

Laboratorio di Matlab

Alessandro Benfenati
Vanna Lisa Coli
Simone Rebegoldi
Serena Crisci
Gaetano Zanghirati

Dipartimento di Matematica e Informatica
Università di Ferrara

Corso di Laurea in Informatica
A.A. 2022–2023

in collaborazione con **Valeria Ruggiero**

Sommario

- **Introduzione a Matlab**
 - navigazione
 - workspace, command history
 - shell
 - editor
- **Le variabili**
 - variabili numeriche
 - vettori
 - matrici
 - array multidimensionali
- **M-script files**
- **Istruzioni di controllo e cicli**
 - istruzioni di controllo
 - cicli
- **M-functions files**
- **Tipi di dato**
 - Strutture
 - Cell arrays
- **Gestione I/O nelle M-functions**
- **Grafica 2D**
- **Miscellaneous**
 - Varie ed eventuali

Matlab (**Mat**rix **lab**oratory) è un ambiente integrato per il calcolo scientifico, basato sul **calcolo matriciale**; consente di eseguire un'istruzione o comando per volta, tali comandi permettono di definire variabili, valutare formule, disegnare grafici 2D e 3D a schermo...

È inoltre un linguaggio di programmazione **intrepretato** (non compilato), dalla sintassi facile ed intuitiva (anche se a prima vista può non sembrare). La notazione in cui i dati vengono inseriti e le soluzioni vengono espresse è una notazione di tipo matematico.

Matlab è un ambiente per studiare soluzioni numeriche di problemi di tipo fisico, economico, statistico, ingegneristico, in molti campi delle scienze applicate. Per aiutare gli utenti nella soluzione di questi problemi, sono stati creati vari **Toolbox**, ovverosia librerie con funzioni già implementate per affrontare le tematiche particolari (ad esempio, lo Statistic Toolbox è stato creato per effettuare analisi statistiche di dati.)

Per maggiori informazioni, visitare <http://www.mathworks.com>.

Alternative a Matlab

Nel caso si disponesse di grosse disponibilità finanziarie, è possibile acquistare Matlab, con un costo che va dai 2000€ (versione base) ai circa 84000€.

Alternative

Octave è un software open source, disponibile per ambienti Windows, Linux, Mac e Android; costituisce una valida alternativa a Matlab, compatibile per oltre il 90% con i codici Matlab. L'attuale versione stabile è la 7.2.0 (28 Luglio 2022) Si può scaricare dal sito

<http://www.gnu.org/software/octave/>

In ambiente Linux basato su Debian (come Ubuntu), Octave è installabile con

```
sudo apt-get install octave
```

Nelle versioni recenti, Octave è già dotato di interfaccia grafica utente (GUI) come Matlab. Può comunque essere utilizzato in modalità testuale, da shell di comando. Dal 2017, un'ampia gamma di toolbox per molte applicazioni è disponibile gratuitamente dal progetto **Octave Forge** (octave.sourceforge.io, stk 2.7.0 del 23 Feb. 2022)

Altra alternativa a Matlab è **SciLab**:

<http://www.scilab.org/>

che non ha lo stesso livello di compatibilità con Matlab posseduto da Octave.

