Introduzione a Matlab

Informatica B a.a. 2021/2022

Francesco Trovò

22 Ottobre 2021

francesco1.trovo@polimi.it

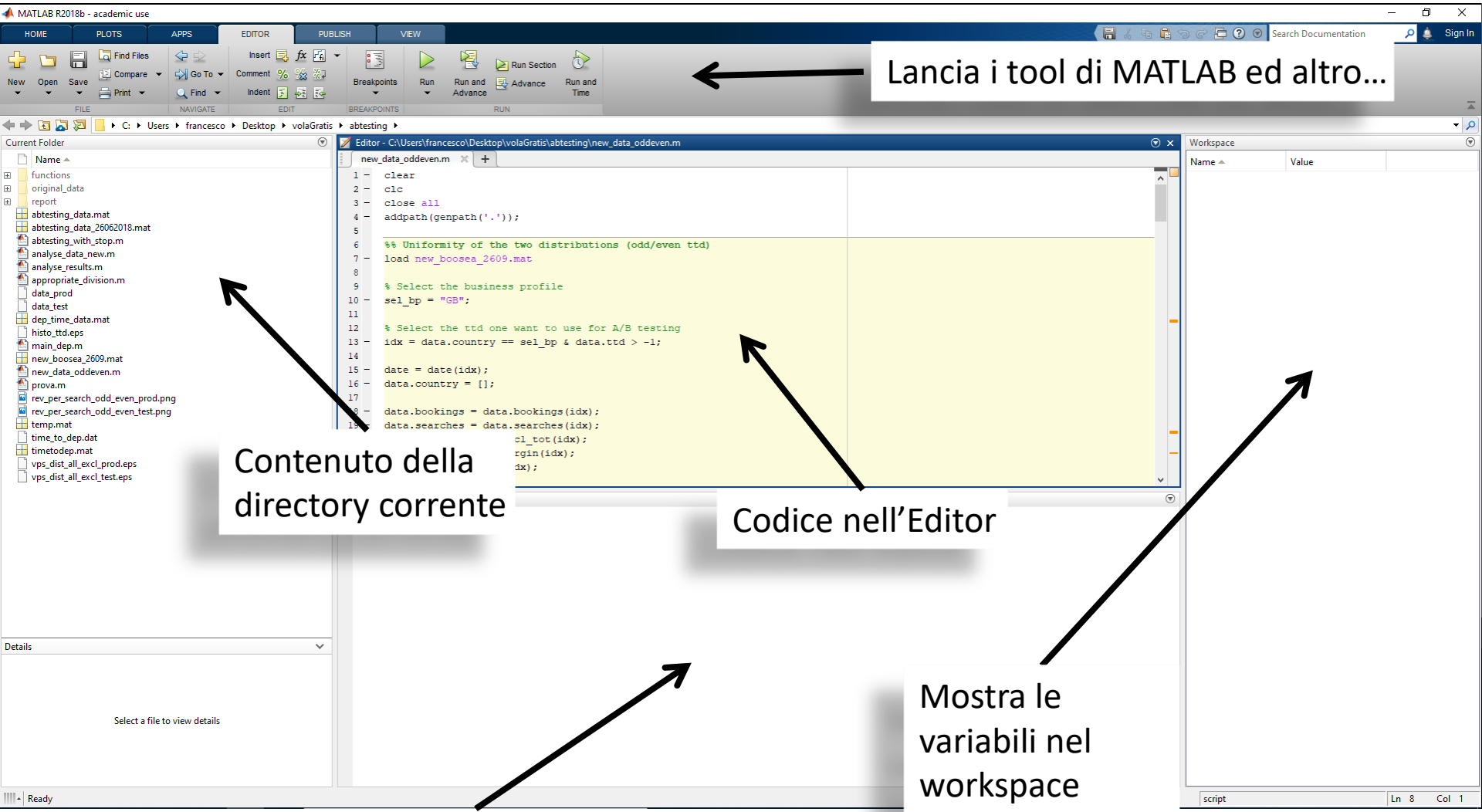
Matlab

MATrix LABoratory

Matlab

- 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:
  - Licenza Studenti fornita dal Politecnico (maggiori dettagli a laboratorio)
  - Dal 2015 Matlab è utilizzabile anche fuori della rete polimi
  - Esiste anche la possibilità di usare la versione online senza doverlo scaricare

Screenshot Interfaccia MATLAB

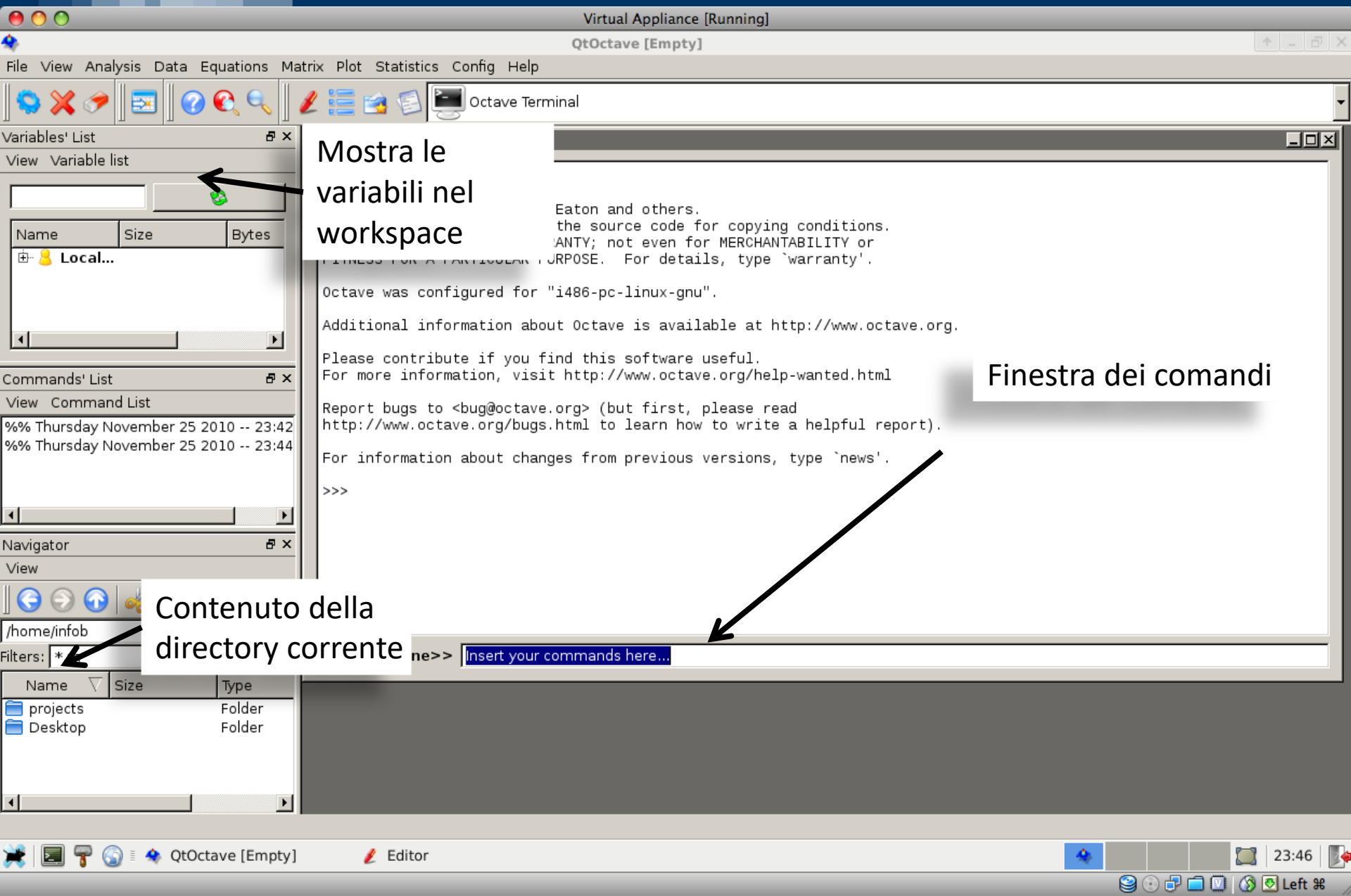


Octave

- Esiste **Octave**, una soluzione alternativa:
  - identico nella concezione, molto simile per gli aspetti operativi
  - disponibile gratuitamente, vedi www.gnu.org/software/octave/
  - occorre installare un'interfaccia grafica qtoctave
  - vedrete tutto a laboratorio

