



# Introduzione a Matlab

Informatica B AA 2017/2018

Luca Cassano

[luca.cassano@polimi.it](mailto:luca.cassano@polimi.it)

6 Novembre 2017



MATrix LABoratory



Cos'è Matlab (MATrix LABoratory):

- Ambiente di sviluppo e un linguaggio di programmazione per calcolo numerico



Cos'è Matlab (MATrix LABoratory):

- Ambiente di sviluppo e un linguaggio di programmazione per calcolo numerico
- È pensato (e ottimizzato) per operare su matrici (ma include generiche funzionalità matematiche)



Cos'è Matlab (MATrix LABoratory):

- Ambiente di sviluppo e un linguaggio di programmazione per calcolo numerico
- È pensato (e ottimizzato) per operare su matrici (ma include generiche funzionalità matematiche)

Lo utilizzerete nei successivi corsi di calcolo numerico

MATLAB è uno strumento commerciale, su licenza NON gratuita,

- Student edition fornita dal Politecnico (maggiori dettagli a laboratorio)



# L'interfaccia MATLAB

MATLAB R2012a

File Edit Debug Parallel Desktop Window Help

Current Folder: C:\Program Files\MATLAB\R2012a\bin

Workspace

Name	Value	Min	M
AXIS_FNT_SZ	22	22	22
FNT_SZ	24	24	24
False_shape	'o'		
K	1	1	1
LN_WDT	2	2	2
MRK_SZ	10	10	10
MS_shape	's'		
TS_color	'r'		
do_cv	1	1	1
do_filtering	1	1	1
do_short	1	1	1
do_show	0	0	0
figureNumber	254	254	25
filename_False	'../DataSet/Rialba_cat...		
filename_MS	'../DataSet/Rialba_bu...		
label_False	-3	-3	-3
label_False_TS	-2	-2	-2
label_False_VS	-1	-1	-1

Current Folder

« MATLAB » R2012a » bin »

Name

- m3iregistry
- registry
- util
- win64
- deploytool.bat
- insttype.ini
- lcdata.xml
- lcdata.xsd
- lcdata\_utf8.xml
- license.txt
- matlab.bat
- matlab.exe
- mbuild.bat
- mcc.bat
- mex.bat
- mex.pl
- mexext.bat

Editor - E:\Research\Roba Mia\2011\_03\_18\_Sensors\Matlab\Experiments\showEvent\_class...

```
1 close all
2 clear
3 clc
4
5 LN_WDT = 2;
6 MRK_SZ = 10;
7 FNT_SZ = 24;
8 AXIS_FNT_SZ = 22;
9
10 addpath('./Functions')
11
12 % modalita = 0 prende asse z
13 % modalita = 1 Registrazione via PCA
14 % modalita = 2 Prende Norme
15 % modalita = 3 Prende assi senza registrazione PCA
16
17 modalita = 1;
18 do_show = 0;
19 do_short = 1;
20 do_filtering = 1;
21
22 do_cv = 1;
23
24 figureNumber = 254;
25
26 TS_color = 'r';
27 test_color = 'b';
28 MS_shape = 's';
29 False_shape = 'o';
30 nodi_color = 'rgcyrckg';
31
32 label_MS_TS = 2;
33 label_False_TS = - 2;
```

Command Window

Warning: Name is nonexistent or not a directory: ..\Functions.  
> In path at 110  
In addpath at 87

fx >> |

computeFeatures.m createFigurePaperHist.m showEvent\_classification\_svm\_te...

Start Ready

OVR



# L'interfaccia MATLAB

MATLAB R2012a

File Edit Debug Parallel Desktop Window Help

Current Folder: C:\Program Files\MATLAB\R2012a\bin

Workspace

Name	Value	Min	M
AXIS_FNT_SZ	22	22	22
FNT_SZ	24	24	24
False_shape	'o'		
K	1	1	1
LN_WDT	2	2	2
MRK_SZ	10	10	10
MS_shape	's'		
TS_color	'r'		
do_cv	1	1	1
do_filtering	1	1	1
do_short	1	1	1
do_show	0	0	0
figureNumber	254	254	25
filename_False	..\DataSet\Rialba_cat...		
filename_MS	..\DataSet\Rialba_bu...		
label_False	-3	-3	-3
label_False_TS	-2	-2	-2
label_False_VS	-1	-1	-1

Current Folder

« MATLAB » R2012a » bin »

Editor - E:\Research\Roba Mia\2011\_03\_18\_Sensors\Matlab\Experiments\showEvent\_class...

```
1 close all
2 clear
3 clc
4
5 LN_WDT = 2;
6 MRK_SZ = 10;
7 FNT_SZ = 24;
8 AXIS_FNT_SZ = 22;
9
10 addpath('..\Functions')
11
12 % modalita = 0 prende asse z
13 % modalita =
14 % modalita =
15 % modalita = 3 Prende assi senza registrazione PCA
16
17 modalita = 1;
18 do_show = 0;
19 do_short = 1;
20 do_filtering = 1;
21
22 do_cv = 1;
23
24 figureNumber = 254;
25
26 TS_color = 'r';
27 test_color = 'b';
28 MS_shape = 's';
29 False_shape = 'o';
30 nodi_color = 'rgcyrckg';
31
32 label_MS_TS = 2;
33 label_False_TS = - 2;
```

Command Window

Warning: Name is nonexistent or not a directory: ..\Functions.  
> In path at 110  
In addpath at 87

fx >> |

Codice nell'Editor

computeFeatures.m createFigurePaperHist.m showEvent\_classification\_svm\_te...

Start Ready

OVR



# L'interfaccia MATLAB

MATLAB R2012a

File Edit Debug Parallel Desktop Window Help

Current Folder: C:\Program Files\MATLAB\R2012a\bin

Workspace

Name	Value	Min	M
AXIS_FNT_SZ	22	22	22
FNT_SZ	24	24	24
False_shape	'o'		
K	1	1	1
LN_WDT	2	2	2
MRK_SZ	10	10	10
MS_shape	's'		
TS_color	'r'		
do_cv	1	1	1
do_filtering			
do_short			
do_show			
figureNumber			
filename_False			
filename_MS			
label_False			
label_False_TS			
label_False_VS			

Mostra le variabili nel workspace

Current Folder

« MATLAB » R2012a » bin »

Editor - E:\Research\Roba Mia\2011\_03\_18\_Sensors\Matlab\Experiments\showEvent\_class...

```
1 close all
2 clear
3 clc
4
5 LN_WDT = 2;
6 MRK_SZ = 10;
7 FNT_SZ = 24;
8 AXIS_FNT_SZ = 22;
9
10 addpath('./Functions')
11
12 % modalita = 0 prende asse z
13 % modalita =
14 % modalita =
15 % modalita = 5 Prende assi senza registrazione PCA
16
17 modalita = 1;
18 do_show = 0;
19 do_short = 1;
20 do_filtering = 1;
21
22 do_cv = 1;
23
24 figureNumber = 254;
25
26 TS_color = 'r';
27 test_color = 'b';
28 MS_shape = 's';
29 False_shape = 'o';
30 nodi_color = 'rgcyrcgk';
31
32 label_MS_TS = 2;
33 label_False_TS = - 2;
```

Codice nell'Editor

Command Window

Warning: Name is nonexistent or not a directory: ..\Functions.  
> In path at 110  
In addpath at 87

fx >> |

computeFeatures.m createFigurePaperHist.m showEvent\_classification\_svm\_te...

Start Ready

OVR





# L'interfaccia MATLAB

MATLAB R2012a

File Edit Debug Parallel Desktop Window Help

Current Folder: C:\Program Files\MATLAB\R2012a\bin

Workspace

Name	Value	Min	M
AXIS_FNT_SZ	22	22	22
FNT_SZ	24	24	24
False_shape	'o'		
K	1	1	1
LN_WDT	2	2	2
MRK_SZ	10	10	10
MS_shape	's'		
TS_color	'r'		
do_cv	1	1	1
do_filtering			
do_short			
do_show			
figureNumber			
filename_False			
filename_MS			
label_False			
label_False_TS			
label_False_VS			

Mostra le variabili nel workspace

Current Folder

« MATLAB » R2012a » bin »

Editor - E:\Research\Roba Mia\2011\_03\_18\_Sensors\Matlab\Experiments\showEvent\_class...

```
1 close all
2 clear
3 clc
4
5 LN_WDT = 2;
6 MRK_SZ = 10;
7 FNT_SZ = 24;
8 AXIS_FNT_SZ = 22;
9
10 addpath('./Functions')
11
12 % modalita = 0 prende asse z
13 % modalita =
14 % modalita =
15 % modalita = 3 Prende assi senza registrazione PCA
16
17 modalita = 1;
18 do_show = 0;
19 do_short = 1;
20 do_filtering = 1;
21
22 do_cv = 1;
23
24 figureNumber = 254;
25
26 TS_color = 'r';
27 test_color = 'b';
28 MS_shape = 's';
29 False_shape = 'o';
30 nodi_color = 'rgcyrckg';
31
32 label_MS_TS = 2;
33 label_False_TS = - 2;
```

Codice nell'Editor

Command Window

Warning: Name is nonexistent or not a directory: Functions.  
> In path at 110  
In addpath at 87

fx >> |

Finestra dei comandi

computeFeatures.m createFigurePaperHist.m showEvent\_classification\_svm\_te...

Start Ready

OVR



# L'interfaccia MATLAB

MATLAB R2012a

File Edit Debug Parallel Desktop Window Help

Current Folder: C:\Program Files\MATLAB\R2012a\bin

Workspace

Name	Value	Min	M
AXIS_FNT_SZ	22	22	22
FNT_SZ	24	24	24
False_shape	'o'		
K	1	1	1
LN_WDT	2	2	2
MRK_SZ	10	10	10
MS_shape	's'		
TS_color	'r'		
do_cv	1	1	1

Mostra le variabili nel workspace

Current Folder

Content of the current directory

Editor - E:\Research\Roba Mia\2011\_03\_18\_Sensors\Matlab\Experiments\showEvent\_class...

```
1 close all
2 clear
3 clc
4
5 LN_WDT = 2;
6 MRK_SZ = 10;
7 FNT_SZ = 24;
8 AXIS_FNT_SZ = 22;
9
10 addpath('./Functions')
11
12 % modalita = 0 prende asse z
13 % modalita =
14 % modalita =
15 % modalita = 3 Prende assi senza registrazione PCA
16
17 modalita = 1;
18 do_show = 0;
19 do_short = 1;
20 do_filtering = 1;
21
22 do_cv = 1;
23
24 figureNumber = 254;
25
26 TS_color = 'r';
27 test_color = 'b';
28 MS_shape = 's';
29 False_shape = 'o';
30 nodi_color = 'rgcyrckg';
31
32 label_MS_TS = 2;
33 label_False_TS = - 2;
```

Codice nell'Editor

Command Window

```
Warning: Name is nonexistent or not a
directory: Functions.
> In path at 110
In addpath at 87
fx >> |
```

Finestra dei comandi

Start Ready



# L'interfaccia MATLAB

The screenshot shows the MATLAB R2012a environment with several windows and annotations:

- Workspace:** A table of variables in the current workspace.
- Editor:** A script file named `showEvent_classification_svm_te...` is open, showing MATLAB code.
- Command Window:** Displays a warning message about a nonexistent directory.
- Current Folder:** Shows the file structure of the current directory.

Annotations with arrows pointing to specific elements:

- Mostra le variabili nel workspace:** Points to the Workspace window.
- Codice nell'Editor:** Points to the Editor window.
- Finestra dei comandi:** Points to the Command Window.
- Contenuto della directory corrente:** Points to the Current Folder window.
- Lancia i tool di MATLAB ed altro...** Points to the MATLAB icon in the taskbar.

