

Introduzione a Matlab

Giacomo Elefante

Laboratorio di calcolo numerico
11/04/23

Sommario

- 1 Introduzione
- 2 Vettori e matrici
- 3 Operazioni e funzioni vettoriali/matriciali
- 4 Costruire funzioni matematiche
- 5 Grafici in Matlab
- 6 Functions ed altre strutture in Matlab

Cos'è Matlab

Matlab (MATrix LABoratory) è un ambiente software per il calcolo numerico creato da Cleve Moler alla fine degli anni settanta.

- In fondo, è una calcolatrice programmabile (potente).
- Ha un suo linguaggio di programmazione (relativamente facile).
- È molto utilizzato! Ci sono molte guide ed informazioni utili nel web.

In particolare in questo corso, lavoreremo con matrici e vettori (e valuteremo i risultati con dei grafici).

Lanciare Matlab in Laboratorio

In laboratorio, possiamo lanciare Matlab in due modi alternativi

- Aprirlo dalla lista programmi.
- Aprire una finestra del Terminale, scrivere `matlab &` e premere Invio.

Lanciare Matlab da casa

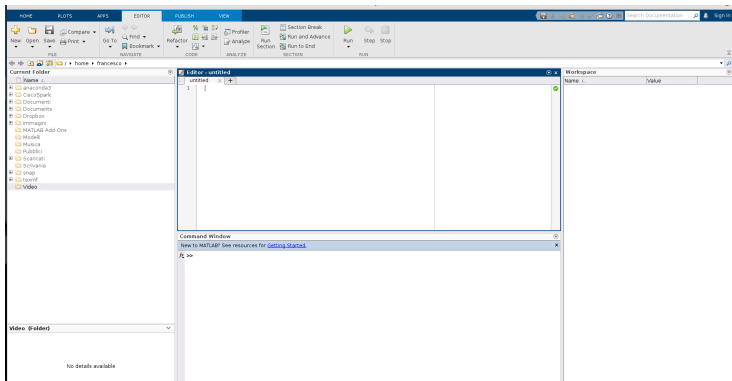
Da casa, possiamo lavorare in Matlab nei seguenti modi.

- Scaricare Matlab (abbiamo una licenza CAMPUS)
- Utilizzare Matlab Online.
- Utilizzare un ambiente simile tipo Octave (sconsigliato).

L'interfaccia di Matlab

Quando apriamo Matlab, abbiamo quattro *spazi* principali

- **Command window**: dove possiamo inserire dei comandi da eseguire.
- **Editor**: dove scriviamo del codice più “elaborato” , per poi lanciarlo o salvarlo.
- **Workspace**: dove vengono visualizzate le variabili.
- **Current folder**: la cartella nella quale siamo posizionati e che Matlab “vede” .



Command window

Proviamo a lanciare alcune semplici operazioni aritmetiche (le scriviamo e premiamo Invio)

```
Command Window
New to MATLAB? See resources for Getting Started.

>> 2 + 9 %Somma
ans =
    11

>> 3 - 5 %Sottrazione
ans =
    -2

>> 2 * (-6) %Prodotto
ans =
   -12

>> 1 / 16 %Divisione
ans =
    0.0625

>> 6^7 %Potenza
ans =
  279936

f1 >>
```

Figure: Alcune operazioni.

Numeri speciali

Oltre ai numeri macchina, Matlab ha dei numeri *speciali*

+Inf	<i>+ infinito (es. 5/0)</i>	-Inf	<i>- infinito (es. -5/0)</i>
NaN	<i>numero indefinito (es. 0/0)</i>	eps	<i>precisione di macchina</i>
pi	<i>pi greco</i>	i	<i>unità immaginaria</i>

Table: Un po' di numeri speciali.

Alcune funzioni utili

sign	<i>segno</i>	abs	<i>valore assoluto</i>
sqrt	<i>radice quadrata</i>	sin	<i>seno</i>
cos	<i>coseno</i>	tan	<i>tangente</i>
cotan	<i>cotangente</i>	asin	<i>arcoseno</i>
acos	<i>arcocoseno</i>	exp	<i>esponenziale</i>
log	<i>logaritmo naturale</i>	log10	<i>logaritmo in base 10</i>
log2	<i>logaritmo in base 2</i>	round	<i>arrotondamento</i>
floor	<i>parte intera</i>	ceil	<i>parte intera + 1</i>

Table: Un po' di funzioni.