Screenshot dell'Interfaccia OCTAVE

Screenshot dell'Interfaccia Grafica QtOctave



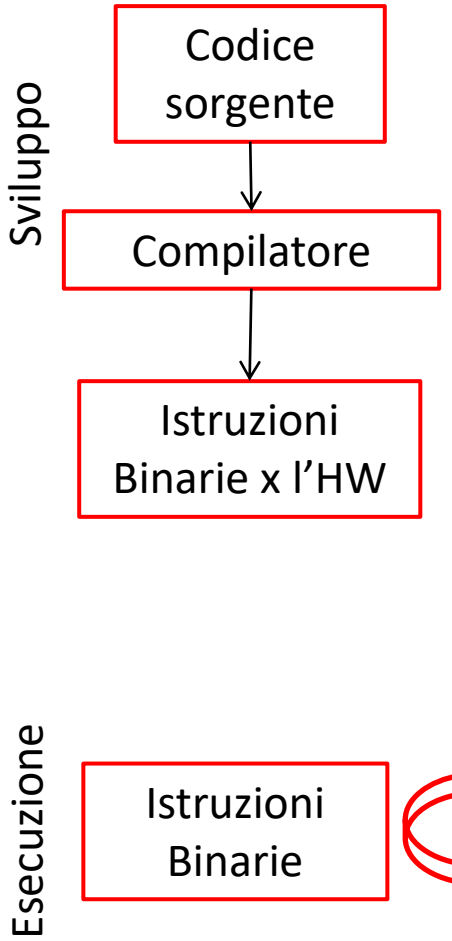
Caratteristiche del Linguaggio Matlab (1)

- 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
  - Matlab invia istruzioni ad un altro programma, **l'interprete**, che li analizza ed esegue azioni da essi descritte

Linguaggi Compilati vs Interpretati

Linguaggi **Compilati**:

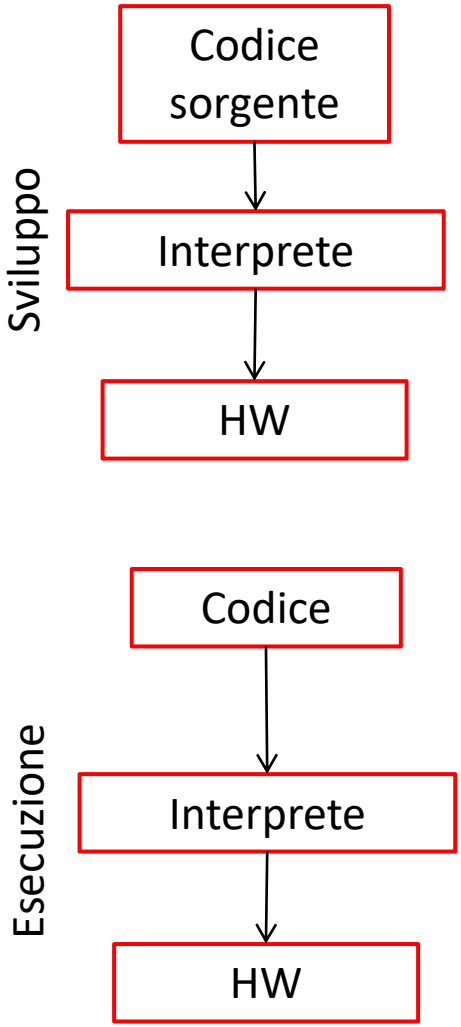
- Il compilatore è un programma che **traduce** le istruzioni del codice sorgente in codice macchina (in binario)
- L'esecuzione del programma non richiede la presenza del codice sorgente, né del compilatore
- I programmi sono efficienti
- Il programma è facilmente portabile su piattaforme analoghe



Linguaggi Compilati vs Interpretati

Linguaggi **Interpretati**:

- L'interprete è un programma che **esegue** istruzioni contenute nel codice sorgente
- L'esecuzione del programma richiede la presenza del codice (talvolta il sorgente) e 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 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:

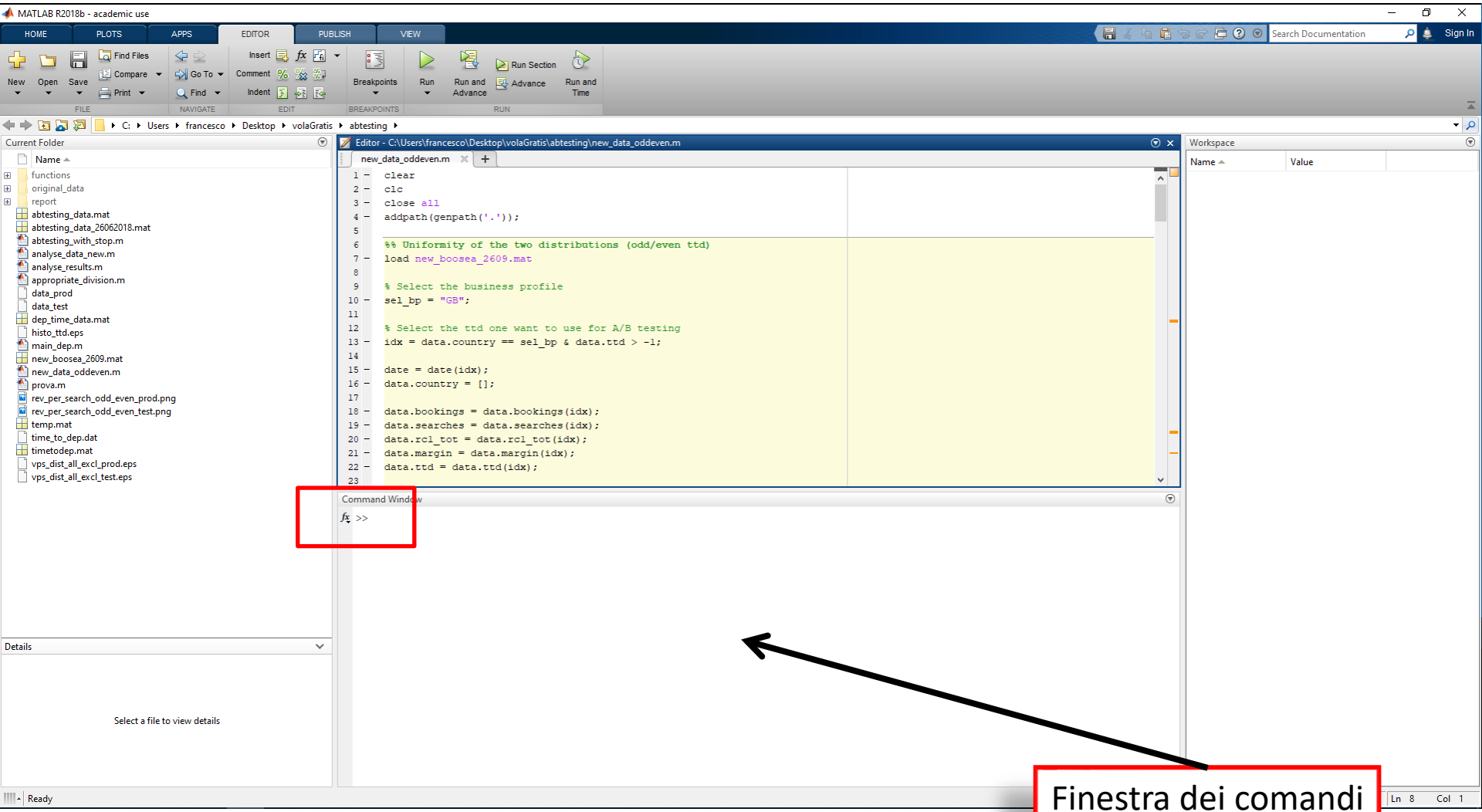
- il linguaggio Matlab che utilizziamo per codificare i programmi
- l'interprete Matlab che viene invocato per eseguire i nostri programmi
- l'ambiente di sviluppo integrato (IDE) che permette di scrivere ed eseguire i programmi e di interfacciarsi con la command window

