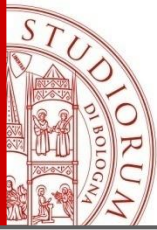


Introduzione all'ambiente MATLAB

Parte II



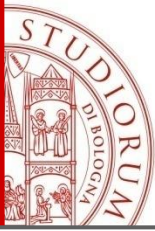
Programmazione Matlab

MATLAB non è un vero e proprio linguaggio di programmazione, ma permette comunque di realizzare programmi utilizzando le classiche strutture di programmazione come i cicli, i flussi di controllo e la gestione input/output.

Due tipi di programmi, noti come “m-file”,
(es. sort.m)

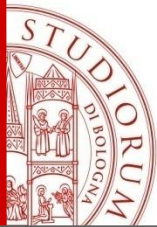
- script**

- function**



Programmazione Matlab

- Per creare un m-file si seleziona **New/m-file** dal menu **File**.
- **Script e funzioni** vengono quindi **inseriti mediante l'editor** di MATLAB. Si **salva poi il file** con il **suffisso .m**
- **N.B.** Assicurarsi di aver selezionato dalla *current directory* il percorso (path) relativo alla directory (cartella) di lavoro contenente l'm-file sul quale si vuole lavorare.



Errori comuni

■ Errori di sintassi

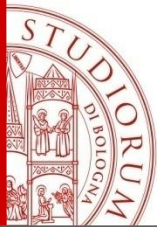
nell'uso del linguaggio di programmazione. **LEGGERE il messaggio visualizzato in rosso nella Command Window** aiuta a capire che tipo di errore è stato fatto.

■ Interruzione dell'esecuzione

Sono causati da overflow, divisioni per zero,.. Istruzioni illegali

■ Errori nel risultato

Sono causati da errori o nell'algoritmo o nella sua implementazione in MATLAB. Si correggono facendo il “**debug**” del programma, cioè seguendo il programma istruzione per istruzione, prevedendo il risultato dell'istruzione e controllando, tramite scrittura su video, i valori interessati nell'istruzione (basta eliminare il “;” alla fine dell'istruzione).

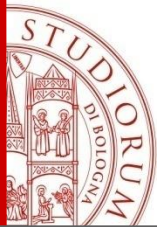


Script

Uno **SCRIPT** è una lista di comandi MATLAB che può eventualmente richiamare funzioni MATLAB built-in o create utilizzando altri m-file.

- Non richiede input
- Non fornisce output espliciti
- Tutte le variabili usate sono globali
(disponibili nel workspace)
- Simile ad un programma principale
- .m file deve essere disponibile nel proprio path corrente

ESEGUIRE UNO SCRIPT: *Per eseguire i comandi contenuti nello script (cioè richiedere a MATLAB che il file venga interpretato) si richiama semplicemente il suo nome dalla Command Window (senza estensione .m) e invio.*



Esempio

Script che calcola la media fra 4 variabili:

- Scrivere nell'editor lo script

```
a=10;
```

```
b=20;
```

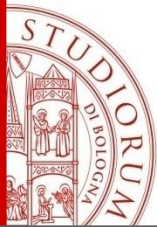
```
c=30;
```

```
d=40;
```

```
media=(a+b+c+d)/4;
```

- Salvare l'm-file come ***prova.m***
- Eseguire lo script digitando nella Command Window ***prova***

NOTA: Matlab non richiede alcun tipo di dichiarazione o dimensionamento a differenza dei vari linguaggi di programmazione (C, Fortran,...). L'assegnazione coincide con la dichiarazione



Function

Sintassi:

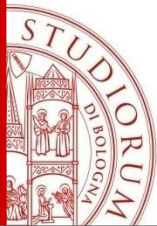
La prima riga dell'm-file che contiene una function deve essere del tipo

function [output] = *functionname* (input)

Le variabili di uscita **[output]**, fra parentesi quadre, e di entrata **(input)**, fra parentesi tonde, sono separati da una virgola.

Il nome dell'm-file deve essere il nome dato alla funzione:
functionname.m

Le variabili al suo interno sono viste solo localmente dalla funzione stessa e non dall'eventuale m-file chiamante o dall'ambiente MATLAB che la richiama.



Function - esempio

Variabili

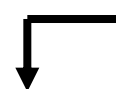
output



Nome function



Parametri input



```
function [media,stdev] = stat(x)
```

```
% esempio di function.
```

```
n = length(x);
```

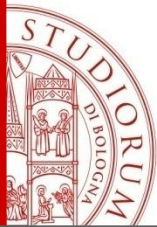
```
media = sum(x) / n;
```

```
stdev = sqrt(sum((x - media).^2)/n);
```

Definisce una nuova funzione STAT che calcola la media e la deviazione standard di un vettore x. Le variabili nel corpo della funzione sono tutte variabili locali.

```
[m,s]=stat (1:5)
```

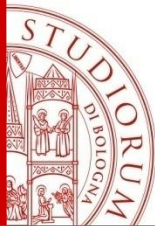
Richiamo della function
da command window o da
uno script o da altra function



Funzioni secondarie

- Uno stesso file .m può contenere più funzioni:
 - la function principale sta all'inizio e dà il nome al file;
 - Seguono le function secondarie.
 - Soltanto la function principale può essere richiamata da altre function esterne al file o dal prompt.
- **Esempio:**

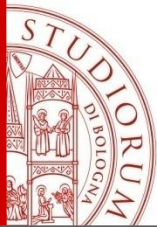
La funzione **polyGeom** mostra l'uso di una funzione principale che richiama due funzioni secondarie per calcolare area e perimetro di un poligono regolare con numero di lati e lunghezza assegnata.



Esempio (in file polyGeom.m)

```
function [a,p] = polyGeom(s,n)
% polyGeom Compute area and perimeter of a regular
% polygon
% Input:  s = length of one side of the polygon
%         n = number of sides of the polygon
%
% Output: a = total area of the polygon
%         p = total perimeter of the polygon
r = s/(2*sin(pi/n)); % "radius" of the polygon
a = area(r,n);
p = perimeter(r,n);
% ===== subfunction "area"
function a = area(r,n)
% area Compute area of an n-sided polygon of radius r
a = n*r^2*sin(2*pi/n)/2;
% ===== subfunction "perimeter"
function p = perimeter(r,n)
% perimeter Compute perimeter of an n-sided polygon of
% radius r
p = n*2*r*sin(pi/n);
```

Le funzioni **area** e **perimeter**
non possono essere
richiamate dalla command
window



Loading e saving

Load e Save in binario e ASCII

ASCII più facile da condividere con altre applicazioni

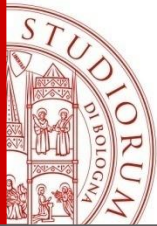
Sia y una matrice di valori

save 'c:\temp\my_matrix.txt' y **-ascii -double**

Salva y in un file *my_matrix.txt* in formato ascii

load 'c:\temp\my_matrix.txt'

Carica righe e colonne di valori nella variabile my_matrix che diventa una variabile



Esempio

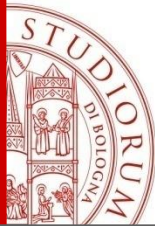
```
» x = [0:10]';
```

```
» y = [x x.^2 x.^3 x.^4]
```

y =

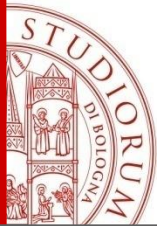
0	0	0	0
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561
10	100	1000	10000

```
» save 'c:\temp\y1.txt' y -ascii
```



Contenuto di y1.txt

0.0000000e+000	0.0000000e+000	0.0000000e+000	0.0000000e+000
1.0000000e+000	1.0000000e+000	1.0000000e+000	1.0000000e+000
2.0000000e+000	4.0000000e+000	8.0000000e+000	1.6000000e+001
3.0000000e+000	9.0000000e+000	2.7000000e+001	8.1000000e+001
4.0000000e+000	1.6000000e+001	6.4000000e+001	2.5600000e+002
5.0000000e+000	2.5000000e+001	1.2500000e+002	6.2500000e+002
6.0000000e+000	3.6000000e+001	2.1600000e+002	1.2960000e+003
7.0000000e+000	4.9000000e+001	3.4300000e+002	2.4010000e+003
8.0000000e+000	6.4000000e+001	5.1200000e+002	4.0960000e+003
9.0000000e+000	8.1000000e+001	7.2900000e+002	6.5610000e+003
1.0000000e+001	1.0000000e+002	1.0000000e+003	1.0000000e+004



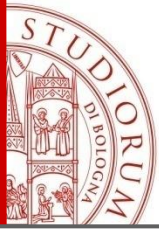
Input/output

Il comando **input** interrompe l'esecuzione e richiede un valore da tastiera come input. Una volta inserito il valore, l'm-file continua la sua esecuzione.