Non ha senso impararsi troppe funzioni a memoria, quando servono possiamo cercare la sintassi corretta.

Variabili

Osserviamo che, ad ogni operazione lanciata, Matlab salva il risultato in una variabile `ans` visibile nel workspace. Possiamo generare delle variabili, assegnando un valore, con la sintassi

```
<nome variabile> = <valore variabile>
```

- Il nome della variabile deve essere alfanumerico, con qualche carattere speciale ammesso,
- non può cominciare con un numero,
- non può contenere spazi,
- non deve coincidere con strutture predefinite in Matlab (tipo `for`, `if` che vedremo).

Variabili in Matlab

```
>> a = 10

a =

    10

>> a = 10; %se metto il ; non ristampa il valore della variabile
>> b = -3;
>> c = a + 2*b

c =

     4

fx >> |
```

Figure: Esempi in Matlab.

Le variabili compaiono nel workspace. Due comandi utili da lanciare nella command window sono `clear all` (cancella tutte le variabili) e `clc` (pulisce la command window). Inoltre, usando `%` possiamo inserire dei commenti che non vengono elaborati da Matlab.

Stringhe in Matlab

Oltre alle variabili numeriche, abbiamo variabili di tipo stringa, dove l'argomento va racchiuso tra apici. Vediamolo con l'esempio in figura. Il comando `disp` stampa a schermo il valore di una variabile.

```
>> s = 'Hello world!';  
>> disp(s)  
Hello world!  
>> s_nope = hello;  
Unrecognized function or variable 'hello'.  
fx >> |
```

Figure: Definizione corretta ed errata di stringa.

Se non usiamo gli apici, Matlab pensa che `hello` sia il nome di una variabile che abbiamo definito. Se non l'abbiamo definita, si lamenta che non sa di cosa si tratta.

Editor di Matlab

Prima di procedere con vettori e matrici, vediamo gli altri spazi nell'interfaccia di Matlab, ovvero l'Editor e la Current folder.

Con l'Editor possiamo fare due cose principalmente, che sono spesso motivo di confusione perché si assomigliano ma sono sostanzialmente diverse.

- Creare degli **script**.
- Creare delle **function**.

Per il momento ci focalizziamo sugli script, introdurremo le function più avanti.

Script in Matlab

Lavorando nella Command window, possiamo eseguire comodamente delle operazioni *brevi*. Per costruire qualcosa di più elaborato e salvare il lavoro nel PC, utilizziamo l'Editor per comporre uno script.

Uno script non è altro che un foglio di istruzioni ed operazioni che possiamo comporre, salvare, e che possiamo successivamente passare a Matlab perché le esegua sequenzialmente. Vediamolo con un esempio.

Esercizio

Scriviamo uno script che assegna il numero 2 alla variabile *a* ed il numero 3π alla variabile *b* e che calcola successivamente il seno del prodotto di queste variabili, stampando poi a schermo il risultato.

Esempio di script

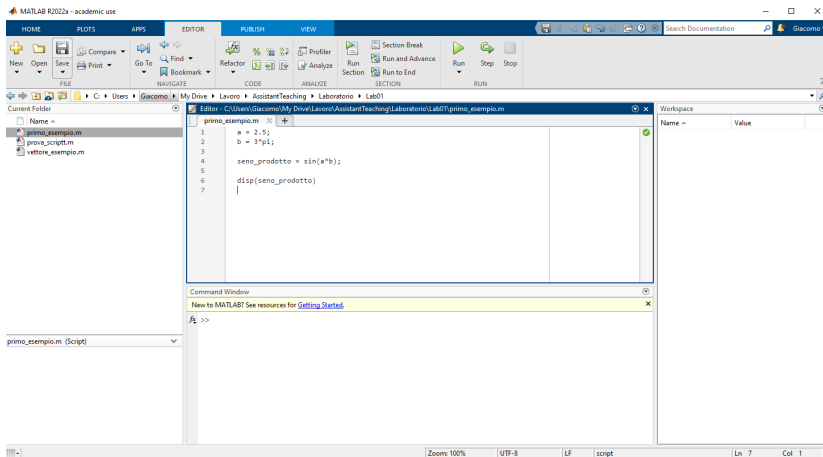


Figure: Scriviamo le operazioni nell'editor (seguendo un ordine sequenziale).