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



Esempio: le Operazioni Aritmetiche

Nella command window è possibile eseguire operazioni aritmetiche

```
>> 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 operazioni aritmetiche

```
>> 5 + 7

ans =

    12
```

```
>> 5 * 7

ans =

    35
```

```
>> 5 ^ 7

ans =

    78125
```

```
>> 5 / 7

ans =

    0.7143
```

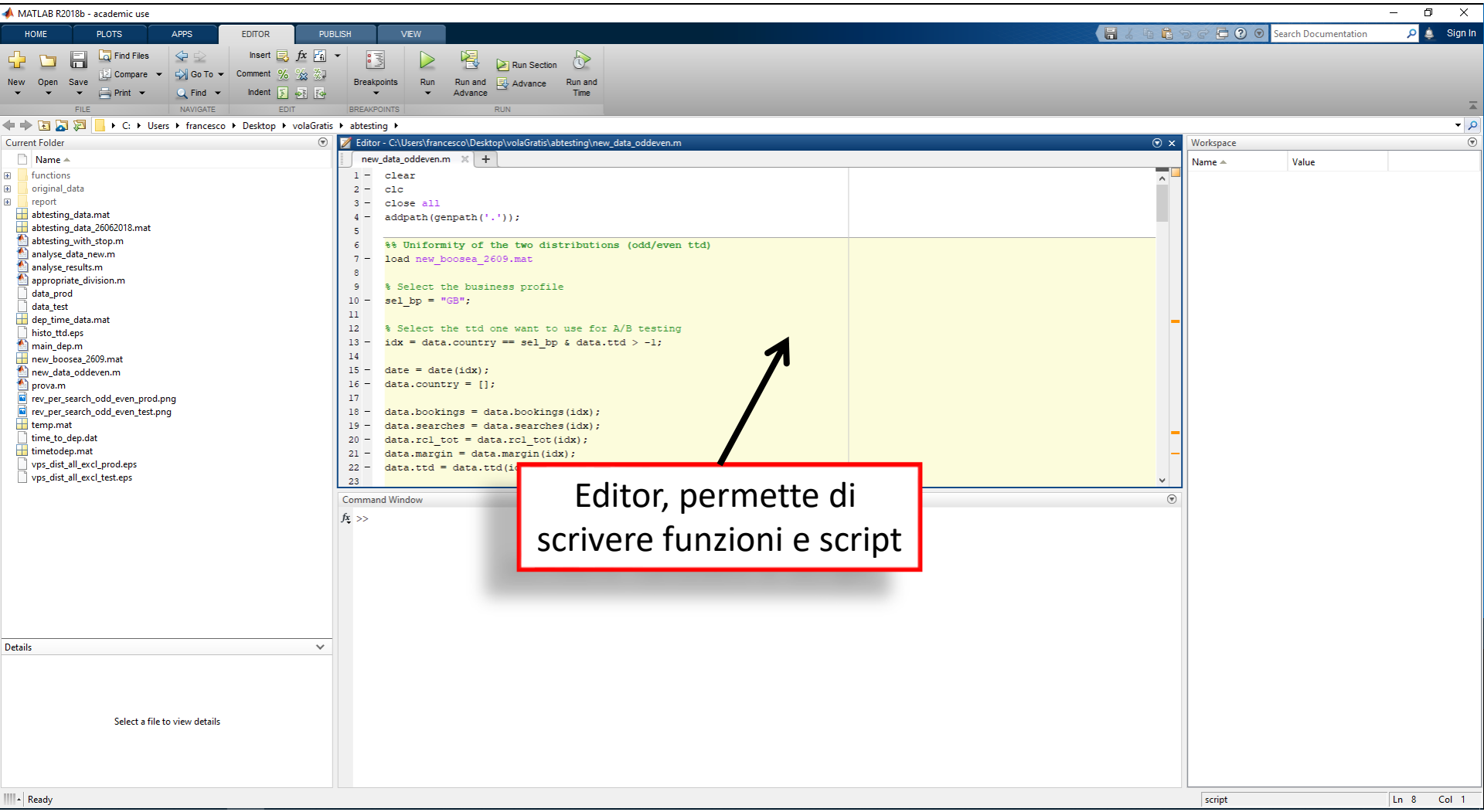
```
>> 'a' + 2

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
- Sono eseguite in maniera sequenziale
- L'esecuzione di uno codice sorgente può essere visto come l'inserimento delle varie istruzioni nella command window nell'ordine in cui sono scritte

Screenshot Interfaccia MATLAB



Il terminatore ';'

- 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

N.B. 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)

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
```

A differenza del C:

- non si deve (non è possibile) dichiarare la variabile **`nomeVariabile`** precedentemente
- l'assegnamento sopra comporta una dichiarazione contestuale 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
```

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

```
>> a = v
Undefined function or variable 'v'.
```

- Ottengo un messaggio di errore dell'interprete Matlab

Caratteristiche del linguaggio di Matlab (3)

- In Matlab tutto è un array:
  - **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 vengono salvate nel workspace, che corrisponde allo spazio di memoria del programma
- È 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
>> whos
>>
```

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 (codice ASCII di '~': 126; alt126)

La negazione in Matlab si ottiene con ~

Definizione di Vettore

- 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 uno colonna e viceversa)

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

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

Definizione Array Mediante Incremento Regolare

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

```
[inizio : step : fine]
```

definisce un vettore che ha:

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

Note sulla Creazione dei Vettori

- Il valore di **`step`** può essere qualsiasi, anche negativo
- Se non precisato, **`step`** vale 1
- Le parentesi [ ] possono essere omesse

NB: i vettori definiti per incremento regolare possono essere vuoti
Es:

```
>> [10 : -1 : 10]
```

Modifica di un Array

- È 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`**

NB: Non è necessario che **entrambi gli array abbiano le stesse dimensioni**

Es.

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

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
```

- 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 un 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

>> 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
```

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