Il comando **echo** permette di visualizzare a video i comandi durante la loro esecuzione.

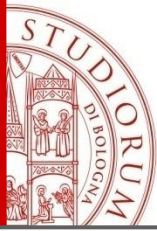
Il comando **pause** interrompe l'esecuzione fino a quando non si digita un tasto, mentre il comando **pause(n)** mette in pausa l'esecuzione per n secondi.

Il comando **keyboard** consente di inserire altri comandi da tastiera durante l'esecuzione di un m-file. Si presenta con **K>>** e attende l'inserimento del comando. Per riprendere la normale esecuzione dell'm-file basterà scrivere **return**.



Flussi di controllo

- **if elseif else end**
- **switch case otherwise end**
- **for end**
- **while end**
- **break**



For *counter = start:step:final* **end**

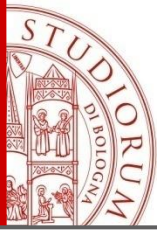
Il ciclo for permette di eseguire uno o più comandi un dato numero di volte.

Es. si vuole riempire una matrice di dimensione $m \times n$.

```
for i=1:m
    for j=1:n
        A(i,j)=1/(i+j);
    end
end
```

Ricordiamo di mettere un **;** alla fine di ogni istruzione per evitare la visualizzazione dell'output su schermo.

Ogni ciclo **for** ovviamente deve terminare con il proprio **end**.



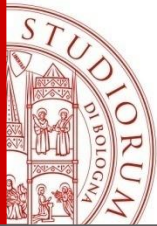
While (*condizione di entrata*)...end

Il ciclo while ripete l'esecuzione finchè la condizione logica risulta vera.

Esempio: Calcolare il più grande intero n per il quale $n!$ (fattoriale) è un numero ≤ 100 .

Possiamo utilizzare un ciclo while che calcola $n!$ e terminare l'esecuzione quando il valore calcolato è maggiore di 100.

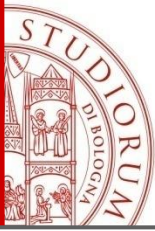
```
n=1;  
prod=1;  
while (prod<=100)  
    n=n+1; prod=prod*n;  
end  
n-1
```



Esempio

Calcolare la somma di alcuni numeri introdotti da tastiera fino ad ottenere un valore maggiore di 100. Il ciclo si interrompe se si inserisce come numero 0.

```
somma = 0;  
n=1;  
while (n~= 0 ) & (somma<=100)  
    n = input('Inserire un numero');  
    somma = somma + n;  
end
```

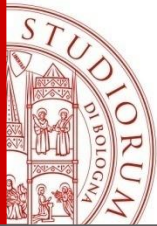


if ... elseif ... else ... end

L'istruzione condizionale **if** può essere accompagnata da **else** o **elseif**. L'istruzione **break** può essere inserita dentro al ciclo per terminare il ciclo.

```
if expression1
    statements1
elseif expression2
    statements2
else
    statements3
end
```

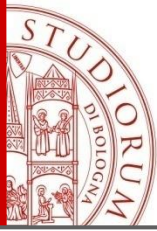
N.B. **else** e **elseif** possono essere omessi



Esempio

Si vuole verificare se il numero inserito sia pari o dispari.

```
n=input('Inserisci n ')\nif n <= 0, break,end\nif rem(n,2)==0 % funzione resto\n    disp(' n pari')\nelse\n    disp('n dispari')\nend
```



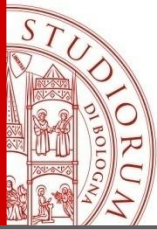
Switch... case...otherwise...end

Il comando switch può essere usato nel caso di if multipli.

```
switch(relation expression)
  case 1(value)
    block of statements
  case 2(value)
    block of statements
  case 3 (value)
    block of statements
  otherwise
  :
end
```

```
if rem(n,2) < 0
    disp ( 'il numero è negativo e dispari' )
elseif rem(n,2) > 0
    disp( 'il numero è positivo e dispari' )
else
    disp( 'il numero è pari' )
end
```

```
switch rem(n,2)
    case -1
        disp ( 'il numero è negativo e dispari' )
    case 1
        disp( 'il numero è positivo e dispari' )
    otherwise
        disp( 'il numero è pari' )
end
```



Puntatore a funzione

`funhandle = @function_name`

Ritorna un puntatore alla funzione di nome `function_name`.

- creiamo la funzione `funprova` nel file **funprova.m**

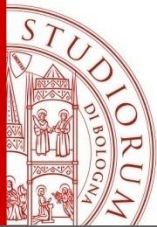
```
function y=funprova(x)  
y=cos(x).*sin(x);
```

- creiamo un puntatore alla funzione

```
f = @funprova;
```

- chiamiamo la funzione con i suoi argomenti

```
funprova(-pi:pi) oppure f(-pi:pi)
```

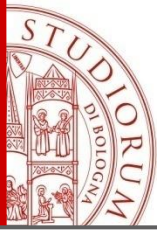


Funzioni di Funzioni

- Una funzione prende un'altra funzione come parametro in input.

Passiamo il puntatore a funzione in una chiamata ad un'altra funzione.

Esempio: scrivere una function che, dati in input una funzione e gli estremi di un intervallo, calcoli il minimo della funzione e il suo integrale sull'intervallo.



Esempio di una funzione MATLAB

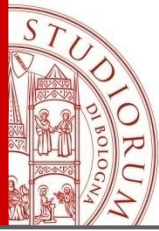
```
function [m,q]=calcolo(fp,vi,vf);
```

Dice a MATLAB che
questa è una funzione

Lista di argomenti di
output, separati da
virgola

Nome della
funzione

Lista degli argomenti
di input, separati da
virgola
fp è una variabile



Esempio (file calcolo.m)

```
function [m,q]=calcolo(fp,vi,vf)
% PLOT della funzione fp in [vi,vf], calcolo del minimo
% e dell'area della funzione in [vi,0].

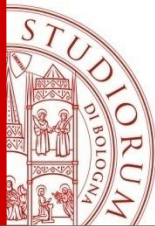
% FPLOTT grafico della funzione puntata da fp nel dominio di x [vi,vf].
fplot(fp,[vi,vf]);

% FMINBND trova il minimo di fp nel dominio (0,vf).
m = fminbnd(fp,0,vf,optimset('Display','off'));
hold on;
plot(m,fp(m),'ro'); % valutazione di funzione dentro al plot

% QUAD trova l'integrale definito di fp nel dominio [vi,vi/2].
q = quad(fp,vi,vi/2);
title(['Area = ',num2str(q)]);
```

Eseguire la funzione da command window

```
>>[minimo,area]=calcolo(@funprova,-pi,pi)
```



Esempio di uno script (randscript.m)

```
% script randscript
% Un semplice script per generare un vettore di n numeri random,
% calcolarne la somma e stampare sullo schermo il tempo impiegato

n = 100000;          % il numero dei random che si vogliono generare
y = zeros(n,1);      % allocazione della memoria per y
fprintf('Simulazione di %d numeri random.....\n\n',n);

% utilizziamo il ciclo for per creare i numeri random
fprintf('Ciclo for ..... \n');
t=cputime;           % inizializzo il timer

for i=1:n
    y(i) = rand(1);
end

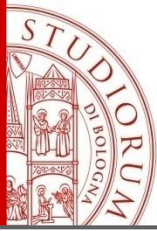
total = sum(y);
fprintf('Somma di %d numeri random = %f\n',n,total);
% tempo utilizzato da MATLAB per svolgere il calcolo (in secondi)
e=cputime-t;

fprintf('Tempo impiegato usando il ciclo for = %6.5f
        microseconds\n\n', (e)*1000);
```

```
for i=1:n
    y(i) = rand(1);
end
```

oppure

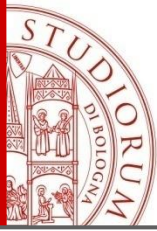
```
y = rand(n,1);
```



Esempio di una funzione MATLAB

```
function [meanr, stdr] = simulate(n);  
% Una semplice funzione che restituisce la media e la deviazione  
% standard di n numeri random uniformemente distribuiti.  
% INPUTS:  
%     n: numero dei random generati (intero positivo)  
% OUTPUTS:  
%     meanr: media degli n numeri random  
%     stdr: deviazione standard degli n numeri random  
% test per verificare se n è un intero positivo  
if (rem(n,1)~=0) | n<=0  
    error('Input n must be a positive integer');  
end  
fprintf('Simulazione di %d numeri random.....\n\n',n);  
z = rand(n,1);                % genero n numeri random  
meanr= mean(z);                % calcolo la media  
fprintf('Media dei %d numeri random = %f\n',n,meanr);  
stdr= std(z);                  % calcolo la deviazione standard  
fprintf('Deviazione standard dei %d numeri random = %f\n',n,stdr);
```

Non serve un'istruzione esplicita di return. Ogni valore non restituito è conosciuto solo localmente all'interno della funzione



Richiamare per eseguire una funzione

» **[m, s] = simulate(100);**

Simulazione di 100 numeri random.....