Vettori in Matlab

In Matlab, un vettore con n componenti è rappresentato come una matrice $n \times 1$ o $1 \times n$ (vettore colonna o riga). L'operazione di trasposizione si ottiene con l'apostrofo '. Una variabile numerica scalare è interpretata come un vettore 1×1 (lo si vede anche nel workspace). Usando le virgole cambio colonna, con ; cambio riga. Dobbiamo usare le parentesi quadre.

```
>> v = [1,3,-2]

v =

     1     3    -2

>> w = [1;3;-2]

w =

     1
     3
    -2

>> u = w'

u =

     1     3    -2

fx >>
```

Figure: Esempi di vettori.

Vettori equispaziati

Spesso avremo bisogno di definire vettori con componenti numeri equispaziati. Chiaramente, se il numero di componenti è grande, non possiamo definirli a mano.

- La funzione `linspace(a,b,n)` ci permette di definire un vettore di n componenti che spazia da a a b .
- Il costrutto `a:s:b` definisce un vettore i cui valori partono da a e continuano di passo s fino al massimo valore minore o uguale di b .

I due costrutti sono molto simili e intercambiabili anche se differenziano di qualcosa. Ad esempio, usando $s = (b - a)/(n - 1)$ si ottiene con il secondo comando il vettore generato dal primo.

Vettori concatenati

Due vettori possono essere concatenati utilizzando la sintassi già presentata. Se v è un vettore $1 \times n$ e w è un vettore $1 \times m$, possiamo definire un vettore di concatenazione u è un vettore $1 \times (n + m)$ come

$$u = [v, w] .$$

Se abbiamo vettori colonna, ad esempio se p è un vettore $t \times 1$ e r è un vettore $q \times 1$, dobbiamo usare il ; nella concatenazione e definire il vettore concatenazione s di dimensione $(t + q) \times 1$ come

$$s = [p; r] .$$

Matrici in Matlab

Analogamente ai vettori, possiamo definire delle matrici $m \times n$ componente per componente. Ad esempio

$$A = [1,2,3;4,5,6] .$$

Mi costruisce la matrice 2×3

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Osservazioni e matrici utili

- Possiamo concatenare e trasporre matrici, rispettando però i vincoli di dimensionalità.
- Per Matlab, i vettori sono casi particolari di matrici e tante operazioni relative si sovrappongono (vediamo tra un attimo).
- Ci sono comandi preimpostati per matrici speciali.
 - ▶ `zeros(m,n)` crea una matrice $m \times n$ di zeri.
 - ▶ `ones(m,n)` crea una matrice $m \times n$ di uni.
 - ▶ `eye(n)` crea una matrice identità $n \times n$.

Funzioni utili per vettori e matrici

Possiamo accedere facilmente a basilari informazioni riguardanti vettori e matrici.

- Se v è un vettore, `length(v)` ci restituisce il numero di componenti di v .
- Se A è una matrice, `size(A)` ci restituisce le dimensioni di A . La funzione ha un input opzionale successivo per specificare la dimensione richiesta della matrice, ad esempio `size(A,2)` mi restituisce il numero di colonne di A .
- Se v è un vettore, scrivendo `v(i)` accediamo all'elemento i -esimo.
Gli indici partono dal valore 1.
- Se A è una matrice, scrivendo `A(i,j)` accediamo all'elemento (i,j) -esimo.
Gli indici partono dal valore 1.

Come già osservato, operazioni tra matrici e vettori non sono entità del tutto distinte in Matlab.

Funzioni nativamente vettoriali

Le operazioni e le funzioni matematiche che abbiamo menzionato possono essere applicate in modo vettoriale. Per esempio, se v è un vettore $1 \times n$, scrivendo $\sin(v)$ applichiamo la funzione seno componente per componente.

```
>> v = [0,pi/2,pi,3/2*pi,2*pi];  
>> w = sin(v)  
  
w =  
  
    0    1.0000    0.0000   -1.0000   -0.0000  
  
↳ >> |
```

Figure: Funzione seno applicata vettorialmente.

Operazioni vettoriali

Tuttavia, alcune operazioni tra vettori possono condurre a scritture ambigue. Se scrivo $*$, si intende il prodotto componente per componente (o **puntuale**) oppure, per esempio, un prodotto scalare? Facciamo luce sulla questione.

Siano v e w vettori (o matrici) della stessa dimensione.