Name	Value	Min	M
AXIS_FNT_SZ	22	22	22
FNT_SZ	24	24	24
False_shape	'o'		
K	1	1	1
LN_WDT	2	2	2
MRK_SZ	10	10	10
MS_shape	's'		
TS_color	'r'		
do_cv	1	1	1

```
close all
clear
clc

LN_WDT = 2;
MRK_SZ = 10;
FNT_SZ = 24;
AXIS_FNT_SZ = 22;

addpath('./Functions')

% modalita = 0 prende asse z
% modalita =
% modalita =
% modalita = 3 prende assi senza registrazione PCA

modalita = 1;
do_show = 0;
do_short = 1;
do_filtering = 1;

do_cv = 1;

figureNum

TS_color = 'r';
test_color = 'b';
MS_shape = 's';
False_shape = 'o';
nodi_color = 'rgcyrckg';

label_MS_TS = 2;
label_False_TS = - 2;
```

Warning: Name is nonexistent or not a directory: Functions.  
> In path at 110  
In addpath at 87

fx >> |

Current Folder: C:\Program Files\MATLAB\R2012a\bin

Details: m3iregistry, registry, util, win64, deploytool.bat, insttype.ini, lcddata.xml, lcddata.xsd, lcddata\_utf8.xml, license.txt, matlab.bat, matlab.exe, mbuild.bat, mcc.bat, mex.bat, mexpl, mtest.bat, mtest.m



Esiste una soluzione alternativa: Octave

- identico nella concezione, molto simile per gli aspetti operativi
- disponibile gratuitamente, vedi [www.gnu.org/software/octave/](http://www.gnu.org/software/octave/)
- Occorre installare un'interfaccia grafica qt octave
- Vedrete tutto a laboratorio



# Caratteristiche del linguaggio di Matlab (1)

Linguaggio di alto livello

- Simile a linguaggi di programmazione C, Java, Pascal



# Caratteristiche del linguaggio di Matlab (1)

Linguaggio di alto livello

- Simile a linguaggi di programmazione C, Java, Pascal
- Possiede comandi sintetici per effettuare complesse elaborazioni numeriche



# Caratteristiche del linguaggio di Matlab (1)

Linguaggio di alto livello

- Simile a linguaggi di programmazione C, Java, Pascal
- Possiede comandi sintetici per effettuare complesse elaborazioni numeriche

Linguaggio interpretato, i comandi e istruzioni

- NON sono tradotti in codice eseguibile dall'hardware



# Caratteristiche del linguaggio di Matlab (1)

Linguaggio di alto livello

- Simile a linguaggi di programmazione C, Java, Pascal
- Possiede comandi sintetici per effettuare complesse elaborazioni numeriche

Linguaggio interpretato, i comandi e istruzioni

- NON sono tradotti in codice eseguibile dall'hardware
- Ma invia istruzioni ad un altro programma, **l'interprete**, che li analizza ed esegue azioni da essi descritte

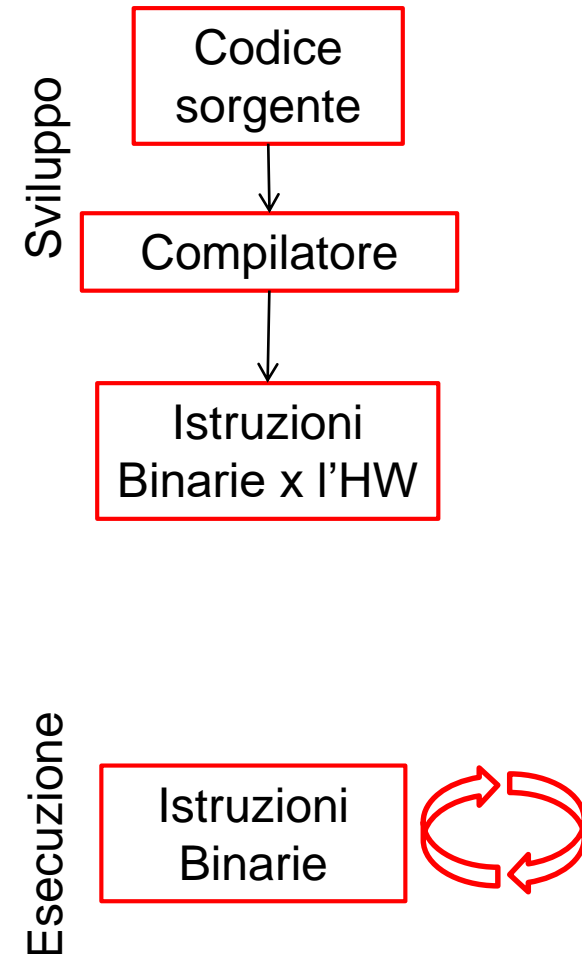




# Linguaggi Compilati vs Interpretati

## Linguaggi **Compilati**:

- Il compilatore è un programma che **traduce** le istruzioni del codice sorgente in codice macchina (in binario)

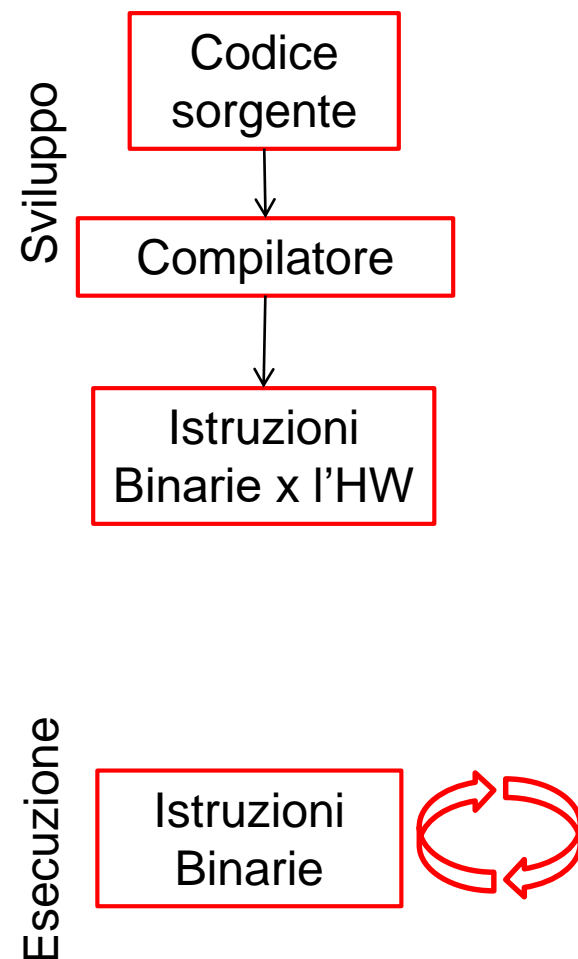




# Linguaggi Compilati vs Interpretati

## Linguaggi **Compilati**:

- Il compilatore è un programma che **traduce** le istruzioni del codice sorgente in codice macchina (in binario)
- L'esecuzione del programma non richiede la presenza del codice sorgente, né del compilatore.

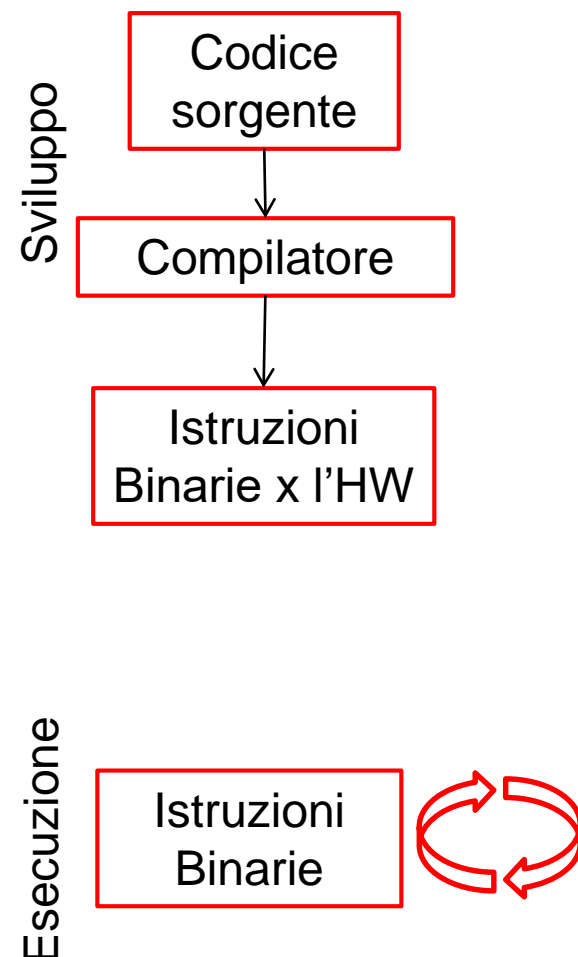




# Linguaggi Compilati vs Interpretati

## Linguaggi **Compilati**:

- Il compilatore è un programma che **traduce** le istruzioni del codice sorgente in codice macchina (in binario)
- L'esecuzione del programma non richiede la presenza del codice sorgente, né del compilatore.
- I programmi sono efficienti

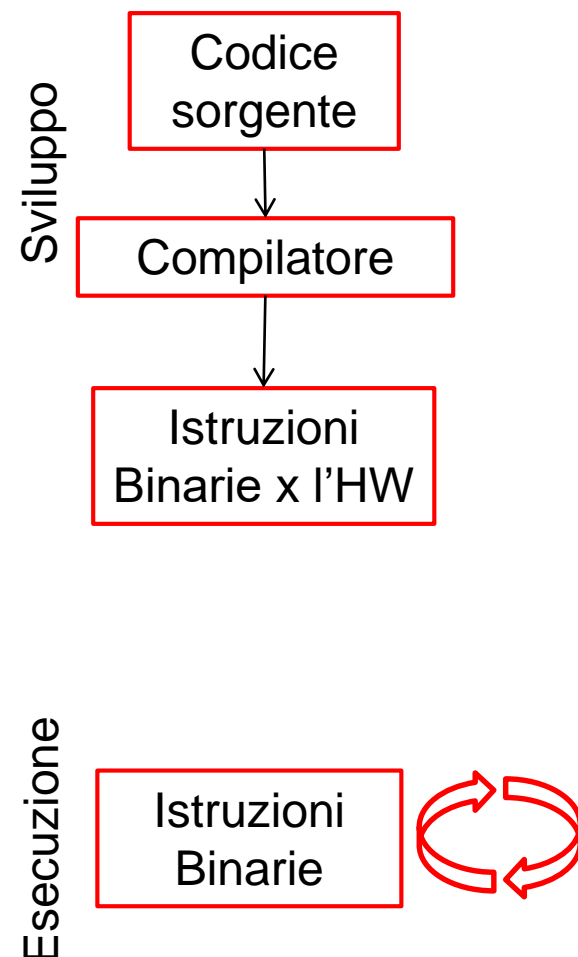




# Linguaggi Compilati vs Interpretati

## Linguaggi **Compilati**:

- Il compilatore è un programma che **traduce** le istruzioni del codice sorgente in codice macchina (in binario)
- L'esecuzione del programma non richiede la presenza del codice sorgente, né del compilatore.
- I programmi sono efficienti
- Il programma è facilmente portabile su piattaforme analoghe

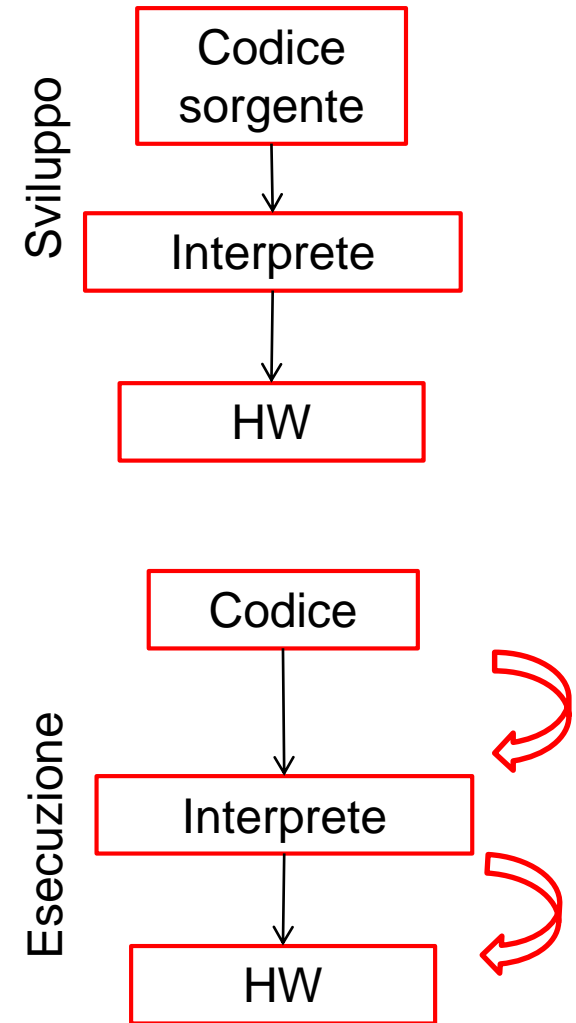




# Linguaggi Compilati vs Interpretati

## Linguaggi **Interpretati**:

- L'interprete è un programma che **esegue** istruzioni contenute nel codice sorgente