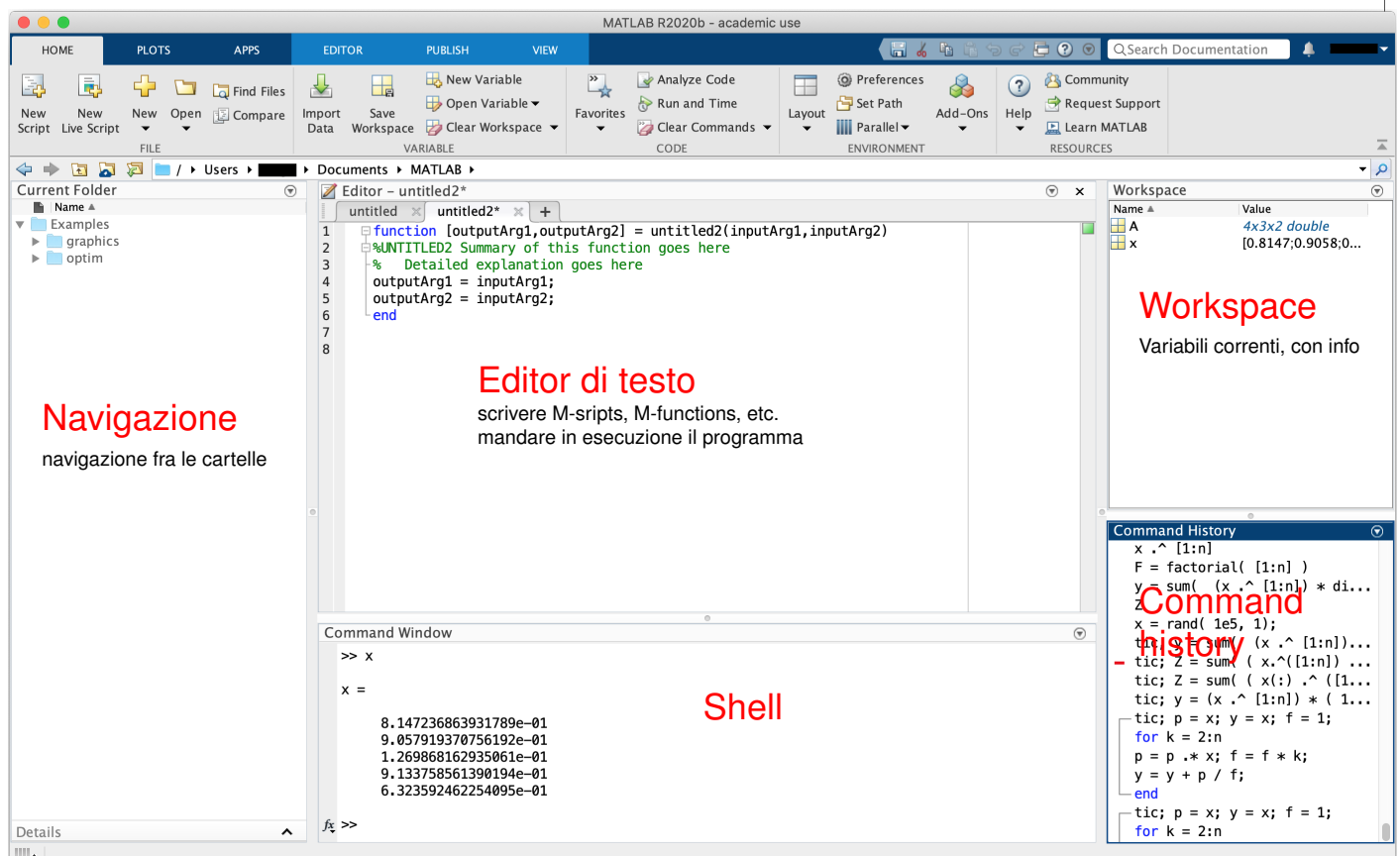
Utilizzare Matlab da linea di comando

Per lanciare Matlab digitare in un terminale semplicemente il comando **matlab**.

Un'altra possibilità è lanciarlo con il seguente comando: **matlab -nosplash -nodesktop -nojvm**.

- non fa apparire la finestra con la versione del programma;
- non apre la finestra grafica, con editor, workspace etc: si utilizza Matlab da terminale, ma è possibile aprire ad esempio l'editor digitando **edit nomefile.m**;
- non lancia la Java Virtual Machine, quindi non è possibile utilizzare finestre quali l'editor, l'help: si utilizza esclusivamente da linea di comando nel terminale.

Ambiente Matlab



Ogni finestra può essere estratta (unlocked), integrata (locked), massimizzata, minimizzata, chiusa. ...

Variabili numeriche

Quando viene eseguito un comando (per esempio `>> 3+2`), il risultato viene memorizzato nella variabile `ans`; se vengono eseguiti più comandi in successione, la variabile `ans` memorizza solo il risultato dell'ultima operazione.

Per memorizzare questi risultati successivi è necessario *definire una variabile*:

```
>> a = 3+2  
a =  
    5
```

Il risultato viene memorizzato nella variabile `a` e viene anche visualizzato.

```
>> a = 3+2;
```

Il risultato viene memorizzato nella variabile `a`, ma non viene visualizzato.

```
>> a = 3+2;  
>> a = 10+3;
```

In questo modo il valore assegnato ad `a` viene sovrascritto: in questo caso il valore memorizzato in `a` è 13.

La procedura per memorizzare un valore in una variabile è la seguente:

```
var_name = var_value ↵  
Il punto e virgola è facoltativo.
```

Variabili numeriche

- MatLab è **case sensitive**: la variabile `M` è diversa dalla variabile `m`.
- Il nome di una variabile **non** può iniziare con una cifra.
Alcuni esempi di nomi validi di variabili sono: `A_min`, `Err3`, `fun_obj4`.
- È opportuno evitare l'utilizzo di caratteri accentati.

variabile NULL

Utilizzando la seguente istruzione:

```
>> x = [];
```

si crea una variabile vuota (NULL) che può contenere dati di qualsiasi tipo.

Repetita iuvant!

`ans` è la variabile temporanea in cui MatLab memorizza il risultato dell'ultima istruzione digitata. **Viene sovrascritta ogni volta che viene utilizzata!**

```
>> sin(23)
ans =
    -0.8462
>> 5+6
ans =
    11
>> 0
ans =
     0
>> log(abs(asin(cos(pi))))
ans =
    0.4516
>> 1+1
ans =
     2
>> ans
ans =
     2
```

Se si scrivono varie operazioni in successione, il risultato che rimane memorizzato in `ans` è **solo** quello dell'ultima operazione eseguita.

Visualizzazione

```
>> disp(x)
```

Il comando `disp` serve a visualizzare sullo schermo il valore della variabile fra parentesi.

Il comando `fprintf` è utilizzato per visualizzare messaggi sulla shell.

```
>> fprintf('Messaggio che viene stampato sulla shell.\n');
```

Si può utilizzare il comando `fprintf` anche per far apparire i valori delle variabili: è necessario specificare all'interno di una stringa di testo il tipo di dato da visualizzare utilizzando il carattere “%” con una lettera che identifichi il tipo; al termine della stringa e dopo una virgola si indica il nome di ciascuna variabile da visualizzare.

```
>> fprintf('Il valore di a: %f', a)
```