- $v+w$ restituisce la somma dei due vettori;
- $v-w$ restituisce la differenza dei due vettori;
- $v.*w$ restituisce il prodotto puntuale dei vettori (infatti c'è il punto);
- $v./w$ restituisce la divisione puntuale dei vettori (infatti c'è il punto);
- $v.^w$ restituisce l'elevamento a potenza puntuale dei vettori (infatti c'è il punto).

Le operazioni componente per componente tra vettori possono essere fatte nel caso di vettori di dimensione uguale

Operazioni vettore/scalare

Sia v un vettore e c uno scalare. Allora scrivere $v + c$, $v - c$, $v .* c$, $v ./ c$, $v .^c$ effettua operazioni puntuali come se c fosse un vettore delle dimensioni di v con componenti tutte uguali.

In questo caso, per la moltiplicazione e per la divisione possiamo anche fare a meno del punto, ma non per la potenza!

Per cui è spesso più conveniente *abituarsi al punto* in queste situazioni.

Operazioni vettore/scalare in Matlab

```
>> v = [1,3,-1,0];  
>> c = 2;  
>> v + c  
  
ans =  
  
     3     5     1     2  
  
>> v - c  
  
ans =  
  
    -1     1    -3    -2  
  
>> v * c  
  
ans =  
  
     2     6    -2     0  
  
>> v / c  
  
ans =  
  
    0.5000    1.5000   -0.5000     0
```

Figure: Alcune operazioni vettore/scalare. Abbiamo ommesso il punto.

Operazioni vettore/scalare in Matlab

```
>> v ^ c
Error using ^
Incorrect dimensions for raising a matrix to a power. Check that the matrix is square and the power
is a scalar. To operate on each element of the matrix individually, use POWER (.^) for elementwise
power.

>> v .^ c
ans =

     1     9     1     0
```

Figure: Per la potenza Matlab ci richiede espressamente l'utilizzo del punto, fornendoci un errore altrimenti. Osservate come si riferisce a v come una matrice.

Prodotto matriciale e scalare

Ricordiamo brevemente la seguente definizione.

Definizione

Siano A una matrice $m \times n$ e B una matrice $n \times k$, $m, n, k \in \mathbb{N}$. Definiamo il **prodotto matriciale** di A e B la matrice $m \times k$ $P := AB$ tale che

$$p_{ij} = \sum_{\ell=1}^n a_{i\ell} \cdot b_{\ell j}, \quad 1 \leq i \leq m, 1 \leq j \leq k.$$

Il prodotto matriciale in Matlab si esprime come $P = A*B$. Inoltre, se v e w sono due vettori $1 \times n$, il loro prodotto scalare lo calcoliamo come $p = v * w'$. Analogamente si calcola il prodotto matrice-vettore.

Prodotto matriciale e scalare in Matlab

```
>> A = [2,3,1;0,1,2]

A =

     2     3     1
     0     1     2

>> B = 2*eye(2) % prodotto matrice per uno scalare

B =

     2     0
     0     2

>> P = A * B
Error using *
Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix
matches the number of rows in the second matrix. To operate on each element of the matrix
individually, use TIMES (.*) for elementwise multiplication.

Related documentation

>> P = B * A

P =

     4     6     2
     0     2     4

>>
```

Figure: Esempio di prodotto matriciale in Matlab. Se le dimensioni non sono opportune, Matlab dà errore e ipotizza cosa intendessimo.

Prodotto matriciale e scalare in Matlab

```
>> v = [1,-2,1]

v =

     1     -2     1

>> w = [0,2,0]

w =

     0     2     0

>> p = v * w
Error using *
Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix
matches the number of rows in the second matrix. To operate on each element of the matrix
individually, use TIMES (.*) for elementwise multiplication.

Related documentation

>> p = v * w'

p =

    -4
```

Figure: Esempio di prodotto scalare in Matlab. Vale quanto detto per il prodotto matriciale.

Esercizio

Produrre uno script dove vengono definiti tre vettori u , v e w di dimensione 1×5 . Si definisca v con componenti equispaziate in $[0, 1]$ (gli altri possono essere definiti a piacere). Si calcoli poi prima il prodotto scalare p tra u e v e poi si definisca un nuovo vettore z ottenuto moltiplicando w per lo scalare p trovato. Infine, data $A = [0, 2, -1, 2, 0; 1, 1, 1, 0, 0]$ si calcoli e stampi a schermo il prodotto matrice-vettore tra A e z .