Media dei 100 numeri random = 0.499702

Deviazione standard dei 100 numeri random = 0.499702

» **[m, s] = simulate(1000000);**

Simulazione di 1000000 numeri random.....

Media dei 1000000 numeri random = 0.499684

Deviazione standard dei 1000000 numeri random = 0.288456

» **m**

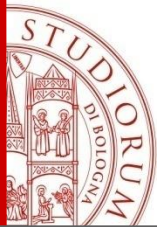
m =

0.4997

» **s**

s =

0.2885



Grafici

plot

plot x-y lineare

loglog

plot log-log x-y

semilogx

semi-log x-y plot(logaritmico in x)

semilogy

semi-log x-y plot (logaritmico in y)

polar

plot in coords polari

mesh

mesh di superficie 3D

contour

plot a linee di livello

bar

plot a barre

stairs

plot a gradini

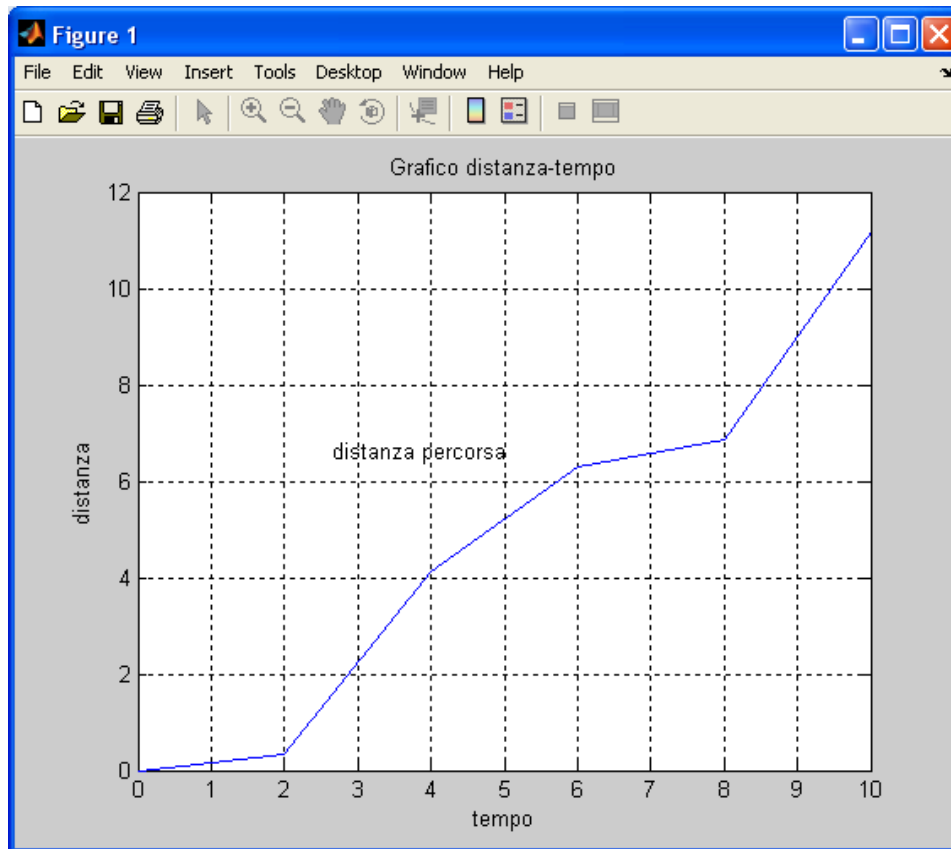
Prodotto il plot, i seguenti comandi possono essere utili per arricchirlo:

title	titolo del plot
xlabel	etichetta associata all'asse x
ylabel	etichetta associata all'asse y
text	testo posizionato in modo arbitrario
gtext	testo posizionato mediante mouse
grid	linee griglia

I seguenti comandi vengono invece utilizzati per gestire il grafico:

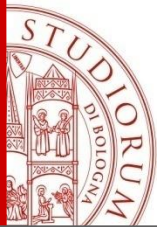
axis	scala degli assi
hold	mantiene il plot corrente sullo schermo
shg	mostra il grafico sullo schermo
clf	pulisce la figura corrente
subplot	suddivide un plot

Esempio: grafico x-y



- » `x= [0:2:10];`
- » `y=[0 0.33 4.13 6.29 6.85 11.19];`
- » `plot(x,y)`
- » `title('Grafico distanza-tempo')`
- » `xlabel('tempo')`
- » `ylabel('distanza')`
- » `grid`
- » `gtext('distanza percorsa')`

N.B. Ogni stringa testo
deve essere racchiusa tra
apostrofi



Più grafici

Per mantenere il grafico attivo nella finestra si dovrà utilizzare il comando **hold on**, altrimenti verrà perso quando si chiede di visualizzare il nuovo plot.

```
>> t=0:5/200:5;
```

```
>> y1=1-exp(-t);
```

```
>> plot(t,y1)
```

```
>> hold on
```