```
>> a(ii) = a(ii - 1) + a(ii + 1)
a =
     1     4     3
```

Operazioni Aritmetiche tra Vettori

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

- La somma tra vettori `c = a + b` è definita elemento per elemento

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

è possibile solo quando `a` e `b` hanno **la stessa dimensione** (che poi coincide con quella di `c`)

- Prodotto tra vettori è il prodotto riga per colonna, restituisce uno scalare ossia `c = a * b` è definito come

$$c = \sum_i a(i)b(i)$$

`a` deve essere un vettore riga e `b` colonna e devono avere lo stesso numero di elementi, `c` è un numero reale

Eccezione per la Somma

- Se sommo un vettore riga (1xn) ed uno colonna (mx1), anche con dimensioni diverse, ottengo una matrice:

```
>> a = [1 : 3]
>> b = [1 : 4]'
>> a + b
ans =
     2     3     4
     3     4     5
     4     5     6
     5     6     7
```

- Replica per un numero di righe opportuno il vettore riga e per un numero di colonne opportune il vettore colonna

Operazioni Puntuali

- È 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 $c(i) = a(i)^{b(i)} \ \ \forall i$

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

Es.

Se `k` è uno scalare e `b` è un vettore:

`c = k * b = k .* 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 rifermento al prodotto vettoriale (equivale a **`v1 * v1`** che vale solo per matrici quadrate)

Per elevare a potenza ogni singolo elemento di **`v1`** si usa:

```
>> v1.^2
ans =
 4     9    25    16
```

che equivale a fare **`v1 .* v1`**

Recap 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	risultato(i,j) = a(i,j) / b(i,j)
Array left division	a .\ b	risultato(i,j) = b(i,j) / a(i,j)
Matrix right division	a / b	a*inversa(b)
Matrix left division	a \ b	inversa(a)*b
Array exponentiation	a .^ b	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:

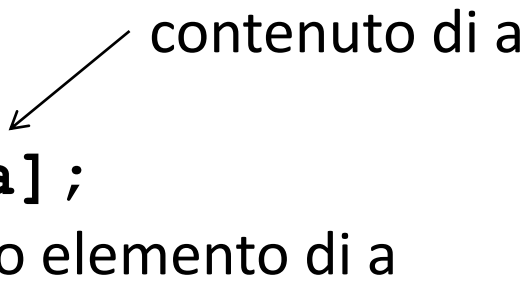
```
>> a = [1,2,3]
a =
     1     2     3
>> b = [a, a + 3 , a + 6]
b =
 1   2   3   4   5   6   7   8   9
>> b = [a, a +3]
b =
 1   2   3   1   2   3   3
```

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

Concatenare i Vettori

Esempi

- `a = [0 7+1];`
- `b = [a(2) 5 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 `'_'`

Es.

```
>> 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

- 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
```

```
>> 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: viene 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

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

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

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

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

Matrix Left Division

- Serve per risolvere sistemi di equazioni lineari

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m
\end{aligned}
$$

$$
\begin{pmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \cdots & a_{mn}
\end{pmatrix}
\begin{pmatrix}
x_1 \\ x_2 \\ \vdots \\ x_n
\end{pmatrix}
=
\begin{pmatrix}
b_1 \\ b_2 \\ \vdots \\ b_m
\end{pmatrix}
\qquad \mathrm{A}\,x = b
$$

La soluzione è $x = A^{-1}b$ oppure $x = A\backslash\mathrm{b}$, che corrisponde a calcolare la matrice A^{-1} (l'inversa di A rispetto all'operazione di prodotto matriciale)

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 il valore nella posizione indicata

```
>> c = 1
c =
     1
>> c(3) = 3
c =
     1     0     3
>> c(2,3) = 5
c =
     1     0     3
     0     0     5
```

NB: Assegnare un valore ad un elemento è diverso da accedere

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 x 3648 x 3 elementi
  - 10 sec di video full HD (1080 x 768) a 24fps richiede una matrice 4D di 1080 x 768 x 3 x (10 x 24) elementi

Esempi di Operazioni su Matrici

`a = [1 2;`
`  3, 4]`
	`a =`
	`  1  2`
	`  3  4`
		a ora è una matrice 2x2, ";" separa le righe; virgola (opzionale) separa elementi

`a`	`a =`
	`  1  2`
	`  3  4`
		restituisce il valore della variabile a

`x = [-1.3`
`sqrt(3)`
`(1+2)/5]`
	`x =`
	`-1.30000`
	`  1.73205`
	`  0.60000`
		Elementi del vettore possono essere espressioni aritmetiche

`x(5) =`
`abs(x(1))`
	`x =`
	`-1.30000`
	`  1.73205`
	`  0.60000`
	`  0.00000`
	`  1.30000`
		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 = a'`	`b =`
	`1 3`
	`2 4`
		matrice trasposta (scambiate righe e colonne)

`c = a+b`	`c =`
	`2 5`
	`5 8`
		somma di matrici, elemento per elemento (sottrazione con "-" simile)

`x=[-1 0 2];`
`y = x'`
	`y =`
	`-1`
	`0`
	`2`
		il ";" blocca l'output, ma non impedisce la valutazione

Le Variabili ed i Tipi

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
>> whos
>>
```

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`

```
Name      Size Bytes Class     Attributes
frase     1x21 42    char
```

Tipo Complex

- In Matlab è possibile rappresentare anche numeri **complessi**
- Parti reali e immaginarie possono essere positive e negative nell'intervallo di valori [10^{-308}, 10^{308}]

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

>> whos
 Name    Size     Bytes  Class    Attributes
  a       1x2     32  double      complex
```

Tipo Complex

- I double possono essere utilizzati per esprimere numeri
  - Reali, es var1 = -10.7;
  - Immaginari, es var2 = 4i; var3 = 4j;
  - Complessi, es var3 = 10.3 + 10i;

Es.

```
x = [-1.3   3.1+5.3j   0]
```

NB: Meglio non usare mai i e j come nome di variabile in quanto hanno di default un significato differente

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
```

Script

Script (m-file)

- Uno script è un **file di testo** contenente una **sequenza di comandi MATLAB**:
  - non deve contenere caratteri di formattazione (solo testo puro)
  - viene salvato con estensione .m
- I comandi all'interno di uno script sono eseguiti **sequenzialmente**, come se fossero scritti nella finestra dei comandi:
  - per eseguire il file (quindi la sequenza di istruzioni che contiene) si digita il suo nome (senza .m)
  - i risultati appaiono nella finestra dei comandi (se non usiamo il ;)

Provare ad eseguire uno script in modalità Debug!

Vantaggi/Svantaggi

- Uno script può:
  - essere ri-eseguito
  - essere facilmente modificato
  - essere facilmente inviato
- Uno script opera sulle variabili del workspace:
  - posso introdurne di nuove durante l'esecuzione
  - posso accedere alle variabili nel workspace
  - posso modificare le variabili del workspace

NB: spesso è bene ripulire il workspace (con il comando **`clear`**) come prima cosa, in modo da evitare di eseguire script su variabili che compaiono già nel workspace

Commenti

- Il simbolo di commento può essere messo in qualsiasi punto della linea
- MATLAB ignorerà tutto quello che viene scritto alla destra del simbolo %

Es.