Descrittori di formato identici a quelli di C

| Tipo del valore | Descrittore | Dettagli |
|--------------------------|--|--|
| intero, con segno | %d oppure %i | Base 10 |
| intero, senza segno | %u %o %x %X | Base 10 Base 8 (ottale) Base 16 (esadecimale) Come %x, ma con lettere "A" – "F" maiuscole |
| numero in virgola mobile | %f %e %E %g %G | Notazione a virgola fissa (si usa un operatore di precisione per specificare il numero di cifre desiderate dopo il punto radice) Notazione esponenziale, come in 3.141593e+00 (si usa un operatore di precisione per specificare il numero di cifre desiderate dopo il punto radice) Come %e, ma con lettera "E" maiuscola, come in 3.141593E+00 Il più compatto fra %e e %f, senza zeri iniziali (si usa un operatore di precisione per specificare il numero di cifre desiderate dopo il punto radice) Il più compatto fra %E e %f, senza zeri iniziali (si usa un operatore di precisione per specificare il numero di cifre desiderate dopo il punto radice) |
| carattere o stringa | %c %s | Singolo carattere Vettore di caratteri o array stringa. Il tipo del testo in output è lo stesso del tipo di <code>formatSpec</code> |

(dalla documentazione di `fprintf`)

Input da tastiera

Si possono inserire dati da tastiera:

```
>> x = input('Introduci il valore x:\n');
Introduci il valore x:
34
>> x
x =
    34
>> whos
  Name      Size      Bytes  Class      Attributes
  x         1x1         8    double
```

La variabile immessa è di tipo `double`.

```
>> x = input('Introduci la stringa x:\n', 's');
Introduci la stringa x:
prova
>> x
x =
prova
>> whos
  Name      Size      Bytes  Class      Attributes
  x         1x5        10    char
```

La variabile immessa è di tipo `char` (stringa).

Vettori: creazione


Matlab è ottimizzato per il **calcolo matriciale** e tutti i dati vengono visti come matrici.

I vettori sono matrici particolari

```
>> v = [12 pi 0];  
>> v = [12, pi, 0];
```

Dichiarazioni di vettori riga: utilizzare o no la virgola è ininfluente.

```
>> w = [9; 42; pi/2];  
>> w = [9  
42  
pi/2];
```

Dichiarazioni di vettori colonna: andare a capo con  o utilizzare il punto e virgola producono il medesimo risultato.

```
>> w = [9; 42; pi/2];  
>> w(2)  
ans =  
42
```

Per accedere agli elementi di un vettore si digita il nome del vettore e fra parentesi **tonde** l'indice dell'elemento desiderato.

Vettori: creazione

Alcuni comandi particolari:

```
>> b = 3:15;
```

Questo comando crea il vettore **b** i cui elementi sono equispaziati di 1.
 $b = (3, 4, 5, \dots, 15)$.

```
>> c = 0:0.1:2;
```

Questo comando crea il vettore **c** i cui elementi sono equispaziati di passo 0.1.
 $c = (0, 0.1, 0.2, \dots, 1.8, 1.9, 2)$.

```
>> d = linspace(5, 7, 101);
```

Questo comando crea il vettore **d** di 101 elementi, il primo elemento è 5 e l'ultimo è 7.

$d = (5, 5.02, 5.04, \dots, 6.96, 6.98, 7.00)$.

Il passo è calcolato automaticamente:

$d = \text{linspace}(q, r, n_p) \Rightarrow \text{passo} = \frac{r - q}{n_p - 1}$.

L'apostrofo **"'**" calcola il *trasposto di un array*.

```
>> var = [17; 42];  
>> var'  
ans =  
17    42
```

Attenzione: **var** e **var'** sono due variabili differenti!

Vettori: operazioni con i vettori

Somma, differenza e prodotto di vettori ricalcano le definizioni matematiche.

```
>> a = [2; 3];
>> b = [4; 6];
>> c = a+b
c =
     6
     9
```

```
>> a = [2; 3];
>> b = [4; 6];
>> dot(a,b)
ans =
    26
>> d = a'*b
d =
    26
```

La somma (differenza) è calcolata componente per componente.

Attenzione alle dimensioni dei vettori:
non è possibile sommare vettori di dimensioni differenti.

Il prodotto scalare $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_i a_i b_i$ è implementato con la funzione `dot`.

Le dimensioni dei vettori devono essere le stesse.

Una alternativa computazionalmente più efficiente utilizza il prodotto $*$:

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b}$$

Le operazioni di divisione, moltiplicazione ed elevamento a potenza non sono definite per i vettori, ma è possibile utilizzare le **operazioni punto**: `.*` `./` `.^` `.\`

Vettori: operazioni con i vettori

Addizione e sottrazione non hanno bisogno del punto

```
>> w = [25, 16];
>> v = [5, 8];
>> w./v
ans =
     5     2

>> w.\v
ans =
    0.2000    0.5000

>> w.*v
ans =
    125    128

>> v.^2
ans =
     25     64

>> a = [2 3]; v.^a
ans =
     25    512
```

In questo modo viene eseguita l'operazione componente per componente.

Nel caso dell'elevamento a potenza, ogni elemento del vettore viene elevato all'esponente indicato: $\mathbf{v}.^2 = (5^2, 8^2)$.

Se si utilizza un vettore come esponente, il risultato è $\mathbf{v}.^{\mathbf{a}} = (5^2, 8^3)$.

Vettori: funzioni sui vettori

```
>> v = linspace(10,15,50);  
>> sum(v)  
ans =  
    625
```

La funzione `sum` calcola la somma degli elementi dell'argomento.

```
>> prod(v)  
ans =  
    4.9307e+54
```

La funzione `prod` calcola il prodotto degli elementi dell'argomento.

```
>> numel(v)  
ans =  
    50
```

La funzione `numel` restituisce il numero degli elementi dell'argomento.

```
>> length(v)  
ans =  
    50
```

La funzione `length` restituisce il massimo numero di elementi fra le dimensioni dell'argomento.

```
>> size(v)  
ans =  
     1     50
```

La funzione `size` restituisce un vettore riga con le dimensioni dell'argomento.