% mantiene il plot precedente

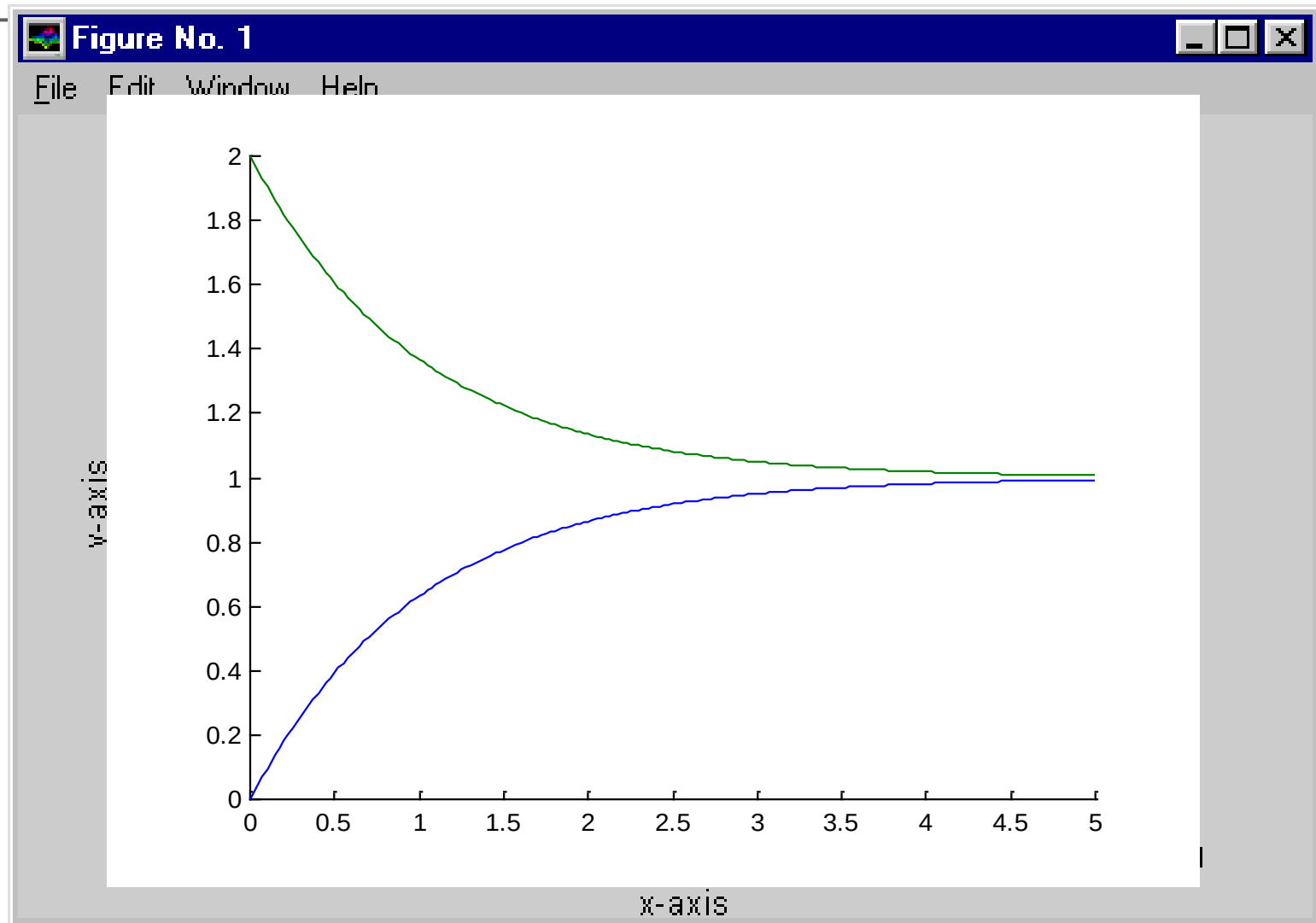
```
>> y2=1+exp(-t);
```

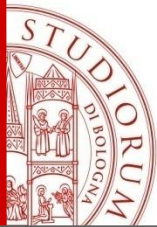
```
>> plot(t,y2)
```

```
>> hold off
```

% disattiva l'hold on

Più grafici

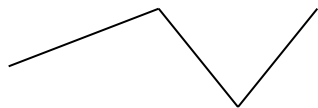




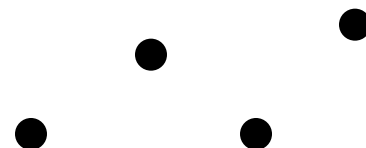
Attributi grafici

Automaticamente MATLAB unisce i punti con delle linee, se si vogliono visualizzare solo i punti si può specificare il tipo di punto desiderato.

>>plot(t,y)



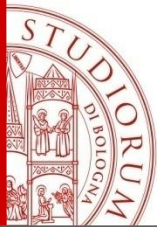
>>plot(t,y,'o')



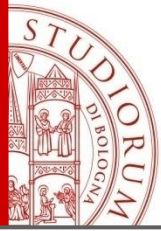
E' possibile caratterizzare un grafico scegliendo il tipo di linea, il colore, o a punti, aggiungendo un altro argomento al comando plot:

>>plot(t,y,'--r')

Produce un grafico a tratteggio di colore **rosso**.



TIPO LINEA		COLORI		TIPO PUNTI	
intera	-	blu	b	punto	.
tratteggio	--	nero	k	croce	+
a punti	:	verde	g	stella	*
tratto-punto	-.	rosso	r	cerchio	o
		giallo	y	segno x	x
		bianco	w	quadrato	s
		ciano	c	triangolo	^
		magenta	m	rombo	d
				pentagono	p



```
>>figure
```

% apre una nuova finestra figura

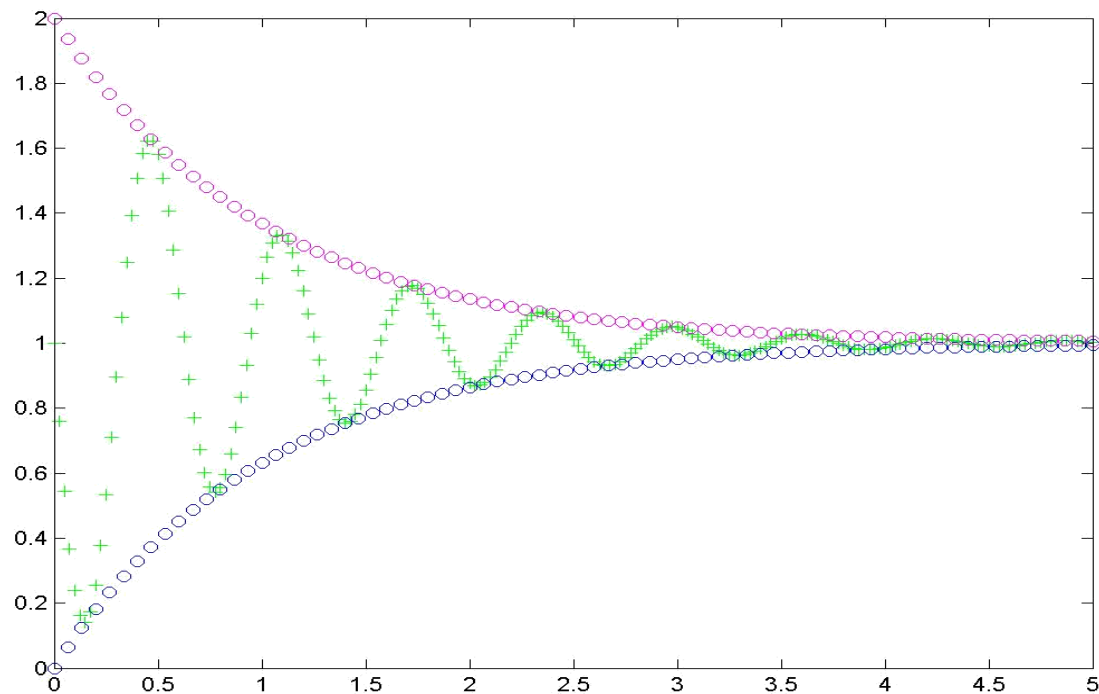
```
>>t=0:1/40:5; t1=0:1/15:5;
```

```
>>y=1-exp(-t).*sin(10*t);
```

```
>>y1=1-exp(-t1);
```

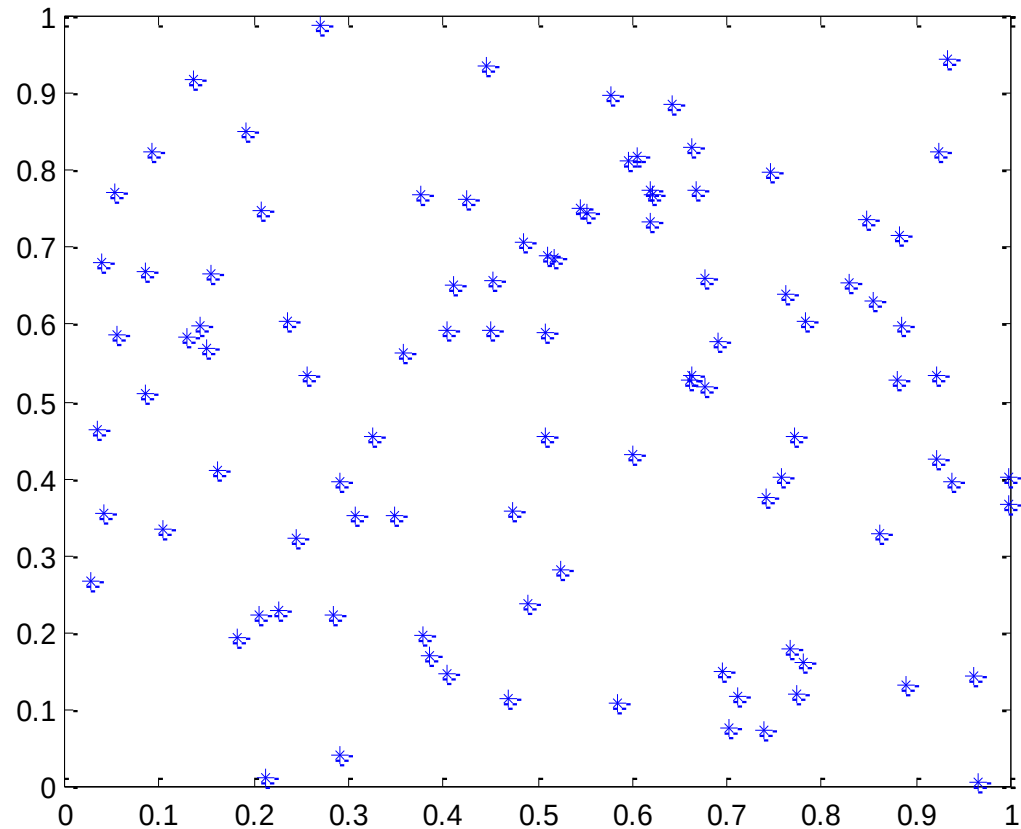
```
>>y2=1+exp(-t1);
```

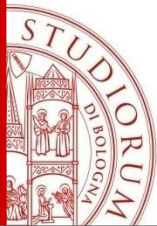
```
>>plot(t,y, 'g+', t1,y1, 'bo', t1,y2, 'mo')
```