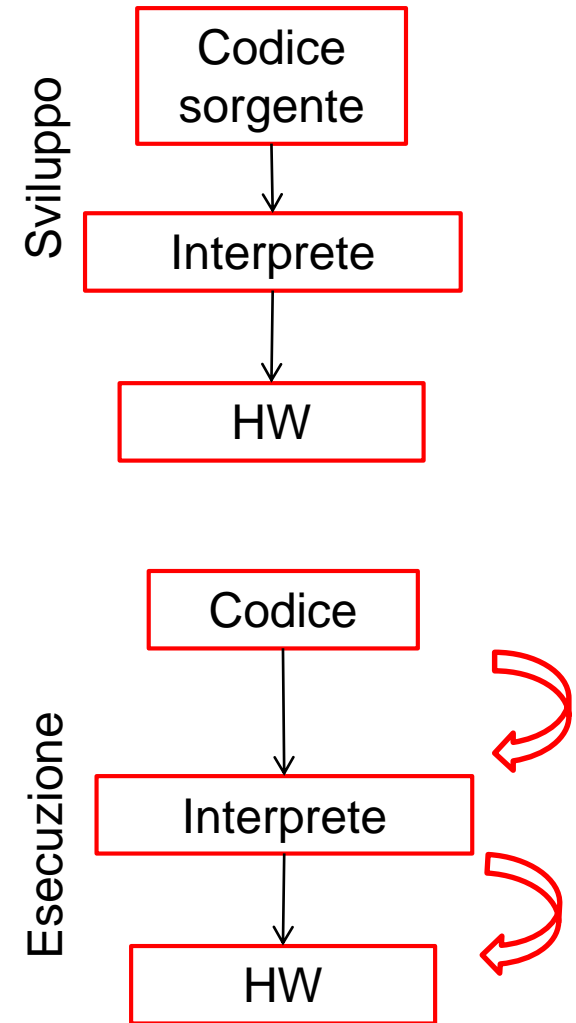




# Linguaggi Compilati vs Interpretati

## Linguaggi **Interpretati**:

- L'interprete è un programma che **esegue** istruzioni contenute nel codice sorgente
- L'esecuzione del programma richiede la presenza dell'interprete.

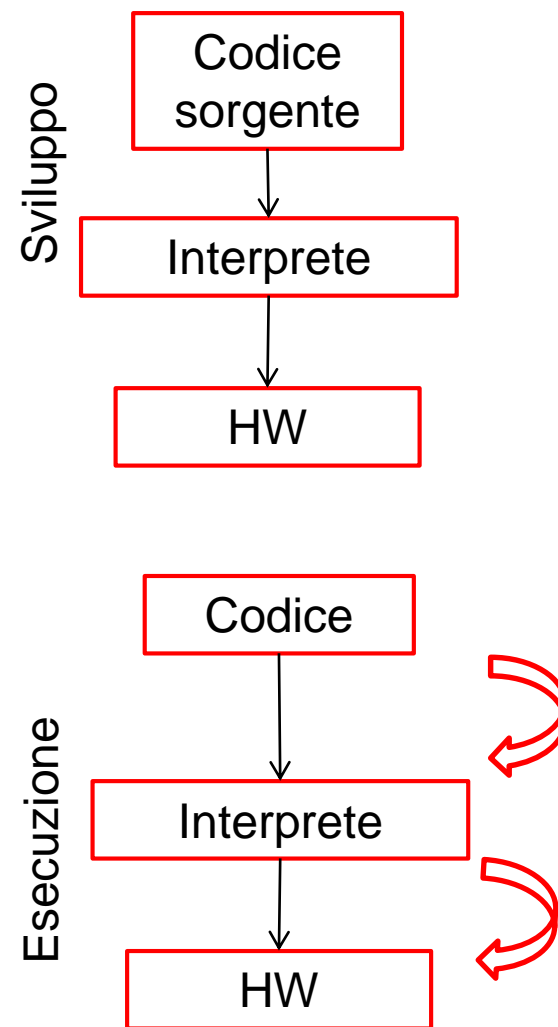




# Linguaggi Compilati vs Interpretati

## Linguaggi **Interpretati**:

- L'interprete è un programma che **esegue** istruzioni contenute nel codice sorgente
- L'esecuzione del programma richiede la presenza dell'interprete.
- I programmi sono meno efficienti di quelli compilati

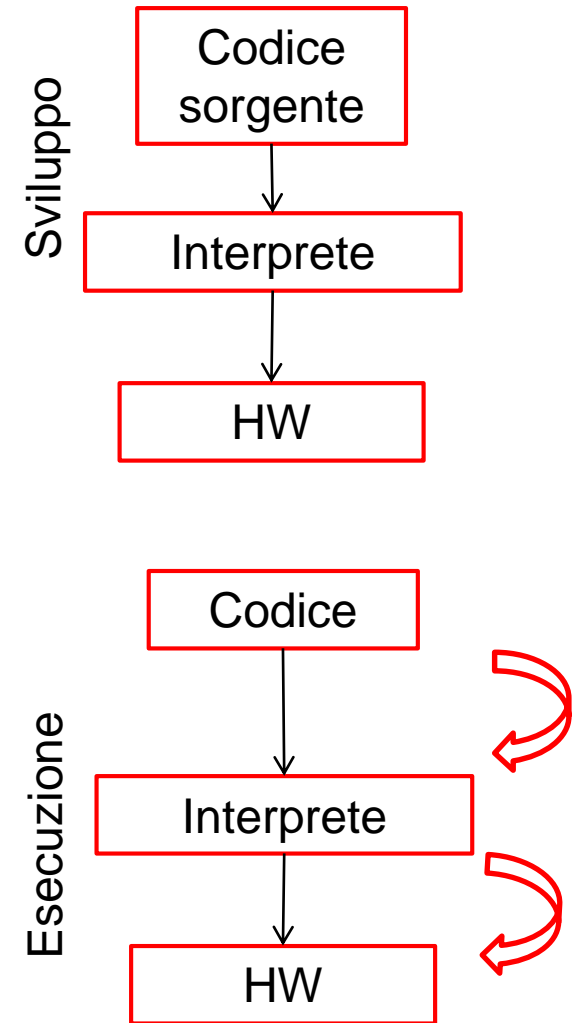




# Linguaggi Compilati vs Interpretati

## Linguaggi **Interpretati**:

- L'interprete è un programma che **esegue** istruzioni contenute nel codice sorgente
- L'esecuzione del programma richiede la presenza dell'interprete.
- I programmi sono meno efficienti di quelli compilati
- Portabilità meno pratica



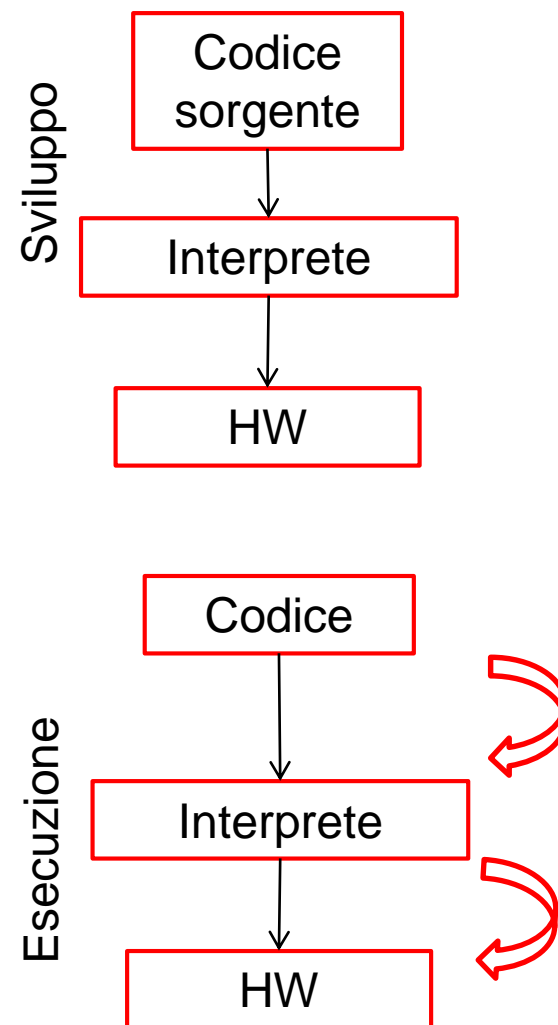




# Linguaggi Compilati vs Interpretati

## Linguaggi **Interpretati**:

- L'interprete è un programma che **esegue** istruzioni contenute nel codice sorgente
- L'esecuzione del programma richiede la presenza dell'interprete.
- I programmi sono meno efficienti di quelli compilati
- Portabilità meno pratica
- Sviluppo più facile: è possibile eseguire le istruzioni mentre si scrive il codice sorgente





## Caratteristiche del linguaggio di Matlab (2)

### Linguaggio dinamico (non tipizzato)

- **NON occorre dichiarare le variabili**
  - risultano definite al primo assegnamento
  - vengono incluse in una struttura detta *tabella dei simboli*



## Caratteristiche del linguaggio di Matlab (2)

### Linguaggio dinamico (non tipizzato)

- **NON occorre dichiarare le variabili**
  - risultano definite al primo assegnamento
  - vengono incluse in una struttura detta *tabella dei simboli*
- **il tipo delle variabili è *dinamico***
  - a una variabile si possono assegnare, successivamente, valori di tipo diverso (scalari, stringhe, vettori, matrici...)



## Nella pratica Matlab...

...può far riferimento a 3 cose diverse:

- Il linguaggio Matlab che utilizziamo per codificare i programmi



## Nella pratica Matlab...

...può far riferimento a 3 cose diverse:

- Il linguaggio Matlab che utilizziamo per codificare i programmi
- L'interprete Matlab che viene invocato per eseguire i nostri programmi



## Nella pratica Matlab...

...può far riferimento a 3 cose diverse:

- Il linguaggio Matlab che utilizziamo per codificare i programmi
- L'interprete Matlab che viene invocato per eseguire i nostri programmi
- L'ambiente di sviluppo integrato che permette di scrivere ed eseguire i programmi



# Le Istruzioni



## Le Istruzioni e la Command Window

Le **istruzioni** possono essere **inviate** direttamente **all'interprete** se scritte nella command window (dopo il simbolo `>>` )

- La command window è come una «super calcolatrice»
- La command window ha un'interfaccia testuale che inizia con `>>`





# Screenshot interfaccia MATLAB

MATLAB R2012a

File Edit Debug Parallel Desktop Window Help

Current Folder: C:\Program Files\MATLAB\R2012a\bin

Workspace

Name	Value	Min	M
AXIS_FNT_SZ	22	22	22
FNT_SZ	24	24	24
False_shape	'o'		
K	1	1	1
LN_WDT	2	2	2
MRK_SZ	10	10	10
MS_shape	's'		
TS_color	'r'		
do_cv	1	1	1
do_filtering	1	1	1
do_short	1	1	1
do_show	0	0	0
figureNumber	254	254	25
filename_False	'./DataSet/Rialba_cat...		
filename_MS	'./DataSet/Rialba_bu...		
label_False	-3	-3	-3
label_False_TS	-2	-2	-2
label_False_VS	-1	-1	-1

Current Folder

« MATLAB » R2012a » bin »

Name

- m3iregistry
- registry
- util
- win64
- deploytool.bat
- insttype.ini
- lcdata.xml
- lcdata.xsd
- lcdata\_utf8.xml
- license.txt
- matlab.bat
- matlab.exe
- mbuild.bat
- mcc.bat
- mex.bat
- mex.pl
- mexext.bat

Editor - E:\Research\Roba Mia\2011\_03\_18\_Sensors\Matlab\Experiments\showEvent\_class...

```
1 close all
2 clear
3 clc
4
5 LN_WDT = 2;
6 MRK_SZ = 10;
7 FNT_SZ = 24;
8 AXIS_FNT_SZ = 22;
9
10 addpath('./Functions')
11
12 % modalita = 0 prende asse z
13 % modalita = 1 Registrazione via PCA
14 % modalita = 2 Prende Norme
15 % modalita = 3 Prende assi senza registrazione PCA
16
17 modalita = 1;
18 do_show = 0;
19 do_short = 1;
20 do_filtering = 1;
21
22 do_cv = 1;
23
24 figureNumber = 254;
25
26 TS_color = 'r';
27 test_color = 'b';
28 MS_shape = 's';
29 False_shape = 'o';
30 nodi_color = 'rgcyrckg';
31
32 label_MS_TS = 2;
33 label_False_TS = - 2;
```

Command Window

Warning: Name is nonexistent or not a directory: Functions.  
> In path at 110  
In addpath at 87

fx >> |

Finestra dei comandi  
(Command Window)

- 33 -

computeFeatures.m createFigurePaperHist.m showEvent\_classification\_svm\_te...