P.S. Quando verranno utilizzate matrici (e array multi-dimensionali in genere) le differenze fra queste ultime tre funzioni saranno più chiare...

Vettori: funzioni sui vettori

```
>> v = rand(10,1);  
>> max(v)  
ans =  
    0.9706  
>> min(v)  
ans =  
    0.1419
```

La funzione `rand(n,m)` genera una matrice $n \times m$ di valori campionati con distribuzione uniforme nell'intervallo $[0, 1]$.

Le funzioni `max` e `min` restituiscono, rispettivamente, il massimo e il minimo del vettore in input.

```
>> [mx, idx] = max(v)  
mx =  
    0.9706  
idx =  
     2  
>> [mn, idx] = min(v)  
mn =  
    0.1419  
idx =  
     6
```

Invocando le funzioni `max` e `min` con due parametri di uscita, si ottengono contemporaneamente in output il valore massimo (o minimo) del vettore e l'indice della posizione in cui è ottenuto.

```
>> mean(v), std(v)  
ans =  
    0.6602    0.3308
```

La funzione `mean` calcola la media del vettore e la funzione `std` la sua deviazione standard.

Vettori: funzioni sui vettori & operatori di confronto

```
>> w = 9 : 0.2 : 15;  
>> find(w < 10)  
ans =  
     1     2     3     4     5
```

La funzione `find(logic_expr)` restituisce gli indici per cui `logic_expr` è verificata.

Operatori di confronto

Gli operatori di confronto in Matlab sono:

- `==` confronto
- `>` maggiore
- `<` minore
- `>=` maggiore o uguale
- `<=` minore o uguale
- `~=` diverso

Nota: per digitare `~` in ambiente Windows: `ALT` + `1 2 6` (sul tastierino numerico); in ambiente Linux: `ALTGR` + `ì`; in ambiente MacOS: `ALT` + `5`.

Vettori: vettori e costanti

```
>> v = [4, 5];  
>> v + 2  
ans =  
     6     7  
>> v / 2  
ans =  
     2.0000     2.5000  
>> v * 2  
ans =  
     8    10
```

Se si eseguono operazioni fra vettori e numeri, Matlab automaticamente tratta il numero in questione come un vettore in cui ogni componente è pari alla costante digitata.

Esempio

Somma: il comando `v + 2` corrisponde a digitare `v + [2, 2]`. La corrispondenza rimane valida per qualsiasi altra operazione assegnata in questo modo.

Vettori: funzioni sui vettori

In Matlab sono definite tutte le funzioni elementari: sin, cos, log, ...
Per un elenco completo digitare nella shell di comando `help elfun`.

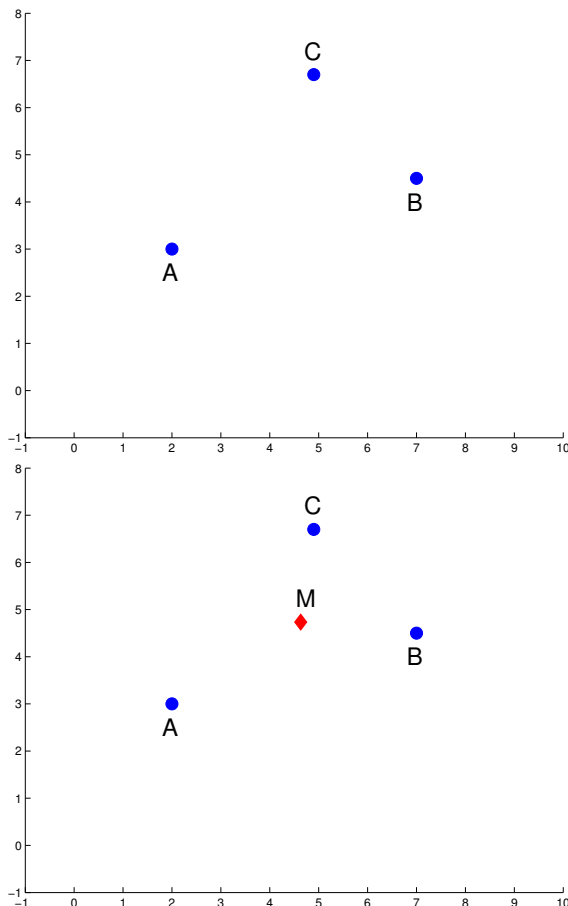
```
>> x = pi/3;  
>> sin(x)  
ans =  
    0.8660  
>> log(x)  
ans =  
    0.0461  
  
>> x = 0 : pi/2 : 3*pi/2  
>> max(x)  
ans =  
    4.7124  
>> min(x)  
ans =  
    0
```

Funzioni con argomenti vettoriali

Una funzione che ha come input un vettore restituisce come output un vettore della stessa dimensione, le cui componenti sono i valori della funzione calcolati sugli elementi del vettore di input.

```
>> x = 0 : pi/2 : 3*pi/2;  
x =  
    0    1.5708    3.1416    4.7124  
>> y = sin(x)  
y =  
    0    1.0000    0.0000   -1.0000
```

Vettori, esempio: calcolo del centro di massa



Si richiede il calcolo del centro di massa dei punti A,B,C. La formula per n punti è la seguente:

$$M = \left(\frac{1}{n} \sum_{i=1}^n x_i, \frac{1}{n} \sum_{i=1}^n y_i \right)$$

Matrici

L'oggetto principale su cui Matlab lavora è la **matrice**.