Funzioni personalizzate

Oltre alle funzioni matematiche menzionate, possiamo costruirne di più elaborate utilizzando le *anonymous functions*. Supponiamo di voler chiamare f la funzione $f(x) = \sin^2(x) + 2\log(x^2 + 1)$. Il costrutto per le anonymous function è quindi

```
Nome_Variabile = @(Variabili) Funzione(Variabili)
```

da cui

```
f = @(x) (sin(x)).^2 + 2.*log(x.^2+1)
```

Dopo la chiocciola, indichiamo la/le variabile/i della funzione. Inoltre, utilizziamo una formulazione vettoriale in modo tale da poter utilizzare la funzione f anche su vettori.

Osserviamo che la variabile della funzione è definita solo “internamente”: la x non è definita al di fuori f (ci torneremo).

Funzioni personalizzate in Matlab

```
>> f = @(x) (sin(x)).^2 + 2.*log(x.^2+1);
>> t = linspace(0,1,6);
>> f(t)

ans =

    0    0.1179    0.4485    0.9338    1.5040    2.0944

>> f = @(x) (sin(x))^2 + 2.*log(x^2+1); % Proviamo con una definizione non vettoriale
>> f(t)
Error using ^
Incorrect dimensions for raising a matrix to a power. Check that the matrix is square and the power
is a scalar. To operate on each element of the matrix individually, use POWER (.)^ for elementwise
power.

Error in @(x)(sin(x))^2+2.*log(x^2+1)

>> f(4) % Con gli scalari invece funziona!

ans =

    6.2392

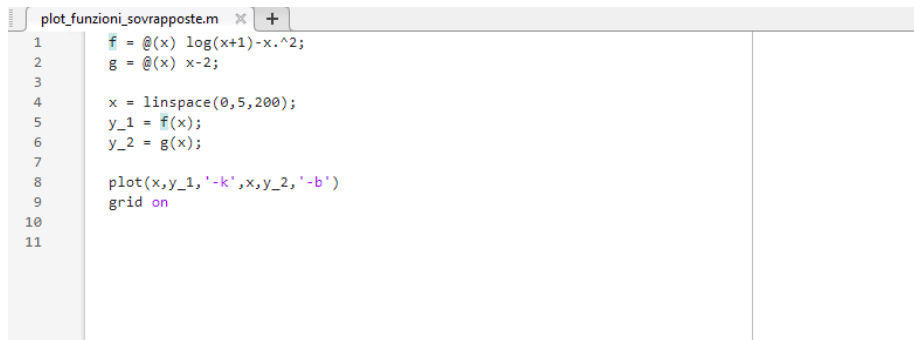
>> |
```

Figure: Esempio di funzione personalizzata in Matlab. Vediamo che la formulazione vettoriale è essenziale se si vuole lavorare puntualmente sui vettori, altrimenti troviamo un errore!

Il comando plot

Per i nostri propositi, sarà molto importante rappresentare funzioni e risultati ottenuti tramite dei grafici. Per questo scopo, è presente la funzione `plot`.

- La funzione vuole in input *vettori* di uguali dimensione.

A screenshot of a MATLAB script editor window. The title bar shows the file name 'plot_funzioni_sovrapposte.m' and a '+' icon. The script contains 11 lines of code. Line 1: `f = @(x) log(x+1)-x.^2;` Line 2: `g = @(x) x-2;` Line 3: (empty line) Line 4: `x = linspace(0,5,200);` Line 5: `y_1 = f(x);` Line 6: `y_2 = g(x);` Line 7: (empty line) Line 8: `plot(x,y_1,'-k',x,y_2,'-b')` Line 9: `grid on` Line 10: (empty line) Line 11: (empty line)

```
plot_funzioni_sovrapposte.m  +
1  f = @(x) log(x+1)-x.^2;
2  g = @(x) x-2;
3
4  x = linspace(0,5,200);
5  y_1 = f(x);
6  y_2 = g(x);
7
8  plot(x,y_1,'-k',x,y_2,'-b')
9  grid on
10
11
```

Figure: Esempio di plot sovrapposti.

Plot

Il comando

```
plot(x,y,x,z)
```

permette di sovrapporre più grafici contemporaneamente.

Altrimenti, il comando `hold on` dopo il plot consente di tenere un grafico e sovrapporlo ad un plot successivo. `hold off`, naturalmente, ha l'effetto contrario.