Start Ready

OV



## Esempio: le operazioni aritmetiche

Nella command window è possibile eseguire qualsiasi operazione aritmetica

```
>> 5 + 7
```

```
ans =
```

```
12
```

```
>> 5 / 7
```

```
ans =
```

```
0.7143
```

**ans** è una variabile «di default» che contiene il risultato di un'istruzione che sia un assegnamento



## Esempio: le operazioni aritmetiche

Nella command window è possibile eseguire qualsiasi operazione aritmetica

Elevamento a potenza

```
>> 5 + 7
```

```
ans =
```

```
12
```

```
>> 5 * 7
```

```
ans =
```

```
35
```

```
>> 5 ^ 7
```

```
ans =
```

```
78125
```

```
>> 5 / 7
```

```
ans =
```

```
0.7143
```

```
>> 'a' + 2
```

```
ans =
```

```
99
```

I caratteri alfanumerici si indicano con l'apice singolo: sono sempre legati agli interi mediante la tabella ASCII



## Istruzioni e Codice Sorgente

Le istruzioni possono essere contenute in un **file sorgente**, in particolare:

- uno script
- una funzione

e quindi eseguite in maniera sequenziale.



## Istruzioni e Codice Sorgente

Le istruzioni possono essere contenute in un **file sorgente**, in particolare:

- uno script
- una funzione

e quindi eseguite in maniera sequenziale.

L'esecuzione di uno codice sorgente può essere visto come l'inserimento automatizzato delle varie istruzioni nella command window.



# Screenshot interfaccia MATLAB

MATLAB R2012a

File Edit Debug Parallel Desktop Window Help

Current Folder: C:\Program Files\MATLAB\R2012a\bin

Workspace

Name	Value	Min	M
AXIS_FNT_SZ	22	22	22
FNT_SZ	24	24	24
False_shape	'o'		
K	1	1	1
LN_WDT	2	2	2
MRK_SZ	10	10	10
MS_shape	's'		
TS_color	'r'		
do_cv	1	1	1
do_filtering	1	1	1
do_short	1	1	1
do_show	0	0	0
figureNumber	254	254	25
filename_False	..\DataSet\Rialba_cat...		
filename_MS	..\DataSet\Rialba_bu...		
label_False	-3	-3	-3
label_False_TS	-2	-2	-2
label_False_VS	-1	-1	-1

Current Folder

« MATLAB » R2012a » bin »

Name

- m3iregistry
- registry
- util
- win64
- deploytool.bat
- insttype.ini
- lcdata.xml
- lcdata.xsd
- lcdata\_utf8.xml
- license.txt
- matlab.bat
- matlab.exe
- mbuild.bat
- mcc.bat
- mex.bat
- mex.pl
- mexext.bat

Editor - E:\Research\Roba Mia\2011\_03\_18\_Sensors\Matlab\Experiments\showEvent\_class...

```
1 close all
2 clear
3 clc
4
5 LN_WDT = 2;
6 MRK_SZ = 10;
7 FNT_SZ = 24;
8 AXIS_FNT_SZ = 22;
9
10 addpath('..\Functions')
11
12 % modalita = 0 prende asse z
13 % modalita = 1 Registrazione via PCA
14 % modalita = 2 Prende Norme
15 % m
16
17 mod
18 do_
19 do_
20 do_filtering = 1;
21
22 do_cv = 1;
23
24 figureNumber = 254;
25
26 TS_color = 'r';
27 test_color = 'b';
28 MS_shape = 's';
29 False_shape = 'o';
30 nodi_color = 'rgcyrckg';
31
32 label_MS_TS = 2;
33 label_False_TS = - 2;
```

Command Window

Warning: Name is nonexistent or not a directory: ..\Functions.  
> In path at 110  
In addpath at 87

fx >> |

Editor, permette di scrivere funzioni e script

- 38 -

computeFeatures.m createFigurePaperHist.m showEvent\_classification\_svm\_te...

Start Ready

OVR



## Istruzioni e ‘;’

Le istruzioni **possono terminare** con ‘;’ ma non è obbligatorio

Di default, il risultato di ogni istruzione viene visualizzato nella command window.



## Istruzioni e ‘;’

Le istruzioni **possono terminare** con ‘;’ ma non è obbligatorio

Di default, il risultato di ogni istruzione viene visualizzato nella command window.

Il ‘;’ **blocca la visualizzazione** del risultato dell’istruzione

- Maggiore velocità di esecuzione
- Visualizzazione più compatta





## Istruzioni e ';'

Le istruzioni **possono terminare** con ';' ma non è obbligatorio

Di default, il risultato di ogni istruzione viene visualizzato nella command window.

Il ';' **blocca la visualizzazione** del risultato dell'istruzione

- Maggiore velocità di esecuzione
- Visualizzazione più compatta

Regola di buona programmazione

- Inserire sempre il ';' a meno che non si voglia ispezionare il valore di una variabile a scopo di debugging



# Gli array (le variabili)

MATrix LABoratory...



## Creazione ed Inizializzazione di una Variabile

Le variabili sono **create** mediante **inizializzazione**

- Cioè alla prima istruzione in cui compaiono. Non occorre dichiarare le variabili come in C



# Creazione ed Inizializzazione di una Variabile

Le variabili sono **create** mediante **inizializzazione**

- Cioè alla prima istruzione in cui compaiono. Non occorre dichiarare le variabili come in C

## Modi di inizializzazione

- **Assegnamento**
- Lettura dati da tastiera
- Lettura da file



## Istruzione di Assegnamento

Come in C,

```
nomeVariabile = espressione
```



# Istruzione di Assegnamento

Come in C,

**nomeVariabile = espressione**

A differenza del C:

- non si deve (non è possibile) dichiarare la variabile **nomeVariabile** prima della assegnamento.



## Istruzione di Assegnamento

Come in C,

**nomeVariabile = espressione**

A differenza del C:

- non si deve (non è possibile) dichiarare la variabile **nomeVariabile** prima della assegnamento.
- L'assegnamento comporta una dichiarazione implicita della variabile **nomeVariabile**.



## Istruzione di Assegnamento

Come in C,

**nomeVariabile = espressione**

A differenza del C:

- non si deve (non è possibile) dichiarare la variabile **nomeVariabile** prima della assegnamento.
- L'assegnamento comporta una dichiarazione implicita della variabile **nomeVariabile**.
- **È possibile eseguire assegnamento tra array**





## Istruzione di Assegnamento

Come in C,

**nomeVariabile = espressione**

A differenza del C:

- non si deve (non è possibile) dichiarare la variabile **nomeVariabile** prima della assegnamento.
- L'assegnamento comporta una dichiarazione implicita della variabile **nomeVariabile**.
- **È possibile eseguire assegnamento tra array**
- Non è richiesto il ; al termine dell'istruzione



## Istruzione di Assegnamento

Come in C,

**nomeVariabile = espressione**

A differenza del C:

- non si deve (non è possibile) dichiarare la variabile **nomeVariabile** prima della assegnamento.
- L'assegnamento comporta una dichiarazione implicita della variabile **nomeVariabile**.
- **È possibile eseguire assegnamento tra array**
- Non è richiesto il ; al termine dell'istruzione
- Il risultato di un'operazione che non comporta un assegnamento viene assegnato alla variabile **ans**



## Assegnamento ed Inizializzazione

Quando assegno un valore ad una variabile che non è stata inizializzata (e.g., **a**), la **variabile viene creata**

```
>> a = 7
```

```
a =
```

```
7
```



## Assegnamento ed Inizializzazione

Quando assegno un valore ad una variabile che non è stata inizializzata (e.g., **a**), la **variabile viene creata**

```
>> a = 7
```

```
a =
```

```
7
```

Ovviamente non è possibile assegnare ad una variabile, il valore di una variabile che non esiste:

```
>> a = v
```

**Undefined function or variable 'v'.**

(messaggio di errore dell'interprete Matlab)



## Caratteristiche del linguaggio di Matlab (3)

In Matlab tutto è un array, i.e.,

- **scalari:** array 1x1
- **vettori:** array con una sola riga o colonna
- **matrici:** array con due dimensioni
- **matrici multidimensionali:** array con più di 2 dimensioni

Il **tipo** delle variabili è definito dal valore che contengono (e viene definito al momento dell'assegnamento)



## Il Workspace:

Tutte le variabili risiedono nel **workspace**

Per vedere le variabili attualmente nel workspace si usa il comando : **whos**

```
>> a = 7
```

```
>> whos
```

Name	Size	Bytes	Class	Attributes
a	1x1	8	double	



## Operazioni algebriche, assegnamento e confronto tra scalari

Input	Output	Commento
1234/6	<code>ans= 205.6667</code>	calcolo di un valore scalare
<code>a=1234/6</code>	<code>a = 205.6667</code>	assegnamento alla variabile <code>a</code> del risultato di 1234/6
2/5	<code>ans = 0.40000</code>	divisione
5/0	<code>ans = Inf</code>	divisione per zero
5^2	<code>ans = 25</code>	potenza
<code>1+1==2</code> <code>1+1~=2</code>	<code>ans = 1</code> <code>ans = 0</code>	1 = vero, 0 = falso, “==” uguale, “~=” diverso



# Operazioni algebriche, assegnamento e confronto tra scalari

Input	Output	Commento
1234/6	ans= 205.6667	calcolo di un valore scalare
a=1234/6	a = 205.6667	assegnamento alla variabile a del risultato di 1234/6
2/5	ans = 0.40000	divisione
5/0	ans = Inf	divisione per zero
5^2	ans = 25	potenza
1+1==2 1+1~=2	ans = 1 ans = 0	1 = vero, 0 = falso, “==” uguale, “~=” diverso

La negazione in Matlab è rappresentata dal simbolo ~





## Definizione di Vettori

I vettori sono definiti tra parentesi quadre:



## Definizione di Vettori

I vettori sono definiti tra parentesi quadre:

- In un vettore riga gli elementi sono separati da virgole (o spazi)

Es:

<code>&gt;&gt; a = [1 2 3]</code>	<code>&gt;&gt; a = [1, 2, 3]</code>
<code>a =</code>	<code>a =</code>
<code>1 2 3</code>	<code>1 2 3</code>



## Definizione di Vettori

I vettori sono definiti tra parentesi quadre:

- In un vettore riga gli elementi sono separati da virgole (o spazi)
- In un vettore colonna gli elementi sono separati da ; (o andando a capo)

Es:

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

```
a =
```

```
1    2    3
```

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

```
a =
```

```
1    2    3
```

```
>> a = [1; 2; 3]
```

```
a =
```

```
1  
2  
3
```



## Operatori per Array: Trasposizione

L'operatore ' esegue la **trasposizione** (i.e. trasforma un vettore riga ad una colonna e viceversa)

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

```
>> a'
ans =
    1
    2
    3
```



## Operatori per Array: Definizione mediante incremento Regolare

L'operatore : definisce **vettori** ad **incremento regolare**:

`vett = [inizio : step : fine]`



## Operatori per Array: Definizione mediante incremento Regolare

L'operatore : definisce **vettori** ad **incremento regolare**:

`vett = [inizio : step : fine]`

Definisce un vettore `vett` che ha:

- primo elemento `inizio`



## Operatori per Array: Definizione mediante incremento Regolare

L'operatore : definisce **vettori** ad **incremento regolare**:

`vett = [inizio : step : fine]`

Definisce un vettore `vett` che ha:

- primo elemento `inizio`
- secondo elemento `inizio + step`



## Operatori per Array: Definizione mediante incremento Regolare

L'operatore : definisce **vettori** ad **incremento regolare**:

`vett = [inizio : step : fine]`

Definisce un vettore `vett` che ha:

- primo elemento `inizio`
- secondo elemento `inizio + step`
- terzo elemento `inizio + 2*step`





## Operatori per Array: Definizione mediante incremento Regolare

L'operatore : definisce **vettori** ad **incremento regolare**:

`vett = [inizio : step : fine]`

Definisce un vettore `vett` che ha:

- primo elemento `inizio`
- secondo elemento `inizio + step`
- terzo elemento `inizio + 2*step`
- ...
- fino al più grande valore `inizio + k*step` (con `k` intero positivo) che non supera `fine` (**`fine` potrebbe non essere incluso**)



# Operatori per Array: Definizione mediante incremento Regolare

L'operatore : definisce **vettori** ad **incremento regolare**:

`vett = [inizio : step : fine]`

Definisce un vettore `vett` che ha:

- primo elemento `inizio`
- secondo elemento `inizio + step`
- terzo elemento `inizio + 2*step`
- ...
- fino al più grande valore `inizio + k*step` (con `k` intero positivo) che non supera `fine` (**`fine` potrebbe non essere incluso**)
- Tipicamente se `inizio < fine`, `step > 0` e se `inizio > fine`, `step < 0`



## Note sulla Creazione dei Vettori

Il valore di **step** può essere qualsiasi, anche negativo (e anche reale).