```
>> % This is a comment
>> x = 2+3 % So is this
x =
     5
```

Come Creare uno Script

- Può essere creato utilizzando un qualsiasi editor di testo
  - Ricordarsi di salvare il file come “solo testo” e di dare l’estensione .m
  - Il file di **script** deve essere **presente nella directory corrente** o il **folder** contenente lo script deve **comparire nel path** di Matlab

Nomi degli Script

- Il nome del file deve **iniziare con una lettera** e può contenere cifre e il carattere underscore, fino a 31 caratteri
- Non dare lo stesso nome al file di script e a una variabile
- Non chiamare uno script con lo stesso nome di un comando o funzione MATLAB
- Per verificare se esiste già qualcosa che ha un certo nome si può utilizzare la funzione **`exist`**

Strutturare e Documentare uno Script

1. Sezione dei commenti:
   - il nome del programma e le parole chiave, nella prima riga
   - la data di creazione e i nomi degli autori nella seconda riga
   - la definizione dei nomi delle variabili per ogni variabile di input e di output
   - il nome di ogni funzione creata dall'utente che viene usata nel programma
   - il comando help visualizza tutta la sezione dei commenti all'inizio dello script
2. Sezione di Input: inserimento dei dati in input e/o uso di funzioni di input
3. Sezione di calcolo
4. Sezione di output: uso di funzioni per visualizzare i risultati del programma

Dati su cui Opera uno Script

- Gli script non accettano argomenti d'entrata e d'uscita
- Usano:
  - variabili già presenti nel workspace
  - variabili acquisite da tastiera o file
  - nuove variabili introdotte nello script
- Le variabili interne allo script diventano variabili del workspace
- Permangono anche dopo l'esecuzione dello script

Sezione di Calcolo

Comprende:

- Calcoli matematici
- Assegnamenti
- Strutture di controllo
  - Condizioni
  - Cicli
- Comandi per la costruzione di grafici
- Chiamate a funzioni

Altri Comandi in Matlab

Esempio di alcuni comandi (analizzeremo quelli più importanti)

- Il prompt accetta i comandi del sistema operativo (DOS, UNIX…)
  - Esempio: in ambiente dos, **`dir`** mostra il contenuto della directory corrente
- **`help`** richiama la guida in linea
- **`who`**, **`whos`** e **`workspace`** mostrano l'elenco delle variabili definite
- **`save`** permette di salvare in un file le variabili definite
- **`load`** le ricarica
- **`clear`** cancella tutte le variabili
- **`close`** chiude tutte le figure

Altre Operazioni sugli Array

Sottoarray (a.k.a. un Vettore come Indice di un Vettore)

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`**

Estende quindi l'accesso ad un singolo elemento dell'array

`nomeVettore(indice)`

Sottovettori Definiti da Vettori di Indici

Quindi, la notazione

```
a(v)
```

corrisponde a

```
[a(v(1)), a(v(2)), ..., a(v(end))]
```

NB: i valori di **`v`** devono essere interi positivi e minori delle dimensioni di **`a`**, ovvero 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

Es.

```
>> a = [1 : 6]
a =
     1     2     3     4     5     6
```

Toglie l'ultimo elemento

```
>> b = a(1 : end - 1)
b =
     1     2     3     4     5
```

Legge il vettore dall'ultimo elemento al primo

```
>> b = a(end : -1 : 1)
b =
     6     5     4     3     2     1
```

Cosa Fanno le Seguenti Istruzioni?

```
>> v = [6 8 4 2 4 5 1 3];
>> v([1 4 7])
ans =     6     2     1

>> v(2:2:6)
ans =     8     2     5

>> v(3:end-2)
ans =     4     2     4     5

>> v(v)
ans =     5     3     2     8     2     4     6     4

>> v([1, 1, 1, 2, end])
ans = 6     6     6     8     3
```

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

```
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)
```

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
```

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)
```

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)
```

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
```

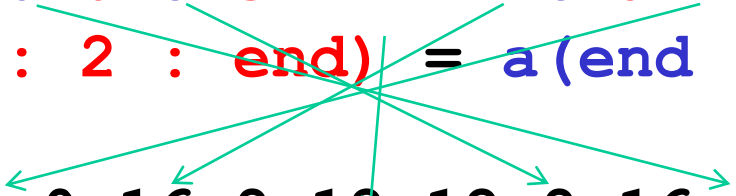
Modificare un Sotto-Array

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

```
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)
a =
    20  0 16  8 12 12  8 16  0 20
```



Sottoarray: Applicazione a Matrici

- Si estrae da una matrice un sottoinsieme di un array usando vettori per valori di ognuni degli indici
- Quindi:

```
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])
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(1:2:5, 1:end)
ans =  9     8     7
       3     2     1
       0     0     0
 >> m(1:2:5, :)
```

tutti gli elementi delle
righe 1, 3 e 5

: equivale ad `1:end`

Sottoarray: Applicazione a Matrici

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

```
>> 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;
% modificare la colonna centrale in 1
A(: , 3) = 1;
% modificare la riga centrale in 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 si 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 : 6]
a =
     1     2     3     4     5     6

>> a(1 : 3) = 0
a =
     0     0     0     4     5     6
```

Assegnamenti con Scalari su Matrici

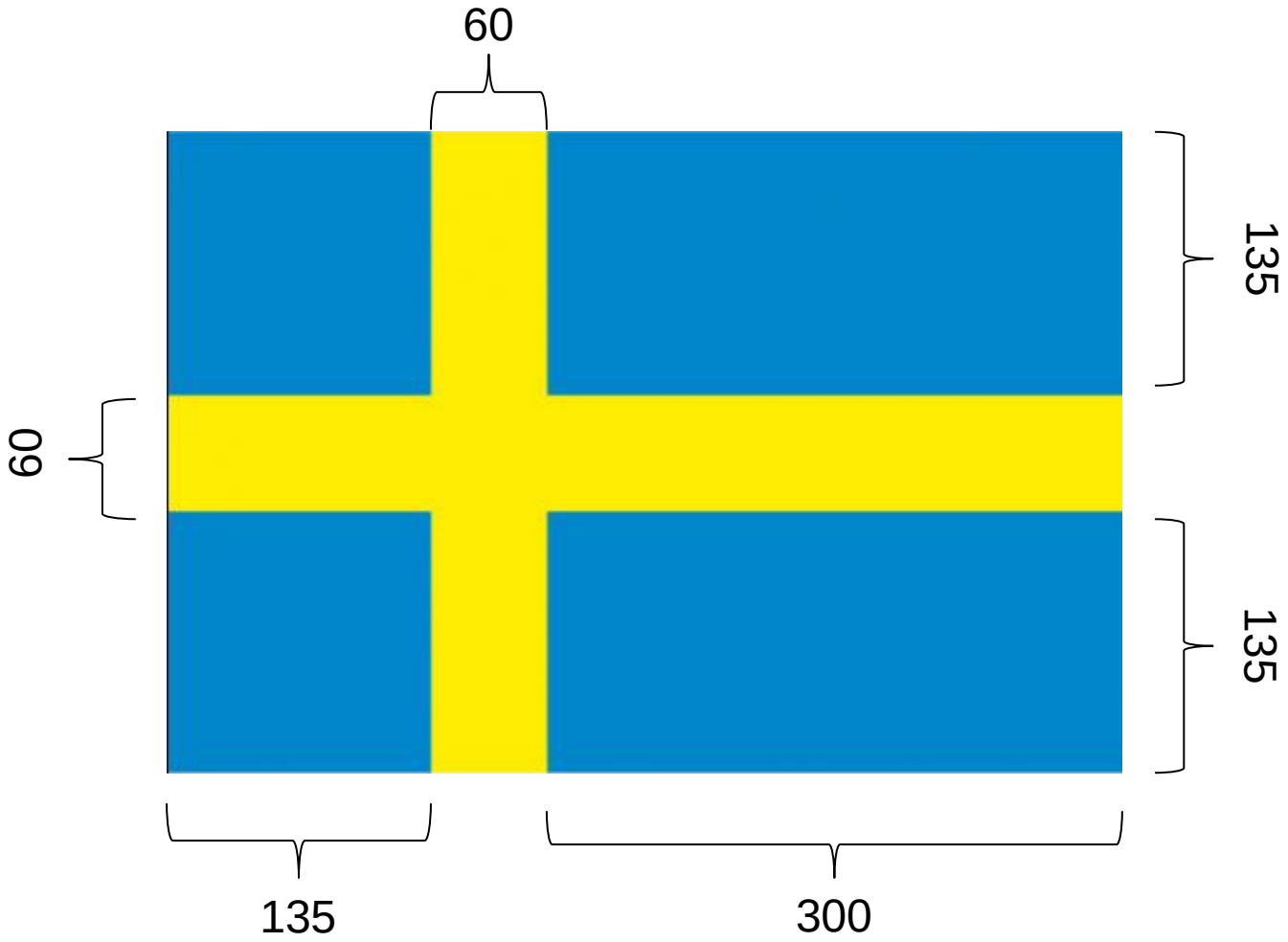
Il modo con cui uno scalare viene assegnato a un array dipende dalla forma dell'array che viene specificata a sinistra dell'assegnamento

Es.