Inoltre, il comando `figure` prima del plot ha l'effetto di determinare una nuova finestra per il plot. In questo modo, è comodo generare più grafici senza sovrapposizioni.

Grafico in scala semilogaritmica

Il comando `semilogy(x,y)` permette di costruire un grafico in scala semilogaritmica.

Per vederlo meglio, affrontiamo il seguente esercizio.

Esercizio

Sia $f(x) = \sin(x)$ nel dominio $I = [0, 2\pi]$. Consideriamo il polinomio di Taylor di grado 3 $p(x) = x - x^3/6$. È noto che tale polinomio approssima f in un intorno di 0. Vogliamo descrivere l'errore assoluto $|f(x) - p(x)|$ per $x \in I$. Rappresentiamo graficamente nel dominio I .

- 1 Le funzioni f e p , nello stesso grafico.
- 2 La funzione $err(x) = |f(x) - p(x)|$.
- 3 Ancora $err(x)$ ma in scala semilogaritmica

Commenti a riguardo

- Plottando f e p nello stesso grafico, ricaviamo qualche informazione **qualitativa**: vediamo che effettivamente si assomigliano vicino a 0 per poi allontanarsi. Informazioni di questo tipo saranno molto comode (ad esempio, per capire se una funzione ha uno zero da qualche parte)
- Mostrando l'errore con `plot`, otteniamo qualcosa di più **quantitativo**. Però vicino a 0 non si capisce quantitativamente cosa succede (sbagliare di un centesimo o di un miliardesimo cambia, ma non è possibile distinguere dal grafico cosa succede).
- Utilizzando `semilogy`, abbiamo informazioni molto più precise su come si comporta l'errore.

Esistono anche `semilogx` (semilogaritmico ma nell'asse x) e `loglog` (completamente logaritmico), possono esserci utili.

Esercizio per casa

Esercizio

Sia $f(x) = \cosh(x)$ (coseno iperbolico) nel dominio $I = [-3, 3]$.
Consideriamo il relativo polinomio di Taylor $p(x)$, centrato in 0, di grado 4.
Ripetere l'esercizio che abbiamo svolto in classe riguardo l'errore assoluto tra f e p .

Function

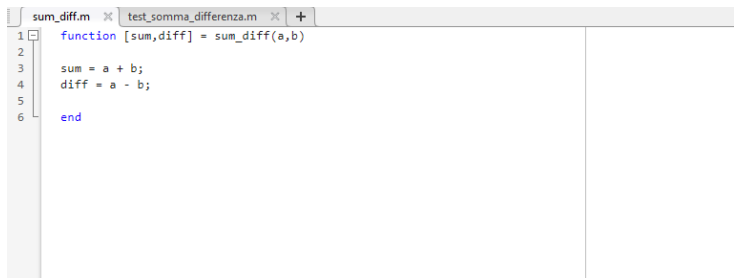
Abbiamo visto come nell'Editor sia possibile comporre degli script per poi essere eseguiti. Tramite l'Editor è anche possibile scrivere delle **function** matlab.

Una function è un file Matlab (.m) che inizia con

```
function [<output>] = <nome_funzione>(<input>)
```

e termina con `end`. Il ruolo della function è quello di prendere degli input, magari da uno script, elaborarli, e ritornare degli output. Vediamolo con un esempio.

Function in Matlab

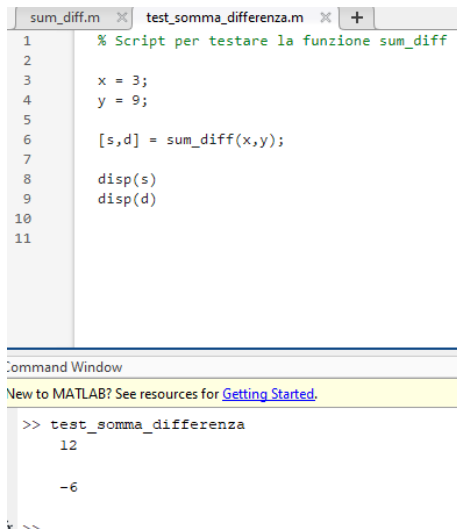


The screenshot shows the MATLAB script editor with two tabs: 'sum_diff.m' and 'test_somma_differenza.m'. The 'sum_diff.m' tab is active, displaying a function definition. The code is as follows:

```
1 function [sum,diff] = sum_diff(a,b)
2
3     sum = a + b;
4     diff = a - b;
5
6     end
```

Figure: Esempio di una function.