## Note sulla Creazione dei Vettori

Il valore di **step** può essere qualsiasi, anche negativo (e anche reale).

Se non precisato, **step** vale 1 (**vett** = **[inizio:fine]**)



## Note sulla Creazione dei Vettori

Il valore di **step** può essere qualsiasi, anche negativo (e anche reale).

Se non precisato, **step** vale 1 (**vett** = **[inizio:fine]**)

Le parentesi [ ] possono essere omesse  
(es>> **y = 1:1:10**)



## Note sulla Creazione dei Vettori

Il valore di **step** può essere qualsiasi, anche negativo (e anche reale).

Se non precisato, **step** vale 1 (**vett** = **[inizio:fine]**)

Le parentesi **[ ]** possono essere omesse  
(es >> **y = 1:1:10**)

Attenzione che i vettori definiti per incremento regolare possono essere vuoti (es >> **[20 : 1 : 10]**)



# Definizione mediante incremento Regolare:

## Esempi

```
>> a = [1 : 1 : 3]
```

```
a =
```

```
1    2    3
```

```
>> a = [1 : 2 : 3]
```

```
a =
```

```
1    3
```

```
>> a = [1 : 2 : 10]
```

```
a =
```

```
1    3    5    7    9
```

```
>> a = [1 : 3]
```

```
a =
```

```
1    2    3
```

```
>> a = [10 : -2 : 3]
```

```
a =
```

```
10    8    6    4
```



## Note sulla Creazione dei Vettori

È ovviamente possibile modificare i valori di un array mediante assegnamento

- Di un singolo elemento (come in C)
- Di una parte dell'array
- Di tutto l'array





## Assegnamento tra Array

In Matlab è possibile eseguire direttamente assegnamenti tra array

`nomeArray1 = nomeArray2`

**Copia** i valori contenuti in `nomeArray2` in `nomeArray1`

Es

```
>> a = [1 2 3];
```

```
>> b = a
```

```
b =
```

```
1      2      3
```



## Accedere agli Elementi di un Vettore

Notazione simile al C

**nomeVettore(indice)**

- Restituisce il valore contenuto in **nomeVettore** alla posizione **indice**.
- Come nel C, una volta specificato l'indice si accede all'elemento del vettore come ad una qualsiasi variabile (per assegnamenti ed altre operazioni)



## Accedere agli Elementi di un Vettore

Notazione simile al C

**nomeVettore(indice)**

- Restituisce il valore contenuto in **nomeVettore** alla posizione **indice**.
- Come nel C, una volta specificato l'indice si accede all'elemento del vettore come ad una qualsiasi variabile (per assegnamenti ed altre operazioni)

### Differenze importanti:

- Si usano le parentesi tonde () invece delle quadre []
- Il primo elemento di **nomeVettore** è alla posizione **1** (l'indice deve essere sempre positivo)



## Accesso ed Assegnamento

È possibile **modificare** un valore in un **vettore**

1. Accedendo all'elemento del vettore
2. Assegnando un nuovo valore nella posizione specifica

```
>> a = [1 : 3]
```

```
a =
```

```
    1    2    3
```

```
>> a(3) = 6
```

```
a =
```

```
    1    2    6
```



## Accesso ed Assegnamento

È possibile **modificare** un valore in un **vettore**

1. Accedendo all'elemento del vettore
2. Assegnando un nuovo valore nella posizione specifica

```
>> a = [1 : 3]
```

```
a =
```

```
1    2    3
```

```
>> a(3) = 6
```

```
a =
```

```
1    2    6
```

È possibile eseguire l'**assegnamento tra vettori**, anche quando i due vettori non hanno le stesse dimensioni: il vettore a cui viene assegnato il valore viene ridefinito

```
>> b = [1 : 4]
```

```
b =
```

```
1    2    3    4
```

```
>> a = b
```

```
a =
```

```
1    2    3    4
```



## Accedere agli Elementi di una Array

Viene segnalato un **errore** quando si **accede** ad una **posizione che non corrisponde ad un elemento** dell'array (vale anche per matrici e array multidimensionali)

```
>> a = [1 : 3]
```

```
a =
```

```
    1    2    3
```

```
>> a(2)
```

```
ans =
```

```
    2
```



## Accedere agli Elementi di una Array

Viene segnalato un **errore** quando si **accede** ad una **posizione che non corrisponde ad un elemento** dell'array (vale anche per matrici e array multidimensionali)

```
>> a = [1 : 3]
```

```
a =
```

```
    1    2    3
```

```
>> a(2)
```

```
ans =
```

```
    2
```

```
>> a(4)
```

```
Index exceeds matrix dimensions
```



## Accedere agli Elementi di una Array

Viene segnalato un **errore** quando si **accede** ad una **posizione che non corrisponde ad un elemento** dell'array (vale anche per matrici e array multidimensionali)

```
>> a = [1 : 3]
```

```
a =
```

```
    1    2    3
```

```
>> a(2)
```

```
ans =
```

```
    2
```

```
>> a(1.3)
```

**Subscript indices must either  
be real positive integers or  
logicals**





## Accedere agli Elementi di una Array

Viene segnalato un **errore** quando si **accede** ad una **posizione che non corrisponde ad un elemento** dell'array (vale anche per matrici e array multidimensionali)

```
>> a = [1 : 3]
```

```
a =
```

```
    1    2    3
```

```
>> ii = 2; ←
```

```
>> a(ii)
```

```
ans =
```

```
    2
```

```
>> a(ii) = a(ii - 1) + a(ii + 1)
```

```
a =
```

```
    1    4    3
```

È possibile utilizzare una variabile per definire l'indice, come in C



## Operazioni Aritmetiche tra Vettori

Le **operazioni aritmetiche** sono quelle **dell'algebra lineare**

- La somma tra vettori  $\mathbf{c} = \mathbf{a} + \mathbf{b}$  è definita elemento per elemento

$$c(i) = a(i) + b(i), \quad \forall i$$

è possibile solo quando  $\mathbf{a}$  e  $\mathbf{b}$  hanno la stessa dimensione (che poi coincide con quella di  $\mathbf{c}$ )



## Operazioni Aritmetiche tra Vettori

Le **operazioni aritmetiche** sono quelle **dell'algebra lineare**

- La somma tra vettori  $\mathbf{c} = \mathbf{a} + \mathbf{b}$  è definita elemento per elemento

$$c(i) = a(i) + b(i), \quad \forall i$$

è possibile solo quando  $\mathbf{a}$  e  $\mathbf{b}$  hanno la stessa dimensione (che poi coincide con quella di  $\mathbf{c}$ )

- Prodotto tra vettori è il prodotto riga per colonna, restituisce uno scalare

$$\mathbf{c} = \mathbf{a} * \mathbf{b}, \text{ i.e. } c = \sum_i a(i)b(i)$$

$\mathbf{a}$  deve essere un vettore riga e  $\mathbf{b}$  colonna e devono avere lo stesso numero di elementi,  $\mathbf{c}$  è un numero reale



## Operazioni Puntuali

E' possibile eseguire operazioni **puntuali**, che si applicano cioè ad ogni elemento del vettore separatamente

$c = a \ . \ * \ b$ , restituisce  $c(i) = a(i) * b(i) \ \forall i$

$c = a \ . \ / \ b$ , restituisce  $c(i) = a(i) / b(i) \ \forall i$

$c = a \ . \ ^ \ b$ , restituisce  $c(i) = a(i)^{b(i)} \ \forall i$



## Operazioni Puntuali

E' possibile eseguire operazioni **puntuali**, che si applicano cioè ad ogni elemento del vettore separatamente

$\mathbf{c} = \mathbf{a} \ . \ * \ \mathbf{b}$ , restituisce  $c(i) = a(i) * b(i) \ \forall i$

$\mathbf{c} = \mathbf{a} \ . \ / \ \mathbf{b}$ , restituisce  $c(i) = a(i)/b(i) \ \forall i$

$\mathbf{c} = \mathbf{a} \ . \ ^ \ \mathbf{b}$ , restituisce  $c(i) = a(i)^{b(i)} \ \forall i$

Come in algebra lineare, le **operazioni tra vettori (array) e scalari** sono possibili, e corrispondono ad operazioni puntuali.

Se  $\mathbf{k}$  è uno scalare e  $\mathbf{b}$  è un vettore

$\mathbf{c} = \mathbf{k} * \mathbf{b} = \mathbf{k} \ . \ * \ \mathbf{b} \quad c(i) = k * b(i) \ \forall i$



## Attenzione: Elevamento a Potenza

```
>> v1 = [2      3      5      4]
```

```
>> v1^2
```

Error using ^

Inputs must be a scalar and a square matrix.  
To compute elementwise POWER, use POWER (.^)  
instead.

L'elevamento a potenza fa' riferimento al prodotto vettoriale  
(equivale a  $\mathbf{v1} * \mathbf{v1}$  che vale solo per matrici quadrate )



## Attenzione: Elevamento a Potenza

```
>> v1 = [2      3      5      4]
```

```
>> v1^2
```

Error using ^

Inputs must be a scalar and a square matrix.  
To compute elementwise POWER, use POWER (.^)  
instead.

L'elevamento a potenza fa' riferimento al prodotto vettoriale  
(equivale a  $\mathbf{v1} * \mathbf{v1}$  che vale solo per matrici quadrate )

Per elevare a potenza ogni singolo elemento di  $\mathbf{v1}$  si usa:

```
>> v1.^2
```

```
ans =
```

```
4      9     25     16
```

che equivale a fare  $\mathbf{v1} .* \mathbf{v1}$



# Operazioni Aritmetiche su Array

Operazione	Sintassi Matlab	Commenti
Array addition	$a + b$	Array e matrix addition sono identiche
Array subtraction	$a - b$	Array e matrix subtraction sono identiche
Array multiplication	$a .* b$	Ciascun elemento del risultato è pari al prodotto degli elementi corrispondenti nei due operandi
Matrix multiplication	$a * b$	Prodotto righe per colonne dell'algebra lineare
Array right division	$a ./ b$	$\text{risultato}(i,j) = a(i,j)/b(i,j)$
Array left division	$a .\backslash b$	$\text{risultato}(i,j) = b(i,j)/a(i,j)$
Matrix right division	$a / b$	$a * \text{inversa}(b)$
Matrix left division	$a \backslash b$	$\text{inversa}(a) * b$
Array exponentiation	$a .^ b$	$\text{risultato}(i,j) = a(i,j)^b(i,j)$





## Concatenare i Vettori

L'operatore , e ; permettono di concatenare vettori, purché le dimensioni siano compatibili (devono essere entrambi riga o colonna).