```
>> m(4, 3) = 3;
>> m(1:2, 1:2) = 4
ans =
     4     4     0
     4     4     0
     0     0     0
     0     0     3
```

Esercizio

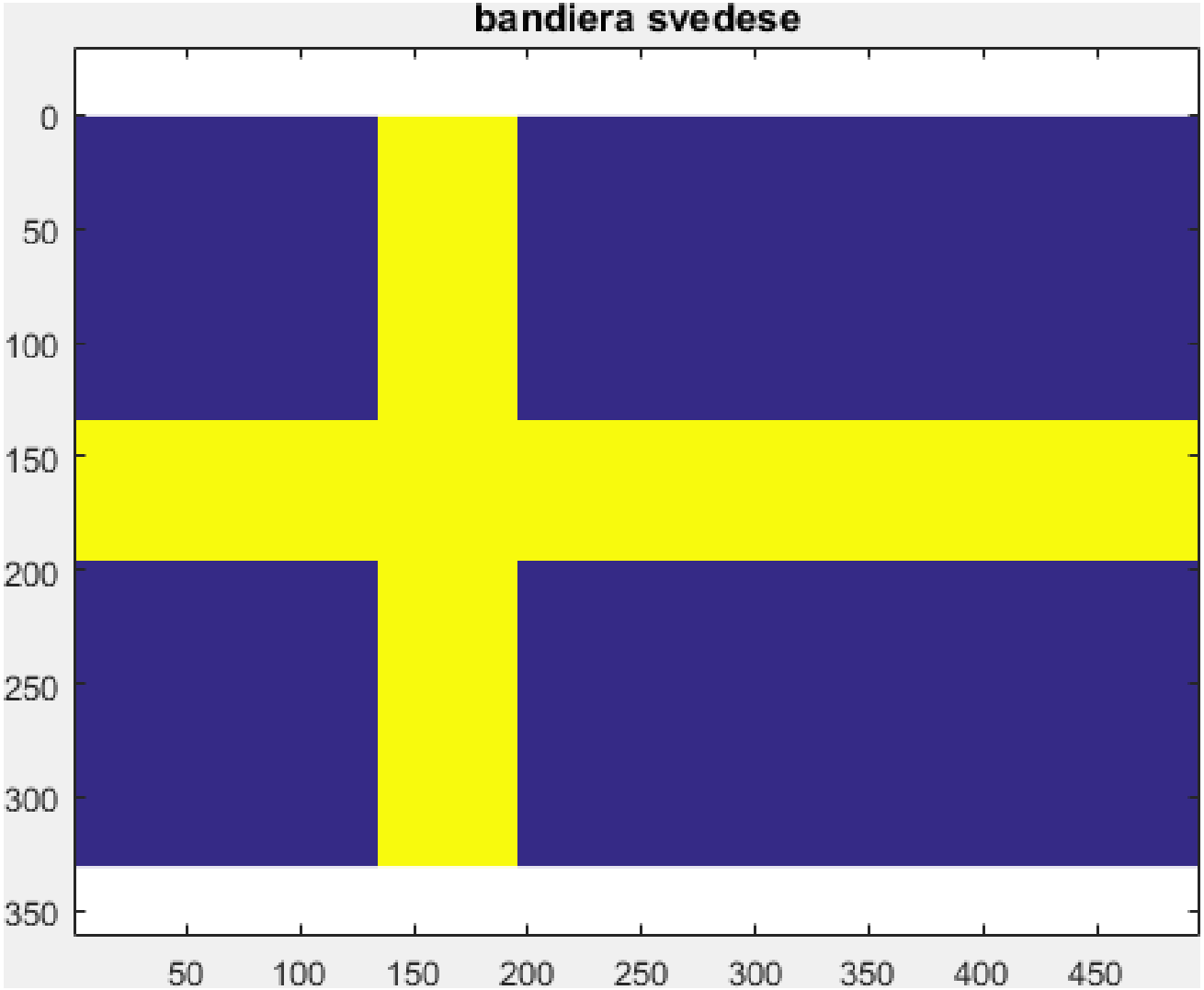
Disegnare la bandiera svedese



Bandiera Svedese

```
clear;
clc;
close all;

A(330, 495) = 0;
A(:, 135:195) = 1;
A(135:195, :) = 1;