Function in Matlab



The image shows a MATLAB script editor with two tabs: 'sum_diff.m' and 'test_somma_differenza.m'. The 'test_somma_differenza.m' tab is active, displaying a script that calls the 'sum_diff' function. Below the script editor is the 'Command Window', which shows the output of the script execution.

```
1 % Script per testare la funzione sum_diff
2
3 x = 3;
4 y = 9;
5
6 [s,d] = sum_diff(x,y);
7
8 disp(s)
9 disp(d)
10
11
```

Command Window

New to MATLAB? See resources for [Getting Started](#).

```
>> test_somma_differenza
    12
    -6
```

Figure: Utilizziamo la function in uno script.

- Osservate come nella Current folder script e function hanno un'icona diversa. Il file relativo ad una function deve essere salvato come il nome della function stessa.
- Se provate ad eseguire direttamente (per esempio con Run) una function, Matlab vi darà un errore. Infatti, state eseguendo qualcosa che necessita di input, ma non specificate quali sono.
- Tutto quello che viene definito all'interno di una function non è visibile al di fuori di essa!

- Quando chiamiamo una function in uno script, l'input e l'output possono chiamarsi in modo diverso rispetto a come sono definiti all'interno della funzione. L'importante è che siano del “numero” giusto e del tipo giusto.
- In questo senso, è importante che l'ordine degli input sia rispettato.

Operatori logici

Matlab gestisce anche variabili di tipo logico, assegnando il valore 1 al **Vero** e 0 al **Falso**. Nella seguente tabella la sintassi dei vari operatori.

<code>==</code>	<i>uguale</i>	<code>~=</code>	<i>diverso</i>
<code>></code>	<i>maggiore</i>	<code><</code>	<i>minore</i>
<code>>=</code>	<i>maggiore uguale</i>	<code><=</code>	<i>minore uguale</i>
<code>&&</code>	<i>and</i>	<code> </code>	<i>or</i>
<code>~</code>	<i>not</i>		

Table: Operatori di relazione e logici.

Operazioni logiche in Matlab

```
>> 1 == 0  
ans =  
logical  
0  
  
>> (2 == 2) && (1 == -1)  
ans =  
logical  
0  
  
>> (2 == 2) || (1 == -1)  
ans =  
logical  
1  
  
>> (2 == 2) && ~(1 == -1)  
ans =  
logical  
1
```

Figure: Operazioni logiche in Matlab.

Qualche commento

- Esistono anche gli operatori `&` e `|`, ma lavorano componente per componente.
- A volte, mescolare operazioni tra numeri floating point ed operatori logici può essere uno sport pericoloso. Per esempio, cosa vi dice Matlab riguardo l'affermazione $3*0.2 == 0.6$?

Cicli ed istruzioni condizionali

Per concludere questa introduzione, trattiamo delle strutture fondamentali per la programmazione non solo in Matlab, ma anche negli altri linguaggi. Durante tutto il corso, potrà capitare che vengano introdotte man mano delle funzioni o strutture aggiuntive (spiegandole ovviamente), ma con le prossime slides otteniamo le basi sostanziali di ciò che ci serve.

Istruzioni condizionali

Il costrutto **if-else** ci permette di eseguire un'operazione se una certa condizione è soddisfatta, oppure di svolgerne un'altra in alternativa. La sintassi è la seguente.

```
if <condizione>  
    <processo A>  
else  
    <processo B>  
end
```

È obbligatorio concludere la struttura con **end**, mentre il ramo alternativo dato da **else** non è necessario. Per indicare più rami opzionali si può utilizzare il comando **elseif**. Oppure istruzioni condizionali multiple.

Istruzioni condizionali

A volte può essere utile il comando `switch` che a seconda del valore di una variabile permette di eseguire una porzione di programma. La sintassi è la seguente.

```
switch <espressione>
case <valore 1>
    <processo A>
case <valore 2>
    <processo B>
...
otherwise
    <processo otherwise>
end
```

È obbligatorio concludere la struttura con `end`.

Esercizio in Matlab

Esercizio

Si scriva una function che, dato in input un numero diverso da zero, restituisce il suo segno (senza utilizzare la funzione `sign`). Se l'input è uguale a zero, la funzione deve ritornare zero e stampare a schermo il messaggio 'Input uguale a zero!'.