Grafici

```
» x = rand(1,100);  
» y = rand(1,100);  
» plot(x,y,'*')
```





Gestione della finestra di una figura

MATLAB permette di gestire interattivamente la finestra della figura utilizzando i menu e le icone presenti. E' possibile:

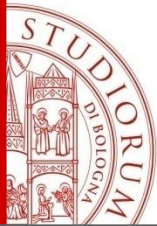
- inserire testi, legende, linee e frecce;
- zoom, ruotare, cambiare la posizione dell'osservatore;
- cambiare colore al grafico, allo sfondo, alla finestra;

Per gestire una figura da MATLAB è necessario salvarla in formato **.fig**:

File → **Save As** → **<figura1.fig>**

Per stamparla o includerla in altri documenti si può salvare in formato **.jpg, .eps, ...**:

File → **Export** → **<figura1.eps>**

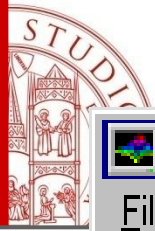


Gestione di più finestre grafiche

Se si desidera avere più finestre grafiche distinte, si utilizza prima del comando plot, il comando

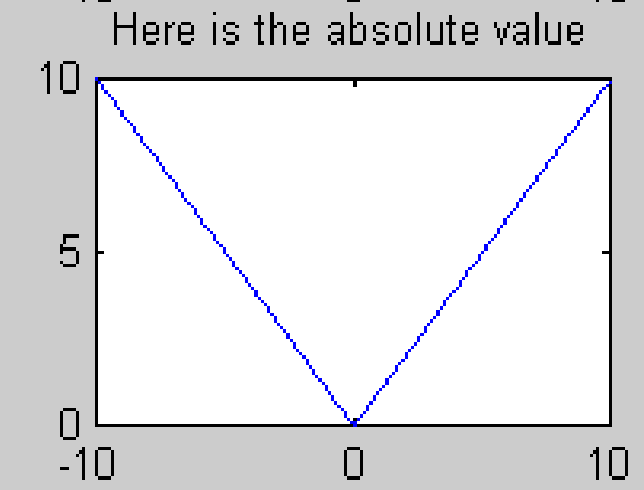
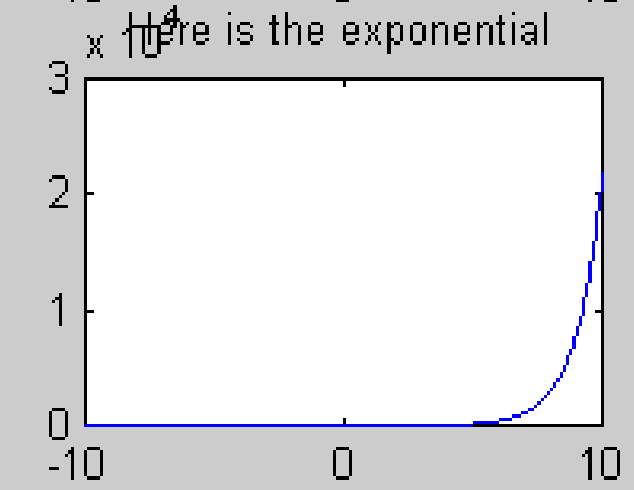
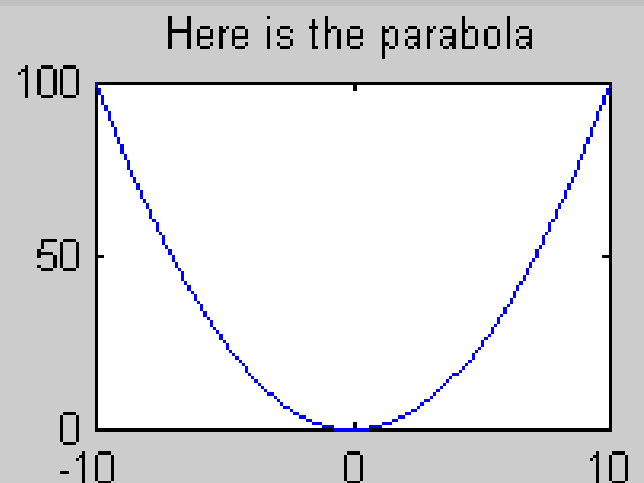
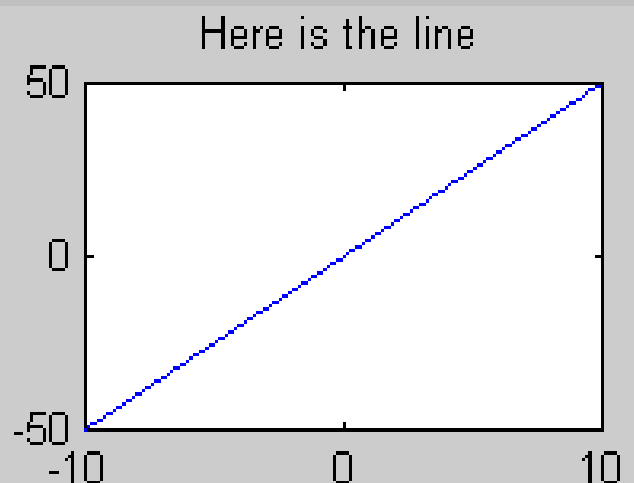
subplot(m,n,p)

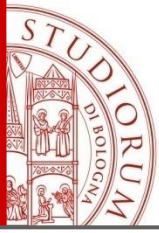
La finestra grafica si divide in una tabella m x n di finestre grafiche e la p-esima è selezionata dal plot corrente.



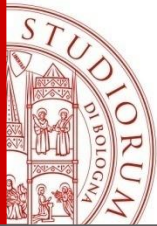
 **Figure No. 1**

File Edit Window Help





- » `t=-10:.01:10;`
- » `y1=t;`
- » `y2=t.^2;`
- » `y3=exp(t);`
- » `y4=abs(t);`
- » `subplot(2,2,1)`
- » `plot(t,y1), title('Here is the line')`
- » `subplot(2,2,2)`
- » `plot(t,y2), title('Here is the parabola')`
- » `subplot(2,2,3)`
- » `plot(t,y3), title('Here is the exponential')`
- » `subplot(2,2,4)`
- » `plot(t,y4), title('Here is the absolute value')`