Esempio:

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

```
a =
```

1

2

3



## Concatenare i Vettori

L'operatore , e ; permettono di concatenare vettori, purché le dimensioni siano compatibili (devono essere entrambi riga o colonna).

Esempio:

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

```
a =
```

```
    1    2    3
```

```
>> b = [a, a + 3 , a + 6]
```

```
b =
```

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



## Concatenare i Vettori

L'operatore `,` e `;` permettono di concatenare vettori, purché le dimensioni siano compatibili (devono essere entrambi riga o colonna).

Esempio:

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

```
a =
```

```
    1    2    3
```

```
>> b = [a, a + 3, a + 6]
```

```
b =
```

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

```
>> b = [a, a + 3]
```

```
b =
```

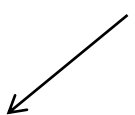

```
 1  2  3  1  2  3  3
```

Viene interpretato  
come `b = [a, a, +3]`  
ATTENZIONE agli spazi



# Concatenare i Vettori

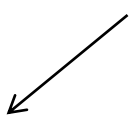

Esempi

- $a = [0 \ 7+1] ;$   contenuto di a
- $b = [a(2) \ 5 \ a] ;$   secondo elemento di a



# Concatenare i Vettori

## Esempi

- $a = [0 \ 7+1] ;$   contenuto di a
- $b = [a(2) \ 5 \ a] ;$   secondo elemento di a

## Risultato

- $a = [0 \ 8]$
- $b = [8 \ 5 \ 0 \ 8]$



# Stringhe

Come in C sono array di caratteri

I valori vengono assegnati mediante apici singoli ' '

Esempi:

```
>> msg = 'ciao mamma';
```

```
>> msg = [msg , ' torno per cena']
```

```
msg =
```

```
ciao mamma torno per cena
```

```
>> msg(1) = 'C'
```

```
msg =
```

```
Ciao mamma torno per cena
```



Le matrici vengono definite **affiancando vettori di dimensioni compatibili**

- Usiamo sempre gli operatori , (spazio) e ; (vai a capo)
- L'operazione di **trasposizione** inverte le righe e le colonne della matrice



## Le Matrici

Le matrici vengono definite **affiancando vettori di dimensioni compatibili**

- Usiamo sempre gli operatori , (spazio) e ; (vai a capo)
- L'operazione di **trasposizione** inverte le righe e le colonne della matrice

Es :

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

a =

1	2
3	4

a' =

1	3
2	4





## Le Matrici: operatore CAT

La concatenazione dei vettori avviene mediante operatore **CAT** che richiede **dimensioni consistenti** dei vettori

```
>> a = [1 : 3]
```

```
a =
```

```
    1    2    3
```

```
>> b = [4; 5; 6]
```

```
b =
```

```
    4
```

```
    5
```

```
    6
```



## Le Matrici: operatore CAT

La concatenazione dei vettori avviene mediante operatore **CAT** che richiede **dimensioni consistenti** dei vettori

```
>> a = [1 : 3]
```

```
a =
```

```
1      2      3
```

```
>> A = [a; b]
```

```
Error using vertcat
```

```
CAT arguments dimensions are  
not consistent.
```

```
>> b = [4; 5; 6]
```

```
b =
```

```
4
```

```
5
```

```
6
```



## Le Matrici: operatore CAT

La concatenazione dei vettori avviene mediante operatore **CAT** che richiede **dimensioni consistenti** dei vettori

```
>> a = [1 : 3]
```

```
a =
```

```
1      2      3
```

```
>> A = [a; b]
```

Error using vertcat

CAT arguments dimensions are not consistent.

```
>> b = [4; 5; 6]
```

```
b =
```

```
4
```

```
5
```

```
6
```

```
>> A = [a, b]
```

Error using horzcat

CAT arguments dimensions are not consistent.



## Le Matrici: operatore CAT

La concatenazione dei vettori avviene mediante operatore **CAT** che richiede **dimensioni consistenti** dei vettori

```
>> a = [1 : 3]
```

```
a =
```

```
1      2      3
```

```
>> A = [a; b]
```

Error using vertcat

CAT arguments dimensions are not consistent.

```
>> b = [4; 5; 6]
```

```
b =
```

```
4
```

```
5
```

```
6
```

```
>> A = [a, b]
```

Error using horzcat

CAT arguments dimensions are not consistent.

```
>> A = [a; b']
```

```
A =
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```



## Accedere agli Elementi di una Matrice

Per accedere agli elementi di una matrice occorre specificare un valore per ogni indice

`nomeMatrice(indice1, indice2)`

Seleziona il valore alla riga `indice1` colonna `indice2` nella variabile `nomeMatrice`

Es

```
>> A = [1 : 3; 4 : 6; 7: 9 ]
```

```
A =
```

1	2	3
4	5	6
7	8	9

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

```
ans =
```

```
6
```

```
>> A(3,5)
```

```
Index exceeds  
matrix dimensions.
```



# Operazioni Aritmetiche con Matrici

## Operazioni per gli array

- Array operation: eseguita sugli elementi corrispondenti degli array coinvolti (devono avere lo stesso numero di righe e colonne); si indica aggiungendo un punto prima dell'operatore aritmetico

- $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad b = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix} \quad a .* b = \begin{bmatrix} 2 & 6 \\ 15 & 28 \end{bmatrix}$



# Operazioni Aritmetiche con Matrici

## Operazioni per gli array

- Array operation: eseguita sugli elementi corrispondenti degli array coinvolti (devono avere lo stesso numero di righe e colonne); si indica aggiungendo un punto prima dell'operatore aritmetico

- $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad b = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix} \quad a .* b = \begin{bmatrix} 2 & 6 \\ 15 & 28 \end{bmatrix}$

- Matrix operation: segue le regole dell'algebra lineare (prodotto righe per colonne)

- $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad b = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix} \quad a * b = \begin{bmatrix} 12 & 17 \\ 26 & 37 \end{bmatrix}$

$$(a * b)_{ij} = \sum_k a_{ik} * b_{kj}$$



## Definizione ed Estensione Automatica di Array

A differenza del C, un **assegnamento** in una posizione in cui il vettore non è definito (invece di avere segmentation fault) **estende l'array** inserendo 0

```
>> c = 1
```

```
c =
```

```
1
```





## Definizione ed Estensione Automatica di Array

A differenza del C, un **assegnamento** in una posizione in cui il vettore non è definito (invece di avere segmentation fault) **estende l'array** inserendo 0

```
>> c = 1
```

```
c =
```

```
1
```

```
>> c(3) = 3
```

```
c =
```

```
1    0    3
```



## Definizione ed Estensione Automatica di Array

A differenza del C, un **assegnamento** in una posizione in cui il vettore non è definito (invece di avere segmentation fault) **estende l'array** inserendo 0

```
>> c = 1
```

```
c =
```

```
1
```

```
>> c(3) = 3
```

```
c =
```

```
1    0    3
```

```
>> c(2,3) = 5
```

```
c =
```

```
1    0    3
```

```
0    0    5
```



## Definizione ed Estensione Automatica di Array

A differenza del C, un **assegnamento** in una posizione in cui il vettore non è definito (invece di avere segmentation fault) **estende l'array** inserendo 0

```
>> c = 1
```

```
c =
```

```
1
```

```
>> c(3) = 3
```

```
c =
```

```
1    0    3
```

```
>> c(2,3) = 5
```

```
c =
```

```
1    0    3
```

```
0    0    5
```

**N.B.** Assegnare un valore ad un elemento è diverso da accedere

>> c(5,8) ERRORE!



## Array Multidimensionali

È possibile specificare una terza (quarta, quinta...) dimensione lungo la quale indicizzare un array.



## Array Multidimensionali

È possibile specificare una terza (quarta, quinta...) dimensione lungo la quale indicizzare un array.

Ad esempio le immagini a colori sono definite con tre piani colore (RGB), quindi

- un'immagine a colori 10 Mpixels, aspect ratio (3:4) richiede una matrice 3D di  $2736 \times 3648 \times 3$  elementi



## Array Multidimensionali

È possibile specificare una terza (quarta, quinta...) dimensione lungo la quale indicizzare un array.

Ad esempio le immagini a colori sono definite con tre piani colore (RGB), quindi

- un'immagine a colori 10 Mpixels, aspect ratio (3:4) richiede una matrice 3D di  $2736 \times 3648 \times 3$  elementi
- 10 sec di video full HD ( $1080 \times 768$ ) a 24fps richiede una matrice 4D di  $1080 \times 768 \times 3 \times (10 \times 24)$  elementi



## Esempi di Operazioni su Matrici

<code>a=[1 2; 3, 4]</code>	<code>a = 1 2 3 4</code>	a ora è una matrice 2x2, “;” separa le righe; virgola (opzionale) separa elementi
<code>a</code>	<code>a = 1 2 3 4</code>	restituisce il valore della variabile a
<code>x=[-1.3 sqrt(3) (1+2)/5]</code>	<code>x = -1.30000 1.73205 0.60000</code>	Elementi del vettore possono essere espressioni aritmetiche
<code>x(5)= abs(x(1))</code>	<code>x = -1.30000 1.73205 0.60000 0.00000 1.30000</code>	Notazione con () per accedere a elementi di un array; abs valore assoluto; NB: vettore x esteso per includere nuovo elemento; elementi non assegnati sono nulli



## Esempi di Operazioni su Matrici

<b><math>b=a'</math></b>	<b><math>b =</math> 1 3 2 4</b>	<b>matrice trasposta (scambiate righe e colonne)</b>
<b><math>c=a+b</math></b>	<b><math>c =</math> 2 5 5 8</b>	<b>somma di matrici, elemento per elemento (sottrazione con “-” simile)</b>
<b><math>x=[-1 \ 0 \ 2];</math> <math>y=x'</math></b>	<b><math>y =</math> -1 0 2</b>	<b>il “;” blocca l’output, ma non impedisce la valutazione</b>





# Le Variabili ed i Tipi



## Il Workspace

Tutte le variabili vengono salvate nel workspace, che corrisponde alla memoria



## Il Workspace

Tutte le variabili vengono salvate nel workspace, che corrisponde alla memoria

E' possibile visualizzare le variabili ed il workspace:

- Il comando **whos** (visualizza tutte le variabili)
- Il comando **whos nomeVariabile** (visualizza solo **nomeVariabile**)
- Il pannello del Workspace



## Il Workspace

Tutte le variabili vengono salvate nel workspace, che corrisponde alla memoria

E' possibile visualizzare le variabili ed il workspace:

- Il comando **whos** (visualizza tutte le variabili)
- Il comando **whos nomeVariabile** (visualizza solo **nomeVariabile**)
- Il pannello del Workspace

Per pulire il workspace e rimuovere tutte le variabili presenti si usa il comando: **clear**

```
>> clear
```



## Tipo Double

Di default, valori numerici danno luogo a variabili di tipo **double**: un double contiene uno scalare espresso con doppia precisione (**64 bit**)



## Tipo Double

Di default, valori numerici danno luogo a variabili di tipo **double**: un double contiene uno scalare espresso con doppia precisione (**64 bit**)

È possibile vedere il tipo delle variabili mediante **whos**

**whos nomeVariabile**

```
>> a = 7;
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x1	8	double	



## Tipo Char

Una variabile di tipo char contiene uno **scalare** o un **array** di valori a 16 bit (8 bit in Octave), ciascuno dei quali rappresenta un carattere

- Es: `frase = 'questa e` una stringa';`



Nome della variabile



Array di 1x21 caratteri

- NB: stringhe racchiuse tra apici ***singoli***
- `whos frase;`

Name	Size	Bytes	Class	Attributes
<code>frase</code>	<code>1x21</code>	<code>42</code>	<code>char</code>	



## Tipo Complex

In Matlab è possibile rappresentare anche numeri **complessi**

parti reali e immaginarie possono essere positive e negative

```
>> a = [sqrt(-1) 7]
a =
    0 + 1.0000i    7.0000
```

```
>> whos
```

Name	Size	Bytes	Class
a	1x2	32	double

Attributes complex
-----------------------





## Tipo Complex

I double possono essere utilizzati per esprimere numeri

- Reali, es  $\text{var1} = -10.7$ ;
- Immaginari, es  $\text{var2} = 4i$ ;  $\text{var3} = 4j$ ;
- Complessi, es  $\text{var3} = 10.3 + 10i$ ;