Ciclo for

Il costrutto `for` ci permette di eseguire un certo codice ripetutamente al variare di un certo indice. La sintassi è la seguente.

```
for <variabile>=<vettore_variabile>  
    <processo>  
end
```

Nella condizione, la variabile viene posta uguale al vettore relativo ai valori da “scorrere”.

Ciclo for in Matlab

```
ciclo_for.m
1  clear all
2  clc
3
4  s = 2;
5
6  for i = 1:5
7      aumento = i^2;
8      disp(s+aumento)
9  end
10
11  % Equivalentemente
12
13  v = 1:1:5;
14
15  for i = v
16      aumento = i^2;
17      disp(s+aumento)
18  end
19
```

Figure: Non è necessario definire la variabile di iterazione *i* prima del ciclo, è definita nella condizione stessa.

Esercizio in Matlab

Esercizio

Si costruisca uno script dove si sommano i primi n numeri naturali, con $n=10$, attraverso un ciclo for e verificare che il valore corrisponde a $n(n+1)/2$.

Ciclo while

Il costrutto `while` ci permette di eseguire un certo codice ripetutamente fino a quando una certa condizione resta soddisfatta. La sintassi è la seguente.

```
while <condizione>  
    <processo>  
end
```

Mentre con il ciclo `for` imponiamo il numero esatto di iterazioni, il `while` ripete finché trova vera la condizione. Le due cose possono risultare equivalenti, ma la filosofia è diversa. Prima un esempio, poi approfondiamo.

Ciclo while in Matlab

```
ciclo_while.m  X  +
1      clear all
2      clc
3
4      s = 2;
5      k = 1; % Per usare il while, bisogna definire prima la variabile
6
7      while k <= 5
8          aumento = k^2;
9          disp(s+aumento)
10
11         k = k + 1; % Ricordarsi di incrementare la variabile di iterazione
12     end
13
```

Figure: La variabile di iterazione i va definita prima del ciclo, ed incrementata “manualmente”. Questo script riproduce gli stessi risultati di quello precedente con il ciclo `for`.

Commenti importanti

- Il ciclo for ha il vantaggio di stabilire all'inizio quante saranno le iterazioni.
- Il ciclo while è più flessibile, ma dobbiamo stare attenti a due aspetti per evitare loop infiniti: ricordarci di aggiornare la variabile di iterazione ed assicurarci che prima o poi la condizione di ripetizione venga infranta.
- Nell'usare il ciclo while, introdurremo una condizione ragionevole che porrà un limite al numero di iterazioni possibili.
- Se ci troviamo in un loop infinito, il comando **Ctrl+C** interrompe l'attività di Matlab.

Esercizio

Esercizio

Ispirandosi al codice presentato sotto, si produca uno script che permette di stimare la precisione di macchina con una base qualsiasi. Si preveda inoltre che il numero massimo di iterazioni possibili sia 1000. Infine, si stampino a schermo i risultati.

```
e=1;    k=0;
        while (e+1 > 1)
            e=e/2; k=k+1;
        end
e=2*e    % è necessario perché si era diviso per 2
k-1     % da l'esponente
```

Table: Calcolo della precisione macchina in base 2

Esercizio (soluzione)

```
precisione_while.m  +
1  clear all
2  clc
3
4  b = 2; % Base
5  iter = 0;
6  s = 1;
7  max_it = 1000; % Mettiamo un numero massimo di iterazioni possibili
8
9  while (1 + s > 1) && (iter <= max_it)
10     s = s/b;
11     iter = iter + 1;
12 end
13
14 s = s*b;
15 prec = (-(iter-1));
16
17 disp(s)
18 disp(prec)
19
20
21
```

Figure: Una possibile soluzione dell'esercizio proposto.

Esercizio per casa

Esercizio

Si consideri il prodotto di Wallis che ci dice che

$$\pi = 2 \prod_{n=1}^{\infty} \frac{2n}{2n-1} \cdot \frac{2n}{2n+1} .$$

- 1 Si costruisca un ciclo for che calcola la produttoria fino al valore $N = 500$. Ad ogni iterazione, ci si salvi il valore temporaneo assunto prodotto.
- 2 Si produca un plot dell'andamento del valore del prodotto rispetto alle iterazioni, ed un altro plot in scala semilogaritmica dell'errore tra π e i valori del prodotto nelle varie iterazioni