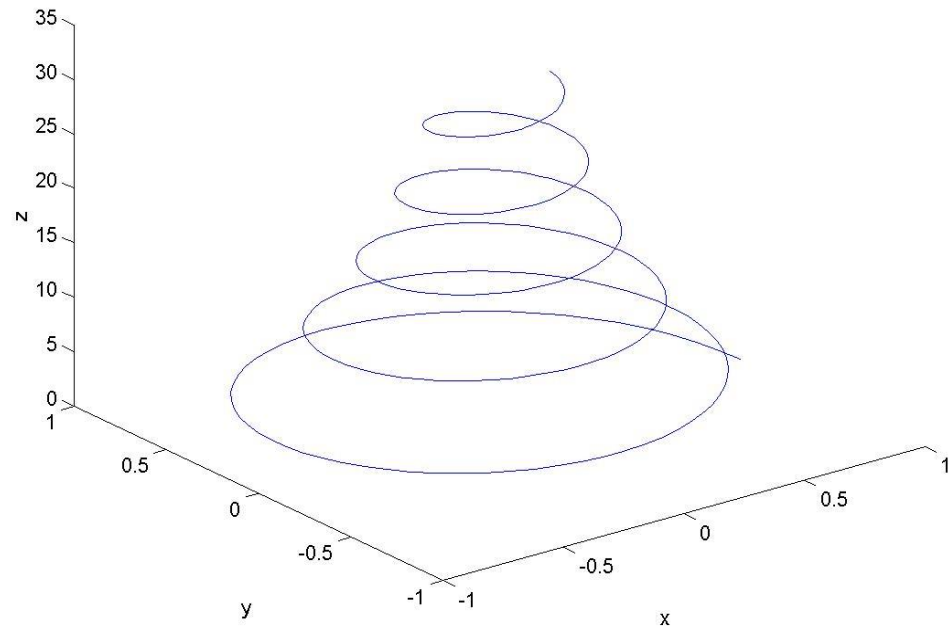


Grafica 3D

- **plot3** permette di disegnare una curva $(x(t), y(t), z(t))$, $t \in [a, b]$ nello spazio 3D
- **mesh** disegna superfici per punti visualizzando solamente le linee che connettono i punti (wireframe)
- **surf** disegna superfici per punti visualizzandole a faccette

```
>>t=0:0.1:10*pi;  
>>x=exp(-t/20).*cos(t);  
>>y=exp(-t/20).*sin(t);  
>>z=t;  
  
>>plot3(x,y,z);  
  
>>title('Esempio di plot 3D');  
>>xlabel('x');  
>>ylabel('y');  
>>zlabel('z');
```

Esempio di plot 3D



```
>>x=linspace(0,2*pi,50);
```

```
>>y=linspace(0,pi,50);
```

```
>>[X,Y]=meshgrid(x,y); ←
```

Prepara una griglia di punti sul piano xy sui quali verra' valutata la funzione bivariata.

```
>>Z=sin(X).*cos(Y);
```

```
>>mesh(X,Y,Z);
```

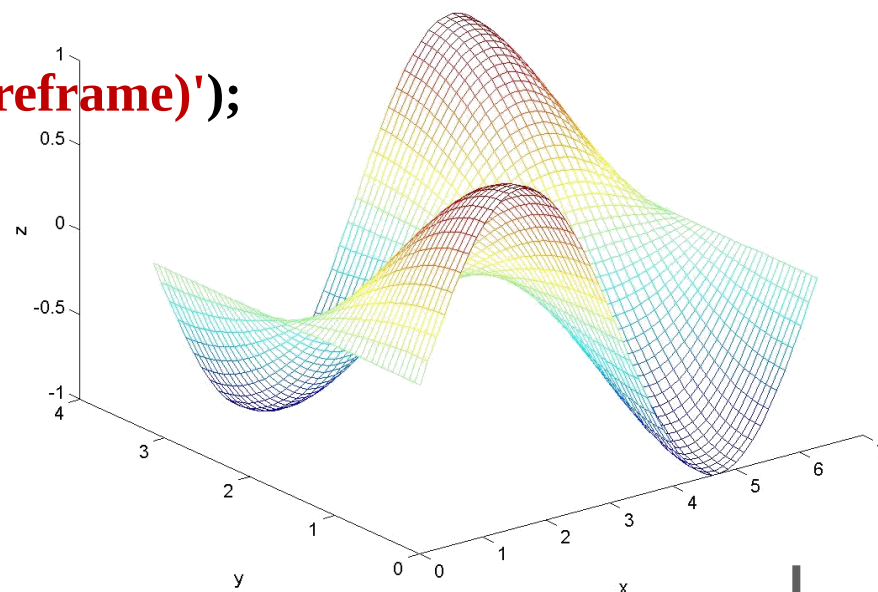
```
>>title('Esempio di mesh (grafica wireframe)');
```

```
>>xlabel('x');
```

```
>>ylabel('y');
```

```
>>zlabel('z');
```

Esempio di mesh (grafica wireframe)



```
>>[X,Y]=meshgrid(-3:.2:3,-2:.2:2);
```

```
>>Z=exp(-(X.^2+Y.^2)/3);
```

```
>>surf(X,Y,Z);
```

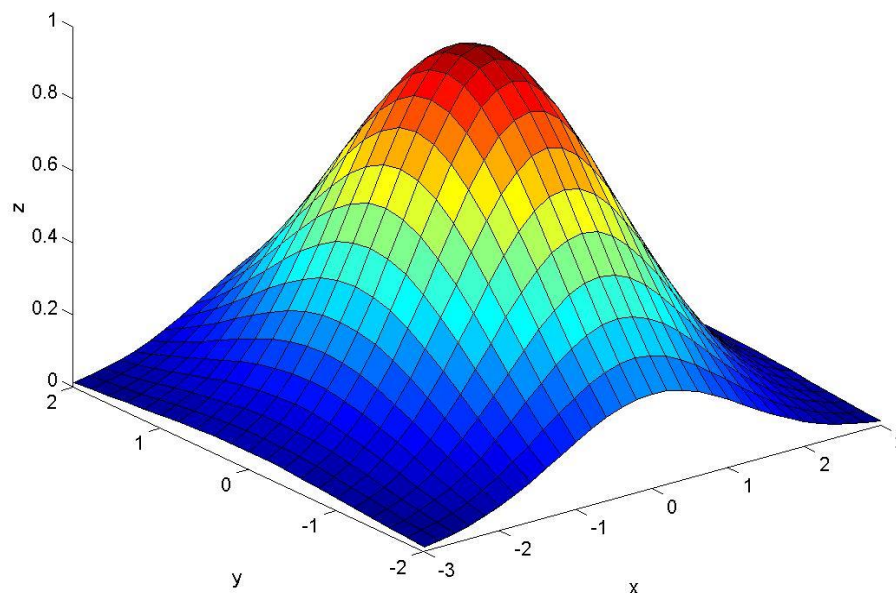
```
>>title('Esempio di surf (grafica a faccette)');
```

```
>>xlabel('x');
```

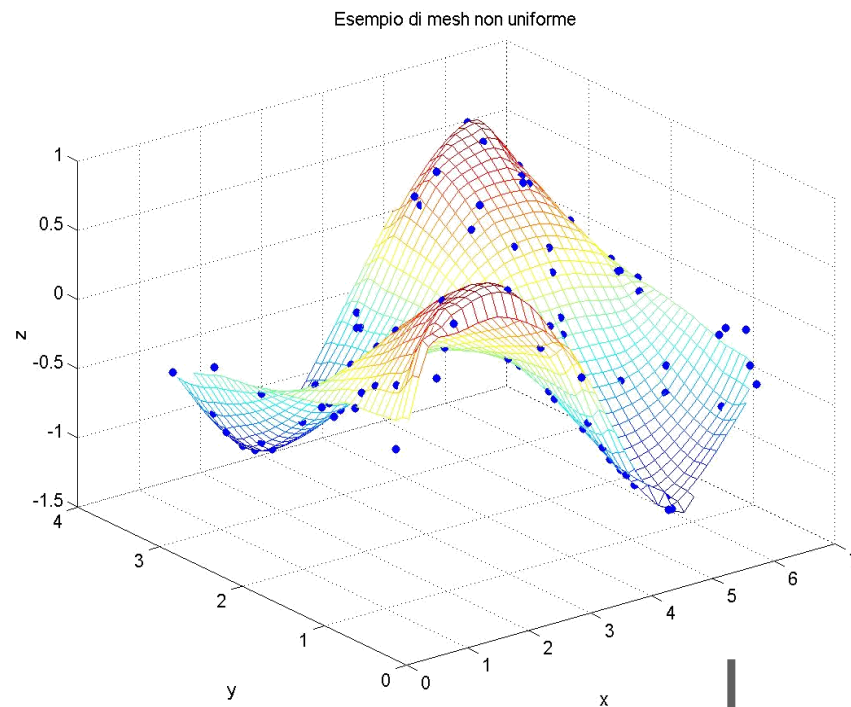
```
>>ylabel('y');
```

```
>>zlabel('z');
```

Esempio di surf (grafica a faccette)

