemical symbol, class, atomic weight, and a seven-element vector for the number of electrons in each shell. Create a structure variable to store the information. An example for lithium follows: Lithium 3 Li alkali_metal 6.94 2 1 0 0 0 0 0