Per creare una matrice A come la seguente si utilizza la sintassi valida per i vettori:

$$A = \begin{pmatrix} 4 & 5 & \pi \\ 0 & 6 & 2 \\ 1 & 0 & e \end{pmatrix}$$

```
>> A = [4 5 pi; 0 6 2; 1 0 exp(1)]
```

```
A =
```

```
    4.0000    5.0000    3.1416  
         0    6.0000    2.0000  
    1.0000         0    2.7183
```

```
>> A(2,3) % Accesso all'elemento di riga 2 colonna 3
```

```
ans =  
    2
```

Carattere jolly :

I due punti `:` permettono di accedere a più elementi della matrice (e, quindi, anche di un vettore!) contemporaneamente: consentono di selezionare un'intera riga, un'intera colonna o una sottomatrice.

IMPORTANTE: data una matrice A , l'istruzione $A(:,)$ crea un vettore contenente gli elementi di A ordinati per colonne ("vettorizzazione").

Matrici

$$A = \begin{pmatrix} 2 & 5 & \pi & 56 & 2 \\ 0 & 6 & 2 & 0 & 2\pi \\ 1 & 0 & e & 3 & 5 \\ 9 & 0 & 3 & 5 & 2 \end{pmatrix}$$

```
>> A(:,1) % seleziona solo la prima colonna
```

```
ans =
```

```
    2  
    0  
    1  
    9
```

```
>> A(2,:) % seleziona solo la seconda riga
```

```
ans =
```

```
    0.0000    6.0000    2.0000    0.0000    6.2832
```

```
>> A(3:4,3:5) % seleziona gli elementi dalla 3a
```

```
ans = % alla 4a riga e dalla 3a alla 5a colonna
```

```
    2.7183    3.0000    5.0000  
    3.0000    5.0000    2.0000
```

Matrici: creazione

Si possono creare matrici “riempiendole” con vettori.

```
>> v = [2;4];
>> w = [0;1];
>> t = [1;3];
>> A = [v,w,t]
A =
     2     0     1
     4     1     3
>> B = [v;w;t]
B =
     2
     4
     0
     1
     1
     3
```

Si inizializzano tre vettori colonna di dimensioni 2×1 utilizzando il carattere “;”.

Per creare la matrice A si *affiancano* i tre vettori utilizzando il carattere “,”, in modo da avere una matrice 2×3 .

Se si utilizza il carattere “;”, i vettori vengono incolonnati e si crea un nuovo vettore di dimensione 6×1 .

```
>> v = [2;4];
>> w = [0;0;1];
>> A = [v,w]
Error using horzcat
CAT arguments dimensions are not
consistent.
```

Attenzione alle dimensioni dei vettori!

Matrici: creazione

Si possono creare matrici “riempiendole” con altre matrici.

```
>> A = [ 2 5 4; 6 9 0];
>> B = [ 9 9 9; 2 5 4];
>> C = [A;B]
C =
     2     5     4
     6     9     0
     9     9     9
     2     5     4
```

La matrice C è stata creata incolonnando la matrice B alla matrice A : rimane necessario prestare attenzione alle dimensioni.

Allocazione dinamica della memoria

Matlab è in grado di creare “spazio” (memoria) dinamicamente in caso di necessità.

```
>> D = [1 5; 5 6]
D =
     1     5
     5     6
>> D(1,4) = 9
D =
     1     5     0     9
     5     6     0     0
```

La matrice D era di dimensione 2×2 ma, avendo inizializzato l'elemento di posto (1,4), Matlab ha creato dinamicamente la memoria per gli elementi mancanti, inizializzandoli a 0.

Matrici: creazione di matrici speciali

```
>> eye(3)
```

```
ans =  
     1     0     0  
     0     1     0  
     0     0     1
```

Il comando `eye(n)` crea la matrice identità di ordine n .

```
>> zeros(3)
```

```
ans =  
     0     0     0  
     0     0     0  
     0     0     0
```

Il comando `zeros(n)` (`ones(n)`) crea una matrice di ordine n con ogni elemento inizializzato a 0 (1).

```
>> ones(2)
```

```
ans =  
     1     1  
     1     1
```

```
>> ones(3,7)
```

```
ans =  
     1     1     1     1     1     1     1  
     1     1     1     1     1     1     1  
     1     1     1     1     1     1     1
```

Matrici: esempio

Si ha a disposizione la matrice A di dimensioni 3×3 e si vuole creare la matrice strutturata nel seguente modo:

$$\begin{pmatrix} I_3 & 0_3 & -I_3 \\ 0_3 & A & 0_3 \\ -I_3 & 0_3 & I_3 \end{pmatrix}$$

Per creare tale matrice è possibile utilizzare un unico comando Matlab:

```
>> [eye(3)    zeros(3)   -eye(3)  
    zeros(3)    A        zeros(3)  
   -eye(3)    zeros(3)   eye(3)];
```

o anche

```
>> [eye(3), zeros(3), -eye(3); zeros(3), A, zeros(3); -eye(3), zeros  
    (3), eye(3)];
```

Matrici: operazioni

```
>> A = [ 2 5 4; 6 9 0];  
>> B = [ 9 9 9; 2 5 4];  
>> A+B
```

```
ans =  
    11    14    13  
     8    14     4
```

```
>> C = A - B  
C =  
    -7    -4    -5  
     4     4    -4
```

```
>> D = [2 6 ; 5 6; 0 0]  
D =  
     2     6  
     5     6  
     0     0
```

```
>> P = A*D  
P =  
    29    42  
    57    90
```

La somma fra matrici è definita componente per componente.