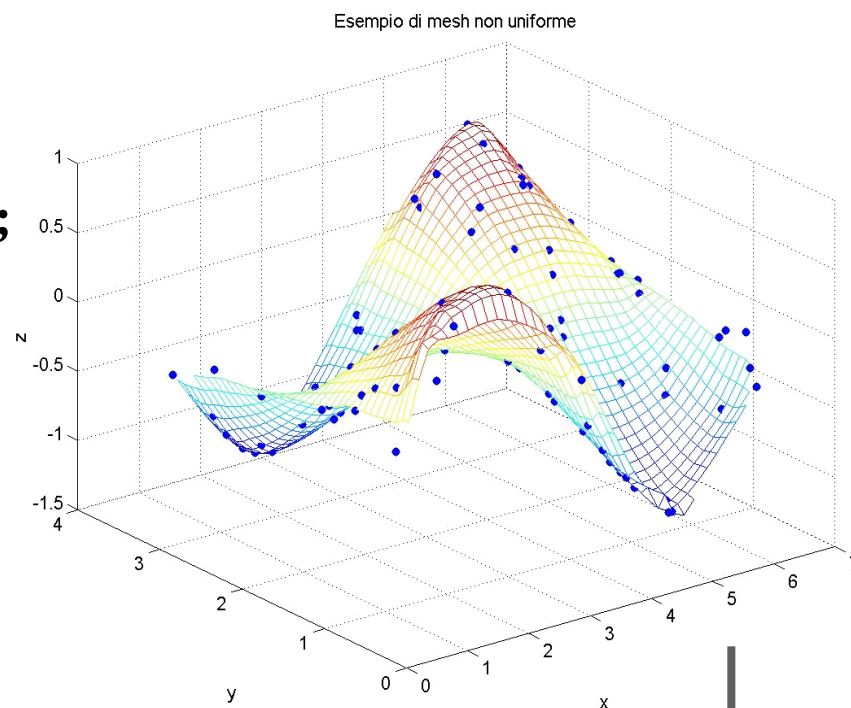


```
>>x=rand(100,1)*2*pi;  
>>y=rand(100,1)*pi;  
>>z=sin(x).*cos(y);  
>>xlin=linspace(min(x),max(x),40);  
>>ylin=linspace(min(y),max(y),40);  
>>[X,Y]=meshgrid(xlin,ylin);  
>>Z=griddata(x,y,z,X,Y,'cubic');  
>>mesh(X,Y,Z);  
>>title('Esempio di mesh non uniforme');  
>>hold on;  
>>plot3(x,y,z,'.','MarkerSize',15);  
>>xlabel('x');  
>>ylabel('y');  
>>zlabel('z');  
>>grid on;
```

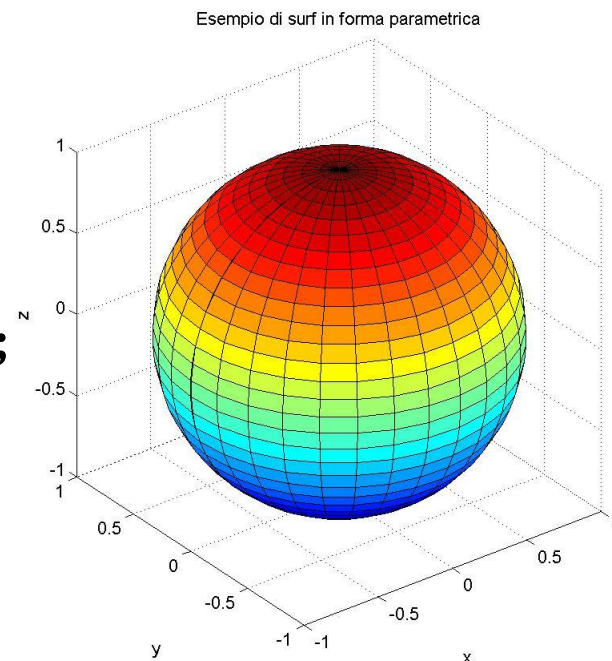



```
>>x=rand(100,1)*2*pi;
>>y=rand(100,1)*pi;
>>z=sin(x).*cos(y);
>>xlin=linspace(min(x),max(x),40);
>>ylin=linspace(min(y),max(y),40);
>>[X,Y]=meshgrid(xlin,ylin);
>>Z=griddata(x,y,z,X,Y,'cubic');
>>mesh(X,Y,Z);
>>title('Esempio di mesh non uniforme');
>>hold on;
>>plot3(x,y,z,'.','MarkerSize',15);
>>xlabel('x');
>>ylabel('y');
>>zlabel('z');
>>grid on;
```

Interpolazione di dati scattered (x,y,z),
Visualizzazione funzione interpolante
valutata nella griglia XY

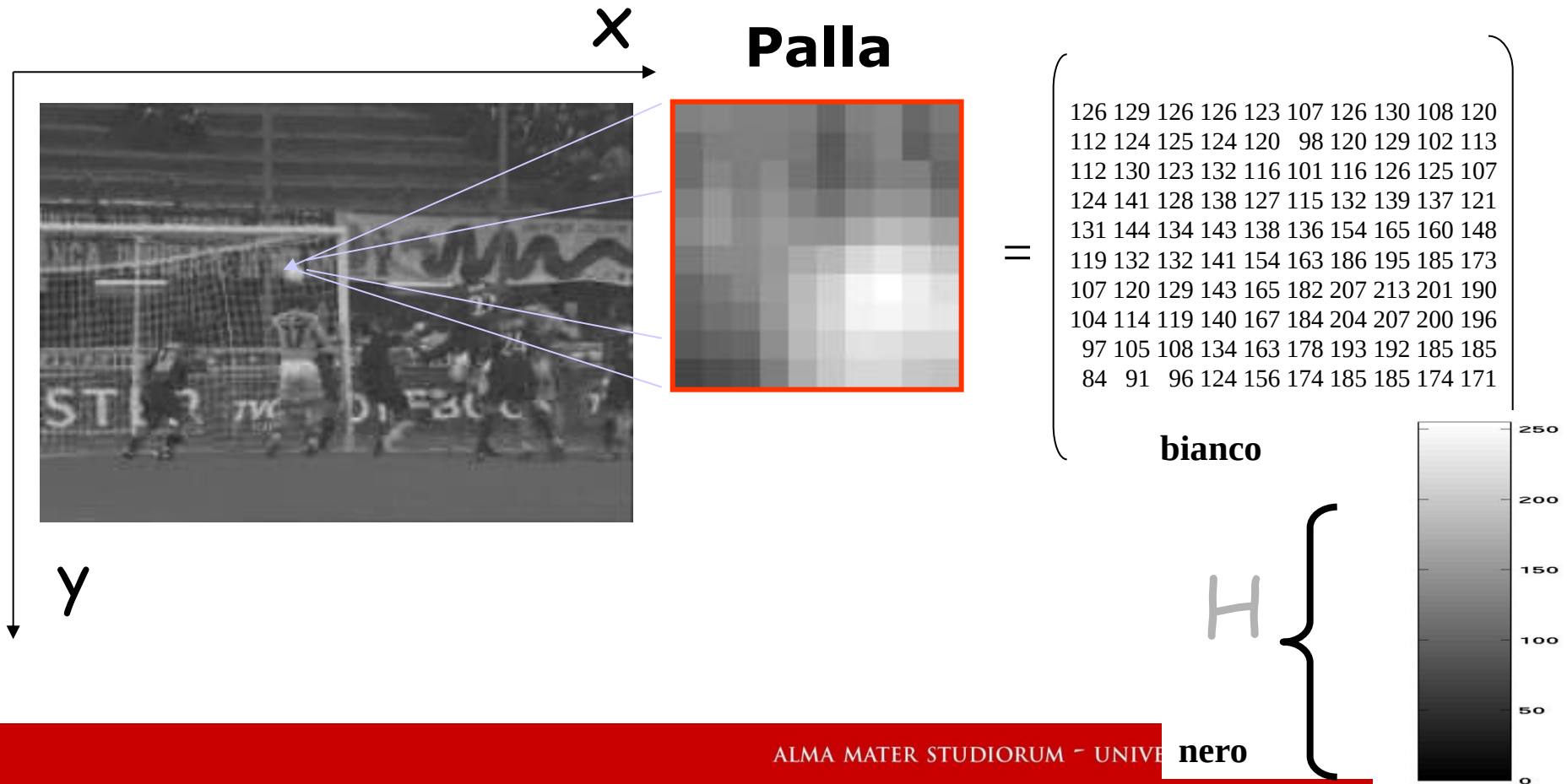


```
>>n=31;  
>>theta=pi*(-n:2:n)/n;  
>>phi=(pi/2)*(-n:2:n)'/n;  
  
>>X=cos(phi)*cos(theta);  
>>Y=cos(phi)*sin(theta);  
>>Z=sin(phi)*ones(size(theta));  
  
>>surf(X,Y,Z);  
  
>>title('Esempio di surf in forma parametrica');  
>>xlabel('x');  
>>ylabel('y');  
>>zlabel('z');  
>>grid on;  
>>axis square;
```