Es:  $x = [-1.3 \quad 3.1+5.3j \quad 0]$

**NB Meglio non usare mai i e j come nome di variabile**



## Gestione Dinamica delle Variabili

I tipi delle variabili possono cambiare:

- mediante conversione esplicita
- mediante assegnamento: **il tipo** di una variabile è **definito dal valore** contenuto

```
>> a = [1 3 5] .^ (0.2)
```

```
a =
```

```
    1.0000    1.2457    1.3797
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x3	24	double	

```
>> a = 'cia';
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x3	6	char	



# Altre Operazioni sugli Array

SubArray e cancellazione elementi



## Sottoarray (un vettore come indice di un vettore)

Estende l'accesso ad un singolo elemento

**nomeVettore (indice)**

Si denota un sottoinsieme di un array usando vettori per valori degli indici

**nomeVettore (vettoreIndici)**

restituisce un vettore che comprende gli elementi di **nomeVettore** che compaiono nelle posizioni **vettoreIndici**



## Esempi

```
> v=[6 8 4 2 4 5 1 3]
```

```
v = 6      8      4      2      4      5      1      3
```

primo, quarto  
settimo elemento

```
>> v([1 4 7])
```

```
ans =      6      2      1
```

2:2:6 è il vettore [2, 4, 6]:  
indica secondo, quarto,  
sesto elemento

```
>> v(2:2:6)
```

```
ans =      8      2      5
```



## Sottovettori definiti da vettori di indici

Quindi, dato un vettore  $\mathbf{v}$ , la notazione

$$\mathbf{a}(\mathbf{v})$$

corrisponde a

$$[\mathbf{a}(\mathbf{v}(1)), \mathbf{a}(\mathbf{v}(2)), \dots, \mathbf{a}(\mathbf{v}(\text{end}))]$$

Attenzione che i valori di  $\mathbf{v}$  devono essere interi positivi e minori delle dimensioni di  $\mathbf{a}$  ..... devono essere indici validi.



## La Keyword **end**

All'interno di **vettoreIndici** si può usare la keyword **end** che assume il valore dell'ultimo indice disponibile su una specifica dimensione di **nomeVettore**.

In questo modo non occorre conoscere le dimensioni del vettore



## La Keyword **end**

All'interno di **vettoreIndici** si può usare la keyword **end** che assume il valore dell'ultimo indice disponibile su una specifica dimensione di **nomeVettore**.

In questo modo non occorre conoscere le dimensioni del vettore

Esempi

```
>> a = [1 : 10]
```

```
a =
```

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





## La Keyword **end**

All'interno di **vettoreIndici** si può usare la keyword **end** che assume il valore dell'ultimo indice disponibile su una specifica dimensione di **nomeVettore**.

In questo modo non occorre conoscere le dimensioni del vettore

Esempi

```
>> a = [1 : 10]
```

```
a =
```

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

```
>> b = a(1 : end - 1)
```

```
b =
```

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

Toglie l'ultimo  
elemento



## La Keyword **end**

All'interno di **vettoreIndici** si può usare la keyword **end** che assume il valore dell'ultimo indice disponibile su una specifica dimensione di **nomeVettore**.

In questo modo non occorre conoscere le dimensioni del vettore

Esempi

```
>> a = [1 : 10]
```

```
a =
```

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

Toglie l'ultimo  
elemento

```
>> b = a(1 : end - 1)
```

```
b =
```

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

Legge il vettore  
dall'ultimo elemento  
al primo

```
>> b = a(end : -1 : 1)
```

```
b =
```

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



## Esempi

```
> v=[6 8 4 2 4 5 1 3]
```

```
v =      6      8      4      2      4      5      1      3
```

```
>> v([1 4 7])
```

```
>> v(2:2:6)
```

```
>> v(3:end-2)
```

```
>> v(v)
```

```
>> v([1, 1, 1, 2, end])
```



## Esempi

```
> v=[6 8 4 2 4 5 1 3]
```

```
v =      6      8      4      2      4      5      1      3
```

```
>> v([1 4 7])
```

```
ans =      6      2      1
```

primo, quarto settimo elemento

```
>> v(2:2:6)
```

```
>> v(3:end-2)
```

```
>> v(v)
```

```
>> v([1, 1, 1, 2, end])
```



## Esempi

```
> v=[6 8 4 2 4 5 1 3]
```

```
v =      6      8      4      2      4      5      1      3
```

```
>> v([1 4 7])
```

```
ans =      6      2      1
```

primo, quarto settimo elemento

```
>> v(2:2:6)
```

```
ans =      8      2      5
```

2:2:6 è il vettore [2, 4, 6]: indica  
secondo, quarto, sesto elemento

```
>> v(3:end-2)
```

```
>> v(v)
```

```
>> v([1, 1, 1, 2, end])
```



## Esempi

```
> v=[6 8 4 2 4 5 1 3]
```

```
v =      6      8      4      2      4      5      1      3
```

```
>> v([1 4 7])
```

```
ans =      6          2          1
```

primo, quarto settimo elemento

```
>> v(2:2:6)
```

```
ans =      8          2          5
```

2:2:6 è il vettore [2, 4, 6]: indica  
secondo, quarto, sesto elemento

```
>> v(3:end-2)
```

```
ans =      4          2          4          5
```

dal terzo al terz'ultimo elemento

```
>> v(v)
```

```
>> v([1, 1, 1, 2, end])
```



## Esempi

```
> v=[6 8 4 2 4 5 1 3]
```

```
v =      6      8      4      2      4      5      1      3
```

```
>> v([1 4 7])
```

```
ans =      6          2          1
```

primo, quarto settimo elemento

```
>> v(2:2:6)
```

```
ans =      8          2          5
```

2:2:6 è il vettore [2, 4, 6]: indica  
secondo, quarto, sesto elemento

```
>> v(3:end-2)
```

```
ans =      4          2          4          5
```

dal terzo al terz'ultimo elemento

```
>> v(v)
```

```
ans =      5      3      2      8      2      4      6      4
```

i *valori* di v usati come indice (!!!!)

```
>> v([1, 1, 1, 2, end])
```



## Esempi

```
> v=[6 8 4 2 4 5 1 3]
```

```
v =      6      8      4      2      4      5      1      3
```

```
>> v([1 4 7])
```

```
ans =      6          2          1
```

primo, quarto settimo elemento

```
>> v(2:2:6)
```

```
ans =      8          2          5
```

2:2:6 è il vettore [2, 4, 6]: indica  
secondo, quarto, sesto elemento

```
>> v(3:end-2)
```

```
ans =      4          2          4          5
```

dal terzo al terz'ultimo elemento

```
>> v(v)
```

```
ans =      5      3      2      8      2      4      6      4
```

i *valori* di v usati come indice (!!!!)

```
>> v([1, 1, 1, 2, end])
```

```
ans = 6      6      6      8      3
```

Indici ripetuti fanno replicare i  
valori nel sottovettore





## Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

$$\mathbf{v1}(\mathbf{vettoreIndici}) = \mathbf{v2}$$

Viene però richiesto che  $\mathbf{v2}$  abbia **le stesse dimensioni** di  $\mathbf{v1}(\mathbf{vettoreIndici})$



## Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

$$\mathbf{v1}(\mathbf{vettoreIndici}) = \mathbf{v2}$$

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a = [1 : 10]
```

```
a =
```

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



## Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

`v1(vettoreIndici) = v2`

Viene però richiesto che `v2` abbia **le stesse dimensioni** di `v1(vettoreIndici)`

```
>> a = [1 : 10]
```

```
a =
```

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

```
>> a(1 : 3) = [0 0 0]
```

```
a =
```

```
0 0 0 4 5 6 7 8 9 10
```



## Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

$$\mathbf{v1}(\mathbf{vettoreIndici}) = \mathbf{v2}$$

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a =
```

```
    0  0  0  4  5  6  7  8  9 10
```

```
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
```



## Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

$$\mathbf{v1}(\mathbf{vettoreIndici}) = \mathbf{v2}$$

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a =  
      0  0  0  4  5  6  7  8  9 10  
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)  
a =  
      0  0  0  8  5 12  7 16  9 20
```



## Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

$$\mathbf{v1}(\mathbf{vettoreIndici}) = \mathbf{v2}$$

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a =  
      0  0  0  4  5  6  7  8  9 10  
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)  
a =  
      0  0  0  8  5 12  7 16  9 20  
>> a(1 : 2 : end) = a(end : -2 : 1)
```



## Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

$$\mathbf{v1}(\mathbf{vettoreIndici}) = \mathbf{v2}$$

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a =  
      0  0  0  4  5  6  7  8  9 10  
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)  
a =  
      0  0  0  8  5 12  7 16  9 20  
>> a(1 : 2 : end) = a(end : -2 : 1)
```



## Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

`v1(vettoreIndici) = v2`

Viene però richiesto che `v2` abbia **le stesse dimensioni** di `v1(vettoreIndici)`

```
>> a =  
      0  0  0  4  5  6  7  8  9 10  
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)  
a =  
      0  0  0  8  5 12  7 16  9 20  
>> a(1 : 2 : end) = a(end : -2 : 1)  
                        20 16 12 8 0
```





## Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

$$\mathbf{v1}(\mathbf{vettoreIndici}) = \mathbf{v2}$$

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a =
```

```
0 0 0 4 5 6 7 8 9 10
```

```
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
```

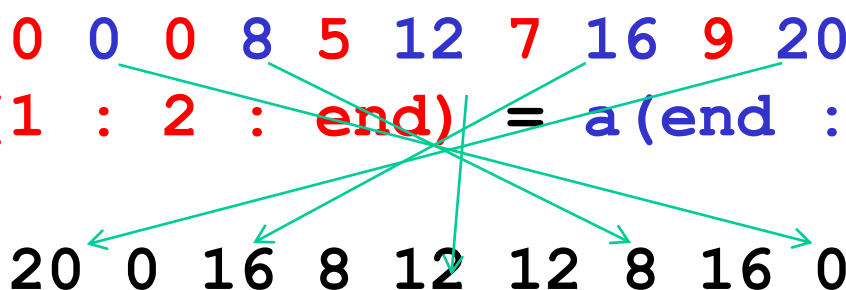
```
a =
```

```
0 0 0 8 5 12 7 16 9 20
```

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

```
a =
```

```
20 0 16 8 12 12 8 16 0 20
```





## Sottoarray: Applicazione a Matrici

Si denota un sottoinsieme di un array usando vettori per valori degli indici

**nomeMatrice(vettore1, vettore2)**

restituisce una matrice che comprende gli elementi di **nomeMatrice** alle righe di indice in **vettore1** e alle colonne di indice in **vettore2**.