Il prodotto è definito per matrici di dimensioni opportune: se $A \in \mathbb{R}^{2 \times 3}$, $D \in \mathbb{R}^{3 \times 2}$ allora $P = AD \in \mathbb{R}^{2 \times 2}$.

$$\begin{aligned} P_{11} &= \langle A_1, D^1 \rangle \\ &= \langle (2, 5, 4), (2, 5, 0) \rangle \\ &= 2 \cdot 2 + 5 \cdot 5 + 4 \cdot 0 = \\ &= 29 \end{aligned}$$

Matrici: operazioni

```
>> A*B  
Error using *  
Inner matrix dimensions must agree  
.
```

Il prodotto deve essere fatto fra matrici di dimensioni opportune.

```
>> A\B  
ans =  
     0     0     0  
 0.2222 0.5556 0.4444  
 1.9722 1.5556 1.6944
```

```
>> A/B  
ans =  
 -0.0000 1.0000  
 0.3810 0.4286
```

$$A \setminus B = A^{-1}B, \text{ mentre } A/B = AB^{-1}.$$

(Queste due funzioni verranno riprese in dettaglio quando verranno trattati i sistemi lineari.)

```
>> A'  
ans =  
     2     6  
     5     9  
     4     0  
>> P^2  
ans =  
    3235    4998  
    6783   10494
```

$$A' = A^T.$$

L'operazione A^p con A matrice e $p \in \mathbb{N}$ è possibile solo quando A è quadrata.

Matrici: operazioni punto

Si possono effettuare operazioni componente per componente tra matrici utilizzando le operazioni "punto" `.*` e `./`

```
>> A = [ 2 5 4; 6 9 0], B = [ 9 9 9; 2 5 4]
```

```
A =
```

```
     2     5     4
     6     9     0
```

```
B =
```

```
     9     9     9
     2     5     4
```

```
>> E = A.*B
```

```
E =
```

```
    18    45    36
    12    45     0
```

$$E_{2,1} = A_{2,1} \cdot B_{2,1}$$

```
>> F = A./B
```

```
F =
```

```
    0.2222    0.5556    0.4444
    3.0000    1.8000         0
```

$$F_{2,1} = A_{2,1}/B_{2,1}$$

```
>> G = A.^3
```

```
G =
```

```
     8    125    64
    216    729     0
```

$$G_{1,1} = A_{1,1}^3$$

Matrici: confronto fra matrici

È possibile confrontare gli elementi di due matrici.

```
>> A = [ 2 5 4; 6 9 0];
```

```
>> B = [ 9 9 9; 2 5 4];
```

```
>> A == B
```

```
ans =
```

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

Il confronto viene fatto elemento per elemento.

```
>> A > B
```

```
ans =
```

```
     0     0     0
     1     1     0
```

Il risultato è una matrice i cui elementi sono valori logici:

0 significa **FALSE**, 1 significa **TRUE**.

```
>> A ~= B
```

```
ans =
```

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

Qualunque valore diverso da zero, quando interpretato da Matlab come valore logico, significa **TRUE**.

```
>> A <= B
```

```
ans =
```

```
     1     1     1
     0     0     1
```


Matrici: prodotto matrice-vettore

Il prodotto tra una matrice e un vettore può essere eseguito quando le dimensioni sono opportune:

se $A \in \mathbb{R}^{n \times m}$ e $\mathbf{x} \in \mathbb{R}^m$, allora il prodotto $A\mathbf{x}$ è ben definito e $A\mathbf{x} \in \mathbb{R}^n$.

```
>> A
A =
     2     5     4
     6     9     0
>> x = [2; 3; 0];
>> b = A*x
b =
    19
    39
```

$$b_1 = \langle A_1, x \rangle = 2 \cdot 2 + 5 \cdot 3 + 4 \cdot 0 = 19$$

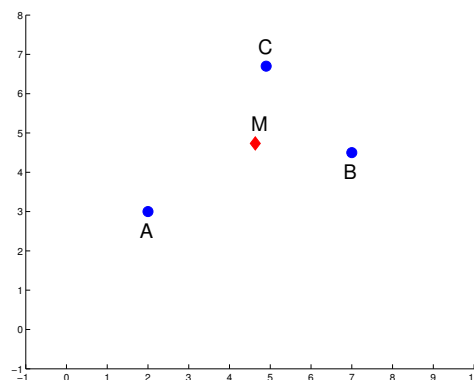
```
>> y = [5;6];
>> A*y
Error using *
Inner matrix dimensions must agree.
```

Attenzione alle dimensioni

M-script

Si consideri l'esempio del calcolo del centro di massa di tre punti. I comandi sono i seguenti:

```
>> A = [2.0; 3.0 ];
>> B = [7.0; 4.5 ];
>> C = [4.9; 6.7 ];
>>
>> M = (A + B + C) / 3;
```



Se si volessero cambiare le coordinate dei punti (oppure aggiungerne/toglierne uno/altri), allora si dovrebbe ripartire dall'inizio e scrivere tutte le istruzioni. . .