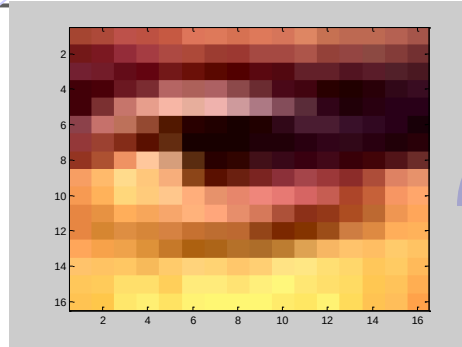
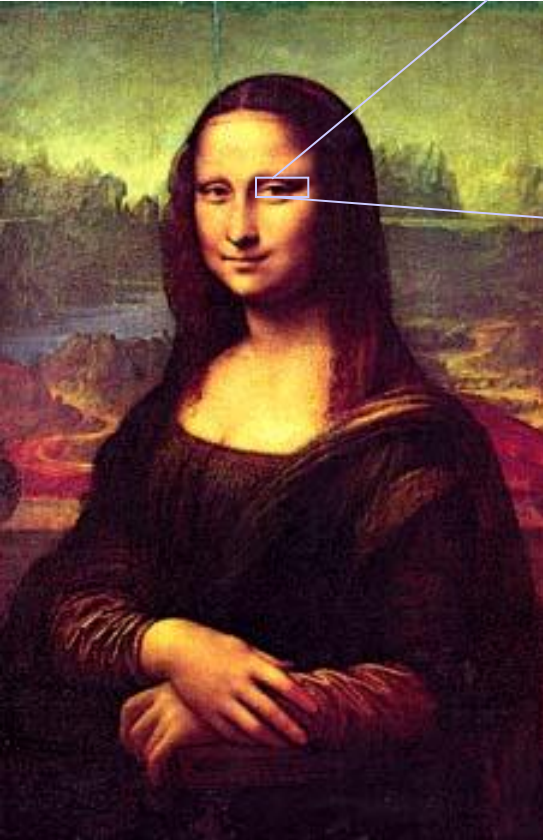


Definizione di immagine digitale

- L'intensità dell'immagine è trattata come funzione dello spazio
- $I_s: (x,y) \in D \longrightarrow I_s(x,y) \in [0, H-1]$

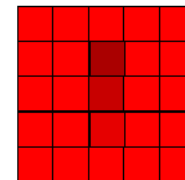


Immagini digitali a colori



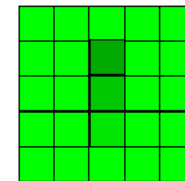
Giorgio Sedmak 2001

Immagini a colori RGB



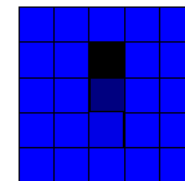
0.7
0.8
0.9
1.0

Componente R
(rosso)



0.7
0.8
0.9
1.0

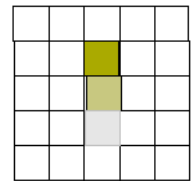
Componente G
(verde)



0.0
0.5
0.9
1.0

Componente B
(blu)

Immagine RGB composta
da (R + G + B)



Immagini in MATLAB

Carica immagine

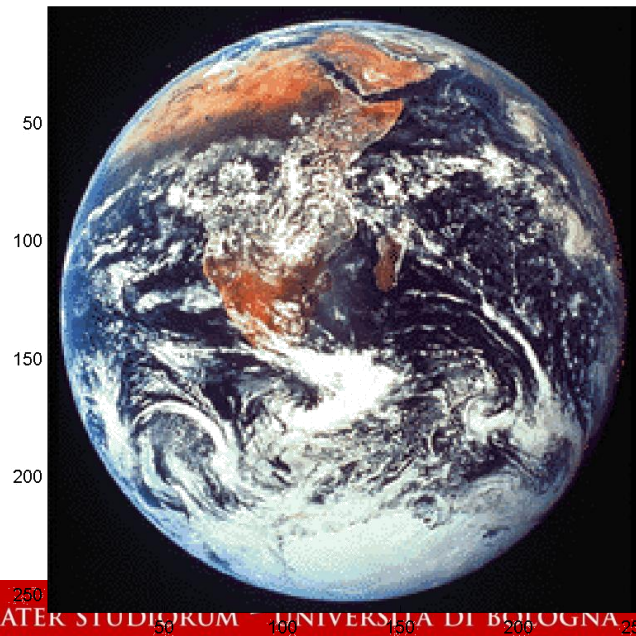
- » `clf`
- » `A = imread(filename, fmt)`
- » `image(A)`
- » `colormap(map)`
- » `axis image`

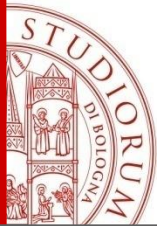
visualizza
un'immagine

associa una
colormap all'immagine

rende la proporzione 1:1
dell'immagine

Legge un'immagine in formato **fmt** a livelli di grigio o a colori dal file specificato dalla stringa **filename**. Se il file non è nella directory corrente, specificare l'intero percorso.





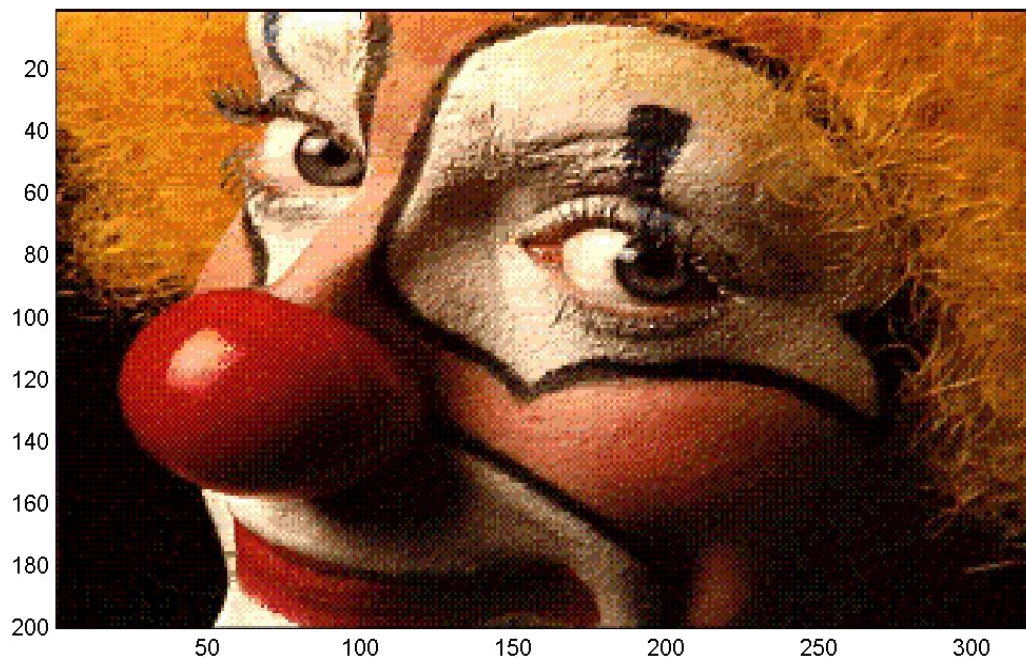
Tipi di immagini

- **Indicizzata** array $m \times n$ di interi tra $[1,p]$ che rappresentano gli indici della **colormap**
- **Intensità** array $m \times n$ di floating point, tra $[0,1]$ tipicamente, che rappresentano le intensità della colormap
- **True color** array $m \times n \times 3$ di floating point tra $[0,1]$ che rappresentano le intensità delle tre componenti dei colori **rgb** (solo con hardware true color)

Immagini in Matlab

```
» clf  
» i = imread('c:\immagini\clown.jpg');  
» image(i)  
» whos i
```

Name	Size	Class
i	200x320x3	uint8



Il miglior modo per
imparare MATLAB
è usare MATLAB