## Sottoarray: Applicazione a Matrici

m =	9	8	7
	6	5	4
	3	2	1
	0	11	12
	0	0	0

```
>> m([1 4], [2 3])
```



## Sottoarray: Applicazione a Matrici

m =	9	8	7
	6	5	4
	3	2	1
	0	11	12
	0	0	0

```
>> m([1 4], [2 3])  
ans = 8 7  
      11 12
```

tutti gli elementi sulle  
righe 1 e 4 e sulle colonne 2 e 3



## Sottoarray: Applicazione a Matrici

```
m = 9      8      7
     6      5      4
     3      2      1
     0     11     12
     0      0      0
```

```
>> m([1 4], [2 3])
ans = 8      7
      11     12
```

tutti gli elementi sulle  
righe 1 e 4 e sulle colonne 2 e 3

```
>> m(1:2:5, 1:end)
```



## Sottoarray: Applicazione a Matrici

```
m = 9      8      7
     6      5      4
     3      2      1
     0     11     12
     0      0      0
```

```
>> m([1 4], [2 3])
```

```
ans = 8      7
      11     12
```

tutti gli elementi sulle  
righe 1 e 4 e sulle colonne 2 e 3

```
>> m(1:2:5, 1:end)
```

```
ans = 9      8      7
      3      2      1
      0      0      0
```

tutti gli elementi delle  
righe 1, 3 e 5



## Sottoarray: Applicazione a Matrici

```
m = 9      8      7  
      6      5      4  
      3      2      1  
      0      11     12  
      0      0      0
```

```
>> m(1:2:5, :)
```

```
ans = 9      8      7  
      3      2      1  
      0      0      0
```

notazione ':' abbreviata per 1:end,  
cioè tutti i valori di quell'indice



## Sottoarray: Applicazione a Matrici

```
m = 9      8      7
     6      5      4
     3      2      1
     0     11     12
     0      0      0
```

```
>> m(1:2:5, :)
```

```
ans = 9      8      7
       3      2      1
       0      0      0
```

notazione ':' abbreviata per 1:end,  
cioè tutti i valori di quell'indice

```
>> m(2:2:4, :) = [-1 -2 -3; -4 -5 -6]
```

```
m = 9      8      7
     -1     -2     -3
       3      2      1
     -4     -5     -6
       0      0      0
```

uso della notazione dei sottoarray  
per individuare elementi oggetto di  
assegnamento





## Esempio

```
% inizializzare una matrice 5x5 con tutti valori a zero  
A(5,5) = 0;
```



## Esempio

```
% inizializzare una matrice 5x5 con tutti valori a zero  
A(5,5) = 0;  
% modificare la colonna centrale in tutti 1  
A(:, 3) = 1;
```



## Esempio

```
% inizializzare una matrice 5x5 con tutti valori a zero  
A(5,5) = 0;  
% modificare la colonna centrale in tutti 1  
A(:, 3) = 1;  
% modificare la riga centrale in tutti 3  
A(3, :) = 3;
```



## Esempio

```
% inizializzare una matrice 5x5 con tutti valori a zero  
A(5,5) = 0;  
% modificare la colonna centrale in tutti 1  
A(:, 3) = 1;  
% modificare la riga centrale in tutti 3  
A(3, :) = 3;  
% sommare 2 ai valori della colonna centrale  
A(:, 3) = A(:, 3) + 2; % NB termini a dx e sx  
dell'uguale hanno la stessa dimensione
```



## Esempio

```
% inizializzare una matrice 5x5 con tutti valori a zero
A(5,5) = 0;
% modificare la colonna centrale in tutti 1
A(:, 3) = 1;
% modificare la riga centrale in tutti 3
A(3, :) = 3;
% sommare 2 ai valori della colonna centrale
A(:, 3) = A(:, 3) + 2; % NB termini a dx e sx
dell'uguale hanno la stessa dimensione
% porre a 2 gli elementi nel primo quadrante
A(1 : 2, 1 : 2) = 2;
```



## Esempio

```
% inizializzare una matrice 5x5 con tutti valori a zero
A(5,5) = 0;
% modificare la colonna centrale in tutti 1
A(:, 3) = 1;
% modificare la riga centrale in tutti 3
A(3, :) = 3;
% sommare 2 ai valori della colonna centrale
A(:, 3) = A(:, 3) + 2; % NB termini a dx e sx
dell'uguale hanno la stessa dimensione
% porre a 2 gli elementi nel primo quadrante
A(1 : 2, 1 : 2) = 2;
% copiare nell'ultima riga la prima riga letta al
contrario
A(end, :) = A(1, end : -1 : 1)
```



## Assegnamenti con Scalari

È possibile associare a qualsiasi sotto array un valore scalare

`nomeVettore(vettoreIndici) = k`

Fa sì che a tutti gli elementi di `nomeVettore` alle posizioni `vettoreIndici` venga assegnato il valore `k`



## Assegnamenti con Scalari

È possibile associare a qualsiasi sotto array un valore scalare

`nomeVettore(vettoreIndici) = k`

Fa sì che a tutti gli elementi di `nomeVettore` alle posizioni `vettoreIndici` venga assegnato il valore `k`

In questo modo è possibile inizializzare nuovi vettori.

```
>> a = [1 : 10]
```

```
a =
```

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

```
>> a(1 : 3) = 0
```

```
a =
```

```
0  0  0  4  5  6  7  8  9 10
```





## Array Vuoto

Un array vuoto si definisce così:

```
nomeVettore = []
```

Può essere una forma di dichiarazione di una variabile



## Array Vuoto

Un array vuoto si definisce così:

```
nomeVettore = []
```

Può essere una forma di dichiarazione di una variabile

```
>> a = []
```

```
a =
```

```
[]
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	0x0	0	double	



## Cancellare Parti di un Vettore

Quando si assegna il valore [] ad un elemento di un vettore, il corrispondente elemento viene rimosso e il vettore ridimensionato: non si crea un 'buco'

```
>> a = [1 : 5]
```

```
a =
```

1

2

3

4

5

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x5	40	double	



## Cancellare Parti di un Vettore

Quando si assegna il valore [] ad un elemento di un vettore, il corrispondente elemento viene rimosso e il vettore ridimensionato: non si crea un 'buco'

```
>> a = [1 : 5]
```

```
a =
```

```
      1      2      3      4      5
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x5	40	double	

```
>> a(3) = []
```

```
a =
```

```
      1      2      4      5
```

```
>> whos a
```

Name	Size	Bytes	Class	Attributes
a	1x4	32	double	



## Cancellare Parti di una Matrice

L'array vuoto [] non è assegnabile a singoli elementi di matrici (non si possono “creare buchi”)

```
>> m[1 : 3, 1:3] = 1
```

```
m =
```

1	1	1
1	1	1
1	1	1



## Cancellare Parti di una Matrice

L'array vuoto [] non è assegnabile a singoli elementi di matrici (non si possono “creare buchi”)

```
>> m[1 : 3, 1:3] = 1
```

```
m =
```

1	1	1
1	1	1
1	1	1

```
>> m(3,3)=[ ]
```

```
??? Subscripted assignment dimension  
mismatch.
```



## Cancellare Parti di una Matrice

È però assegnabile a intere righe o colonne di matrici, che vengono cancellate (ricompattando la matrice)

```
>> m(:, 2) = []
```

```
m =
```

```
    1    1  
    1    1  
    1    1
```

```
>> whos m
```

Name	Size	Bytes	Class	Attributes
m	3x2	48	double	



# Variabili Predefinite





## Variabili predefinite

Matlab definisce un insieme di variabili predefinite (es, `pi`)  
Queste variabili spesso rappresentano importanti costanti della matematica (`pi` è pigreco, `i` e `j` sono  $\sqrt{-1}$  )



## Variabili predefinite

Matlab definisce un insieme di variabili predefinite (es, `pi`)

Queste variabili spesso rappresentano importanti costanti della matematica (`pi` è pigreco, `i` e `j` sono  $\sqrt{-1}$  )

- Attenzione! Il valore di queste variabili può essere modificato, per esempio
  - `circ1=2*pi*10;`
  - `pi = 3;`
  - `circ2=2*pi*10;`
- Il valore di `circ2` non sarà più la circonferenza di un cerchio



## Variabili predefinite

Matlab definisce un insieme di variabili predefinite (es, `pi`)  
Queste variabili spesso rappresentano importanti costanti della matematica (`pi` è pigreco, `i` e `j` sono  $\sqrt{-1}$  )

- Attenzione! Il valore di queste variabili può essere modificato, per esempio
  - `circ1=2*pi*10;`
  - `pi = 3;`
  - `circ2=2*pi*10;`
- Il valore di `circ2` non sarà più la circonferenza di un cerchio

E` fortemente sconsigliato modificare il valore di una variabile predefinita ( $\Rightarrow$  evitare di usare variabili `i` e `j` come contatori)



## Variabili predefinite più comuni

Variabile	Scopo
pi	contiene 15 cifre significative di $\pi$
i, j, 1i, 1j	contiene il valore i ( $\sqrt{-1}$ )
inf (o Inf)	rappresentazione dell'infinito (ottenuto di solito come risultato di una divisione per 0)
NaN, nan	Not-A-Number è il risultato di una operazione matematica non definita, es 0/0
clock	contiene la data e l'orario corrente. E` un vettore di sei valori numerici (anno, mese, giorno, ora, minuti, secondi)
date	contiene la data corrente sotto forma di stringa (es. 03-Jan-2017)
eps	epsilon: la più piccola differenza rappresentabile tra due numeri ( $2^{-52}$ )
ans	Variabile speciale usata per immagazzinare risultati non assegnati ad altre variabili



# Input/Output



## Acquisizione Dati da Tastiera (input)

Funzione input

```
valore = input(stringaDaVisualizzare) ;
```



## Acquisizione Dati da Tastiera (input)

Funzione input

```
valore = input(stringaDaVisualizzare) ;
```

Matlab stampa a video la **stringaDaVisualizzare** e **attende** un **input in formato Matlab**

- Un numero (i.e., uno scalare)
- Un carattere (delimitato da apici singoli)
- Array/Matrice, se racchiuso tra [ e ], oppure
- Stringa, se racchiusa tra ' e ', oppure
- Una qualsiasi espressione Matlab



## Acquisizione Dati da Tastiera (input)

Funzione input

```
valore = input(stringaDaVisualizzare) ;
```

Matlab stampa a video la **stringaDaVisualizzare** e **attende** un **input in formato Matlab**

- Un numero (i.e., uno scalare)
- Un carattere (delimitato da apici singoli)
- Array/Matrice, se racchiuso tra [ e ], oppure
- Stringa, se racchiusa tra ' e ', oppure
- Una qualsiasi espressione Matlab

Il dato inserito dall'utente viene memorizzato nella variabile **valore**

**stringaDaVisualizzare** **deve essere racchiusa tra apici singoli**





## Acquisizione Dati da Tastiera (input)

Funzione input

```
valore = input(stringaDaVisualizzare) ;
```

Matlab stampa a video la **stringaDaVisualizzare** e **attende** un **input in formato Matlab**

- Un numero (i.e., uno scalare)
- Un carattere (delimitato da apici singoli)
- Array/Matrice, se racchiuso tra [ e ], oppure
- Stringa, se racchiusa tra ' e ', oppure
- Una qualsiasi espressione Matlab

Quando si vuole leggere una stringa da tastiera, la sintassi da usare è:

```
str = input(stringaDaVisualizzare, 's') ;
```



## Stampa dei Risultati

I risultati di un'operazione sono mostrati immediatamente se non si inserisce il ;

Altre due funzioni: `disp` e `fprintf`



`disp ( stringa ) ;`

- accetta come parametro un array.
- viene usato in congiunzione con la funzione `num2str` quando è necessario stampare sia testo che numeri

Esempio:

```
str = ['il valore di pi e` ', num2str(pi)];
```

```
disp(str);
```

Stampa: "il valore di pi e` 3.1416"



## Scrittura con `fprintf`

`fprintf ( stringa ) ;`

- *stringa* sequenze di caratteri (i.e., stringa) delimitata da apici singoli ‘ ’.
- Possono essere
  - **caratteri normali** (lettere, cifre, punteggiatura)
  - caratteri speciali (es, vai a capo)
  - Placeholders (e.g. ‘%d’ per il contenuto di variabili)

I caratteri nella *stringa* vengono riportati a schermo.



## *stringaControllo:*

### Alcuni **caratteri speciali per la stampa**

- `'\n'` manda a capo
- `'\t'` spazia di un «tab»

### Alcuni **caratteri di conversione**

- `%d` intero decimale
- `%f` numero reale
- `%c` carattere
- `%s` sequenza di caratteri (stringa)



## disp vs. fprintf

`disp` è in grado di stampare anche valori complessi

```
>> x=1-2*i;  
>> str=[ 'disp: x = ' num2str(x) ] ;  
>> disp(str) ;  
disp: x = 1-2i
```

`fprintf` ne stampa solo la parte reale

```
>> fprintf('fprintf: x = %8.4f\n', x);  
fprintf : x = 1.0000
```



## disp vs. fprintf (2)

`disp` stampa correttamente matrici e vettori

```
>> a = [1 1 1; 1 1 1]
```

```
>> disp(a)
```

```
      1      1      1
      1      1      1
```

`fprintf` stampa solo su una riga (ok vettori, problemi con matrici)

```
>> fprintf('%d', a)
```

```
111111>>
```



## disp vs. fprintf (3)

`disp` permette di stampare anche vettori concatenati con stringhe se le dimensioni sono compatibili

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

```
>> disp(['hai inserito ' num2str(x)])
```

```
hai inserito 1  2  3
```

`fprintf` opera diversamente

```
>> fprintf('hai inserito %d\n',x)
```

```
hai inserito 1
```

```
hai inserito 2
```

```
hai inserito 3
```