figure();
imagesc(A);
title('Bandiera svedese');
axis equal;
```

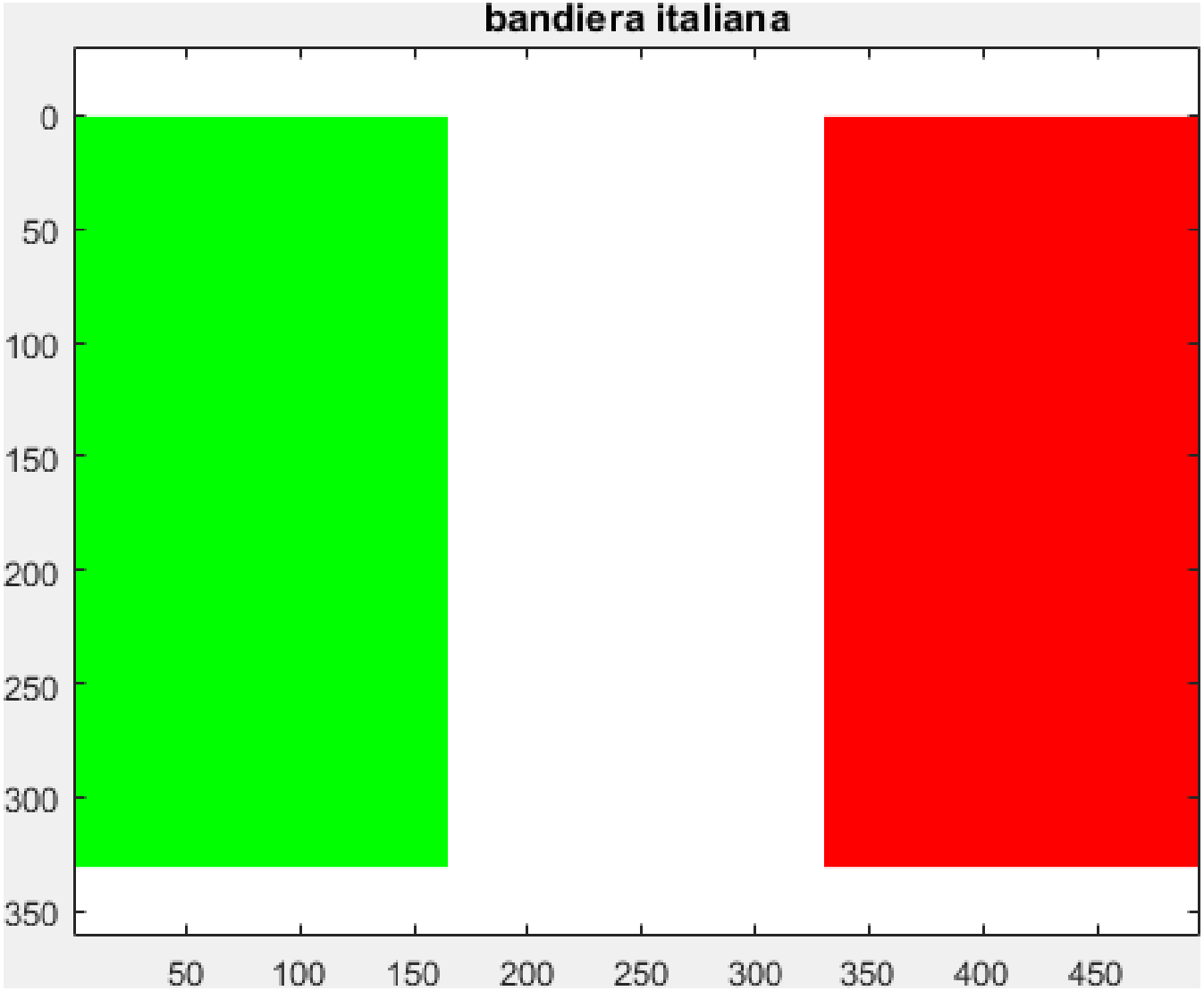
Ecco il Risultato (Immagine 2D)



Bandiera Italiana (Immagine 3D)

```
clear;
clc;
close all;

A(330, 495, 3) = 0;
A(:, 1 : 495/3, 2) = 1; % verde
A(:, 495/3 : (2*495)/3, :) = 1; % bianco
A(:, (2*495)/3 : end, 1) = 1; % rosso

figure(); imagesc(A);
title('Bandiera italiana');
axis equal;
```

Ecco il Risultato



GRAN CONTEST DELLE BANDIERINE

Regole:

- la bandiera deve essere realizzata interamente con uno script MatLab, senza interazione con l'utente
- è necessario usare operazioni vettoriali, si possono usare cicli
- verrà valutata sia la veridicità della bandiera, sia la struttura del codice utilizzato per realizzarla
- potete presentare una sola bandiera a persona
- inviate tramite mail:
  - un'immagine (png) della bandiera
  - lo script che la genera (opportunamente indentato e commentato)
- deadline: 8 Gennaio 2022 (23.59 UTC+1)

GRAN CONTEST DELLE BANDIERINE

Come creare le immagini a colori:

- è possibile riprodurre i colori in due modi:
  - creando una matrice 3D B che ha N_righe x N_colonne x 3 dove il terzo piano indica il colore (B(:,:,1) è il rosso, B(:,:2) il verde, B(:,:,3) il blu) e visualizzando con imshow
  - creando una matrice 2D e modificando la colormap

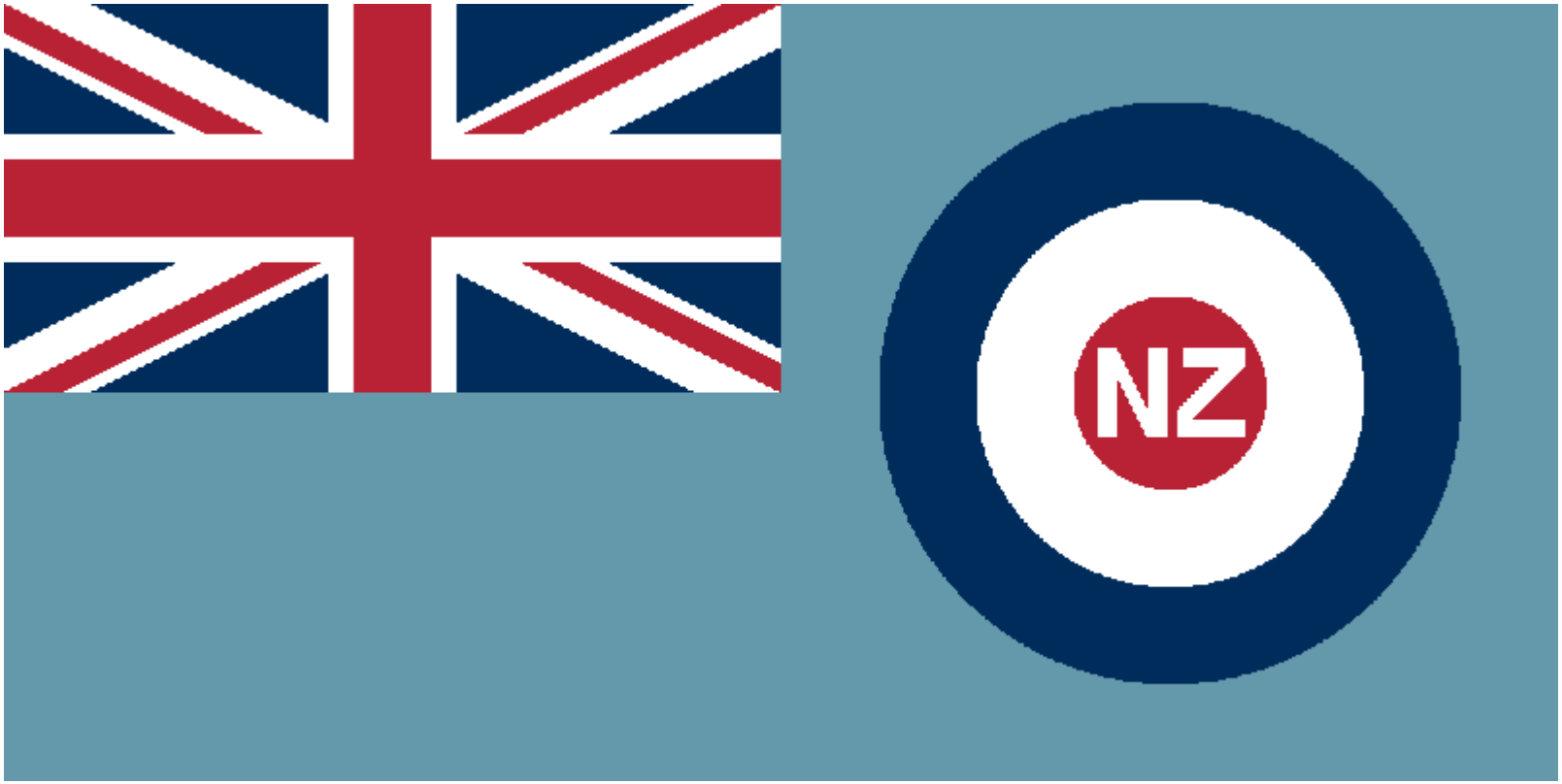
Premi:

- un punto alla bandierina più bella dello Scaglione (se lo studente è in grado di descrivere come ha raggiunto il risultato)

Esempi di Bandiere Realizzate dai Vostri Colleghi…

Bandiera della Catalogna

Esempi di Bandiere Realizzate dai Vostri Colleghi…



New Zeland Airforce

Esercizio

Visualizzare a schermo le prime 10 tabelline (utilizzando solo operazioni matriciali)