M-script files

... e invece no! Basta salvare la lista delle istruzioni in quello che viene chiamato **M-script file**. Una volta scritta questa lista e salvato il file, è possibile eseguire questa serie di comandi scrivendo il nome dell'M-file nella shell di comando.

Questo tipo di M-file prende il nome di **M-script**.

Nota bene:

- per eseguire un M-script, è necessario essere nella directory dove tale script è salvato;
- un M-script può accedere a tutte le variabile del workspace corrente, modificandole; tutte le variabili create nello script rimarranno in memoria;
- per rendere *effettive* le modifiche effettuate in un M-script, è necessario salvarle: quando le modifiche fatte non sono salvate, viene mandata in esecuzione l'ultima versione *salvata* dell'M-script. La presenza di un asterisco nella finestra dell'editor accanto al nome del file indica che sono presenti modifiche non salvate;
- per avviare un M-script, si può digitare il nome del file nella shell *oppure* cliccare sul pulsante a forma di “play” sulla barra di modifica dell'editor: in questo secondo modo, le modifiche vengono salvate e le istruzioni contenute nello script vengono eseguite;
- uno script può chiamare un altro M-script (p.es. una *M-function* (in seguito));
- le prime tre righe del **primo** M-script che viene chiamato, spesso contengono le seguenti istruzioni:

```
▶ clear all      % cancella tutte le variabili nel workspace
▶ close all      % chiude tutte le finestre aperte
▶ clc            % cancella tutti i comandi sulla shell
```

Costrutto *if-then-else*

Il costrutto `if--then--else` ha la seguente sintassi:

```
if condition1
    instructions_1
else
    instructions_2
end
```

Se `condition_1` è soddisfatta, allora la serie di istruzioni `instruction_1` viene eseguita, altrimenti viene eseguita la lista di istruzioni `instructions_2`.

```
if condition1
    instructions_1
elseif condition2
    instructions_2
.....
elseif condition_n
    instructions_n
else
    instructions
end
```

Se `condition_1` è soddisfatta, allora la serie di istruzioni `instruction_1` viene eseguita;

se `condition_2` è soddisfatta, allora la serie di istruzioni `instruction_2` viene eseguita e così via.

Se l'istruzione `else` non contiene istruzioni può essere omessa

Costrutto *switch-case*

Se sono presenti molti casi da controllare, è possibile utilizzare l'istruzione `switch` :

```
switch var
    case number_1
        instruction_1
    case number_2
        instructions_2
    .....
    otherwise
        instructions_o
end
```

(equivale a usare `elseif`)

Se nessuna delle precedenti è soddisfatta, allora vengono eseguite le istruzioni in `instructions_o`.

Esempio

Scrivere uno script MatLab che risolva l'equazione di secondo grado

$$ax^2 + bx + c = 0$$

dati i coefficienti a , b e c .

Ciclo *for*

Il ciclo `for` esegue una serie di istruzioni ripetute. Per esempio, se si assegnano i seguenti comandi

```
v = 1:10;
for ii = 1:10
    v(ii) = v(ii) - ii^2;
end
```

il ciclo scritto percorre tutto il vettore `v` e ad ogni elemento sostituisce l'elemento stesso diminuito di ii^2 .

```
v =
    0    -2    -6   -12   -20   -30   -42   -56   -72   -90
```

SINTASSI VETTORIALE

Per eseguire calcoli fra matrici e vettori, **evitare il più possibile i cicli `for`!!!** Matlab è creato per eseguire velocemente operazioni fra matrici e vettori in maniera automatica, l'utilizzo dei cicli `for` può far aumentare il tempo di calcolo in maniera notevole.

Il ciclo `while` esegue una serie di istruzioni finché viene soddisfatta una condizione assegnata.

```
while cond_1
    instructions
end
```

ATTENZIONE! Bisogna sempre assicurarsi che il ciclo `while` possa avere termine:

```
a = 1;
while a < 5
    b = a + 1;
end
```

Nell'esempio, l'istruzione `b = a + 1;` viene eseguita fino all'esaurimento della memoria disponibile sul calcolatore.

M-function

Le *M-function* sono particolari *M-files* che possono prendere parametri in input e restituirne altri in output, *senza modificare quelli in entrata*. Possono essere sviluppate dall'utente e utilizzate come le funzioni native di Matlab.

La sintassi di scrittura di una *M-function* è la seguente:

```
function [ output_args ] = function_name( input_args )
% Descrizione in una riga ("Summary" o "One-line description")
% Spiegazione dettagliata, significato dei parametri di I/O

..... corpo della function .....

end % Questo "end" e' opzionale
```

Il nome dell'M-file DEVE coincidere con il nome della *M-function* (nell'esempio sopra riportato il nome da assegnare è `function_name.m`).

Se una *M-function* viene chiamata da un *M-script*, essa deve trovarsi nella stessa cartella dell'*M-script*, ovvero in una delle cartelle contenute nell'elenco restituito dal comando `matlabpath`.

Esempio

Scrivere una *M-function* che restituisca come output le radici dell'equazione di secondo grado $ax^2 + bx + c = 0$, ricevendo in input i coefficienti a , b e c .