```
>> a = [1 : 10]
a =
    1  2  3  4  5  6  7  8  9 10
% replico le righe -> matrice 5x10
>> A =  [a; a; a; a; a];
% faccio una matrice 10x10
>> A = [A; A];
% tabellina del 10
>> T = A.*A';
```

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
>> 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 matrice In quel caso si creerebbero dei buchi

```
>> m(1 : 3, 1:3) = 1
m =
     1     1     1
     1     1     1
     1     1     1
>> m(2 , : ) = 5
m =
     1     1     1
     5     5     5
     1     1     1
>> m(3,3)=[]
??? Subscripted assignment dimension
mismatch.
```

Cancellare Parti di una Matrice

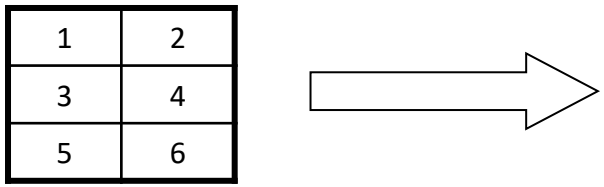
È però assegnabile a intere righe o colonne di matrici, che vengono cancellate (ricompattando la matrice)

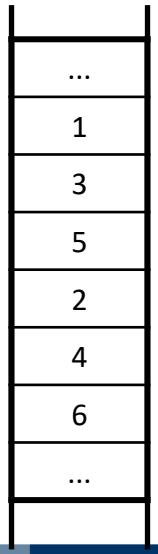
```
>> m(: , 2) = []
m =

     1     1
     5     5
     1     1
>> whos m
  Name        Size    Bytes Class    Attributes
  m           3x2        48 double
```

Memorizzazione degli Array

- Gli array vengono salvati linearmente in memoria
- In particolare le matrici sono memorizzate:
  - per colonna: colonna 1, poi colonna 2, 3, etc.
  - ogni colonna memorizzata per indici di riga crescenti
- Array memorizzati in forma lineare nella RAM variando
  - più velocemente i primi indici
  - più lentamente quelli successivi

NB: differente da quanto avviene in C





Array: forma *linearizzata*

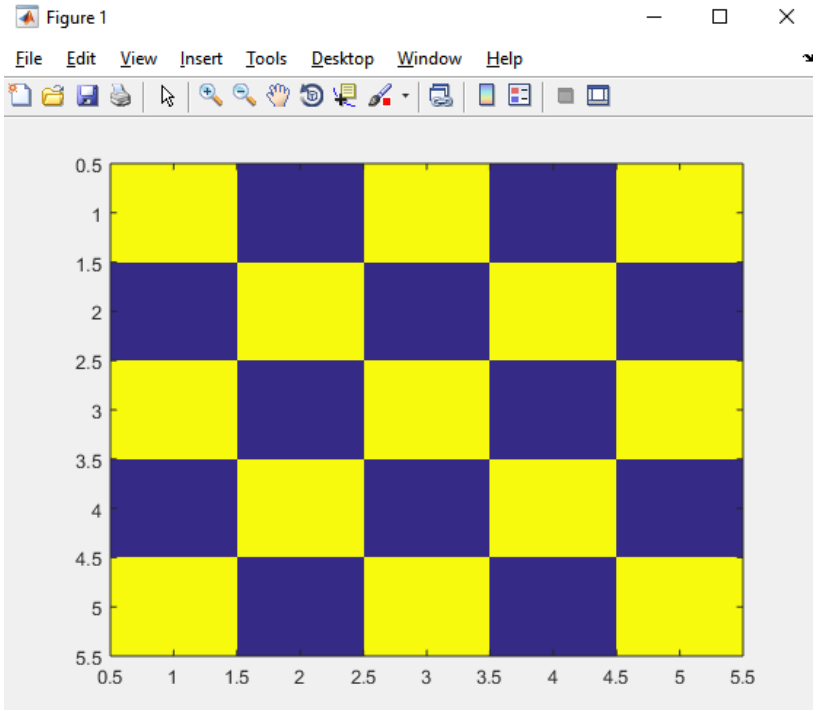
- Si può accedere a un array a più dimensioni come se ne avesse una sola
- Usando un unico indice si segue l'ordine della memorizzazione

```
>> a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
a =
     1     2     3
     4     5     6
     7     8     9
    10    11    12

>> a(3, 2)
ans =
     8
>> a(10)
ans =
     6
```

Esempio

Scrivere uno script che stampa una scacchiera 5 x 5 (alternando elementi uguali a 0 ed uguali ad 1)

```
clear;
clc;

s = 5;
A(s,s) = 0;
A(1:2:end) = 1;
figure();
imagesc(A);
```



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))
  - 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

NB: È 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 π
`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 elementi (anno, mese, giorno, ora, minuti, secondi)
`date`	contiene la data corrente sotto forma di stringa
`eps`	epsilon: la più piccola differenza rappresentabile tra due numeri
`ans`	Variabile speciale usata per immagazzinare risultati non assegnati ad altre variabili

Input/output

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

Acquisizione Dati da Tastiera (input)

Funzione input

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

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

- numero (i.e., uno scalare)
- carattere (delimitato da apici singoli)
- array, se racchiuso tra `[]`, oppure
- stringa, se racchiusa tra `' '`, oppure
- una qualsiasi espressione Matlab

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

Se `stringaDaVisualizzare` è una sequenza di caratteri, questa deve essere racchiusa tra apici singoli

Stampa dei Risultati (1)

- Abbiamo già visto che i risultati di un'operazione sono mostrati immediatamente se non si inserisce il ;
- Altre due funzioni: **`disp`**
  - accetta come parametro un array
  - occorre spesso concatenare stringhe mediante l'operatore horzcat (spazo o virgola)
  - viene usato in congiunzione con la funzione **`num2str`** che trasforma un valore numerico in stringa

Es.

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

stampa: "il valore di pi e` 3.1416"

Scrittura con `fprintf`

Per scrivere a schermo:

```
fprintf(stringaControllo);
```

- ***stringaControllo*** sequenze di caratteri (i.e., stringa) delimitata da doppi apici singoli ‘ ‘

Possono essere:

- caratteri normali (lettere, cifre, pungeggiatura)
- caratteri speciali (es, vai a capo)
- placeholders (ad esempio ‘%d’ per le variabili)

I caratteri nella ***stringaControllo*** 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 = 2*(1-2*i)^3;
>> str = ['disp: x = ' num2str(x)];
>> disp(str);
     disp: x = -22+4i
```

fprintf ne stampa solo la parte reale

```
>> fprintf('fprintf: x = %8.4f\n', x);
     fprintf: x = -22.0000
```

`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 andando colonna per colonna (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
```

Esercizio

Modificare lo script della scacchiera per richiedere all'utente le dimensioni e stampare anche un messaggio a schermo

```
clear;
clc;

s = input('Quanto grande la scacchiera? ');
A(s, s) = 0;
A(1:2:end) = 1;
figure();
imagesc(A);

disp(['scacchiera ', num2str(s), 'x',
num2str(s)])
% fprintf('scacchiera %dx%d\n', s, s)
```