M-function

```
function [x1, x2] = roots_deg2(a, b, c)
%ROOTS_DEG2 - Calcolo delle soluzioni dell'eq. di II grado.
% SINTASSI: [x1, x2] = roots_deg2(a, b, c)
% INPUT: a, b, c (coefficienti)
% OUTPUT: x1, x2 (soluzioni)
x1 = []; x2 = []; % le variabili di output sono inizializzate vuote
Delta = b^2 - 4*a*c; % calcolo del discriminante
switch sign(Delta)
    case -1
        fprintf('L'eq. non ha soluzioni reali.\n')
        return; % uscita dalla funzione
    case 0
        fprintf('L'eq. ha due soluzioni reali coincidenti:\n')
        x1 = -b/(2*a);
        x2 = x1;
        fprintf(' x1 = x2 = %f \n', x1);
    case 1
        fprintf('L'eq. ha due soluzioni reali distinte:\n')
        x1 = (-b - sqrt(Delta))/(2*a);
        x2 = (-b + sqrt(Delta))/(2*a);
        fprintf(' x1 = %f \n x2 = %f \n', x1, x2);
end % termine del costrutto switch
end % termine della M-function
```

È sconsigliabile mandare in output messaggi di testo dal codice delle M-functions

M-function

Si osservi che le variabili *a*, *b* e *c* non vengono modificate: il passaggio delle variabili alle *M-functions* viene effettuato **per valore, non per indirizzo**. Il *workspace* contenente le variabili utilizzate dalle *M-functions* è **temporaneo** e viene liberato quando termina l'esecuzione delle istruzioni contenute nella *M-function* (le variabili non passate in output vengono eliminate).

Esempio

```
>> cff1 = 2;
>> cff2 = 56;
>> cff3 = 9;

>> [ sol1, sol2 ] = ...
    roots_2deg(cff1,cff2,cff3);

sol1 =
    -27.838353
sol2 =
    -0.161647
```

Nel workspace principale le variabili hanno i nomi *cff1*, *cff2* e *cff3*, mentre all'interno della funzione *roots_2deg* hanno i nomi *a*, *b* e *c*.

Gli output nel workspace principale vengono chiamati *sol1* e *sol2* mentre nel workspace della funzione sono *x1* e *x2*. Le variabili *a*, *b*, *c*, *x1*, *x2* esistono **solamente** per il tempo di esecuzione della funzione. I valori di *x1* e *x2* vengono passati in output nelle variabili *sol1* e *sol2*.

Nota: la stringa “...” permette di mandare a capo la scrittura di istruzioni lunghe (è possibile posizionarla solo in alcuni punti delle istruzioni!).

```
>> format short
>> format short e
>> format short g
>> format long
>> format long e
>> format long g
>> format rat
```

```
>> pi
>> Nan
>> Inf
>> realmin
>> realmax
```

Il comando `format` cambia il modo di visualizzare un numero:

il formato di default è `short`, `long` consente la visualizzazione di un maggior numero di cifre.

Il formato `rat` permette la visualizzazione dei numeri razionali come frazioni.

Matlab ha in memoria alcune costanti, come `pi` (π), `i` (i , l'unità immaginaria, memorizzata anche come `j`);

`Nan` significa “not-a-number” e rappresenta il risultato assegnato a operazioni come $0/0$ o ∞/∞ ;

`Inf` è la rappresentazione sul calcolatore di ∞ .

Cronometro e workspace, operatori logici

```
>> tic
% lista di istruzioni ...
>> toc
```

```
>> whos
```

```
>> exit
>> quit
```

Il comando `tic` inizializza il cronometro, il comando `toc` lo ferma.

Permette di visualizzare nella shell un elenco delle variabili memorizzate, contenente le dimensioni, il tipo, lo spazio di memoria utilizzato. . .

Comandi per uscire da Matlab.

Gli operatori logici in Matlab si scrivono nei modi seguenti:

- `&` o `&&`: AND
- `|` o `||`: OR
- `~`: NOT

```
>> A = fix(50*rand(3))
A =
     5     17     37
    24     29     12
    47     11     25
>> max(A)
ans =
    47     29     37
>> min(A)
ans =
     5     11     12
>> sum(A)
ans =
    76     57     74
>> sum(A,2)
ans =
    59
    65
    83
>> size(A)
ans =
     3     3
>> numel(A)
ans =
     9
```

Le operazioni di somma, prodotto, massimo e minimo sulle matrici vengono fatte sulle *colonne*: è possibile modificare l'orientamento all'occorrenza.

Consultare SEMPRE l'`help` (o il `doc`) di Matlab per avere informazioni sull'utilizzo delle funzioni!

Salvataggio e caricamento, arresto

```
>> save dati.mat
>> load dati

>> x = [1, 7, 9];
>> y = 2*pi;
>> z = cos(y);
>> save dati.mat x z

>> help elfun
```

Il comando `save` salva l'intera *workspace* nel file `dati.mat`.

Per caricare un file di dati salvati si utilizza il comando `load`.

È possibile salvare in un file `.mat` **solo** alcune variabili.

Visualizza l'elenco delle funzioni elementari implementate.

Arresto forzato

Qualora fosse necessario arrestare l'esecuzione dei comandi su Matlab prima del termine, si utilizza il comando `CTRL` + `C`.

Memorandum

- `clc`, `clear all`, `close all` nelle prime righe di ogni script principale
- "Commentate, gente, COMMENTATE!"
- `help` o `doc` per imparare a usare una function
- Utilizzare la sintassi vettoriale (pochi cicli `for`!)
- `CTRL` + `C` per forzare l'arresto di un'esecuzione

(PROVARE A) FARE GLI ESERCIZI ! ! !