Introduzione a MATLAB

Luca Zanni, Marco Prato

Calcolo Numerico
Corsi di Laurea in Matematica e Informatica

MATrix LABoratory

MATLAB è nato principalmente come programma destinato alla gestione di matrici. E' un interprete di comandi in cui l'unità base dei dati è un vettore o una matrice

I comandi possono essere forniti interattivamente o contenuti in files su disco (m-files)

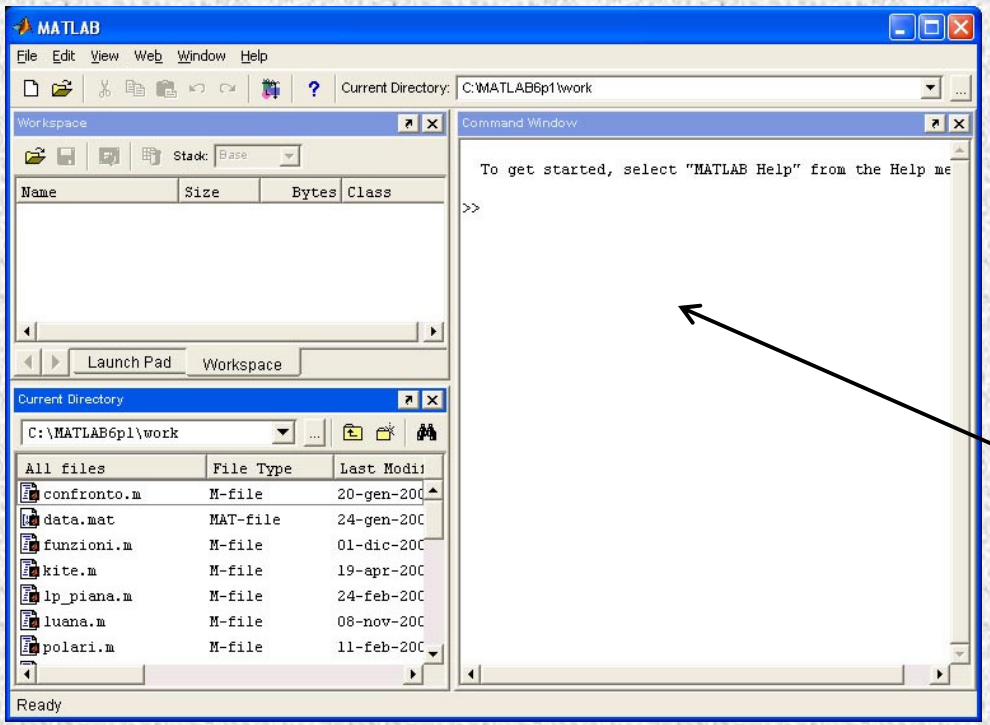
Comprende un vasto set di funzioni predefinite e numerose librerie (toolbox) per svariate applicazioni che possono essere ampliate

Ha una buona potenzialità grafica

Esistono versioni di MATLAB per Unix/Linux, Windows e MAC. I files creati sono portabili da una piattaforma all'altra

Potenzialità di MATLAB

Calcolatrice



digitare 3 + 2

premere invio

risposta immediata!

Potenzialità di MATLAB

Calcolo matriciale

```
>> A = magic(3)
>> B = [4 5 0; 3 8 3; 4 9 1]
>> A*B

ans =

     59   102     9
     55   118    22
     51   110    29
```

Potenzialità di MATLAB

Soluzione di equazioni

```
>> s = solve('cos(2*x)+sin(x)=1')

s =

0

pi

1/6*pi

5/6*pi
```

Potenzialità di MATLAB

Derivate di funzioni

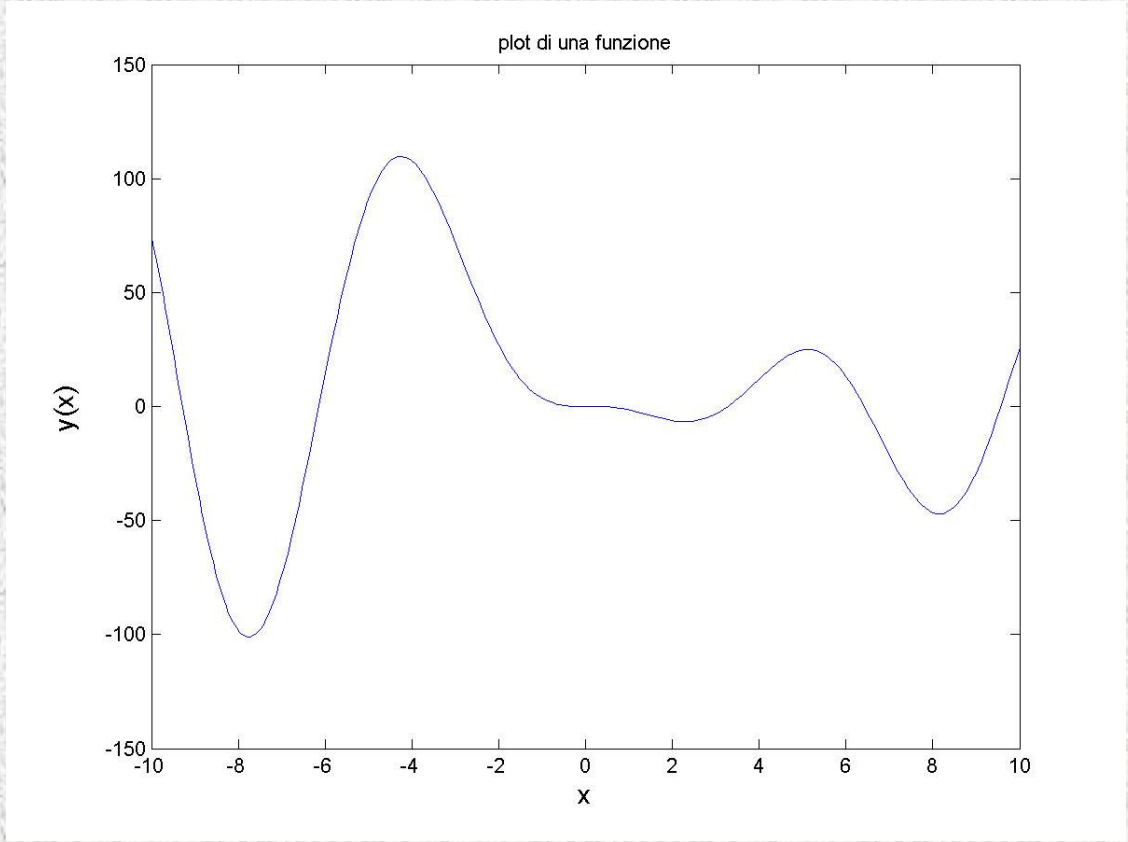
```
>> syms x        % calcolo simbolico
>> f= 2*(sin(x+3)/(x+3))*(4*x^2);
>> diff(f)

ans =
8*cos(x+3)/(x+3)*x^2-8*sin(x+3)/(x+3)^2*x^2+16*sin(x+3)/(x+3)*x
```

```
>> pretty(ans)
                          2                   2
                 cos(x + 3) x        sin(x + 3) x        sin(x + 3) x
               8 ------------- - 8 ------------- + 16 -------------
                     x + 3                   2              x + 3
                                      (x + 3)
```

Potenzialità di MATLAB

Grafici 2D di funzioni

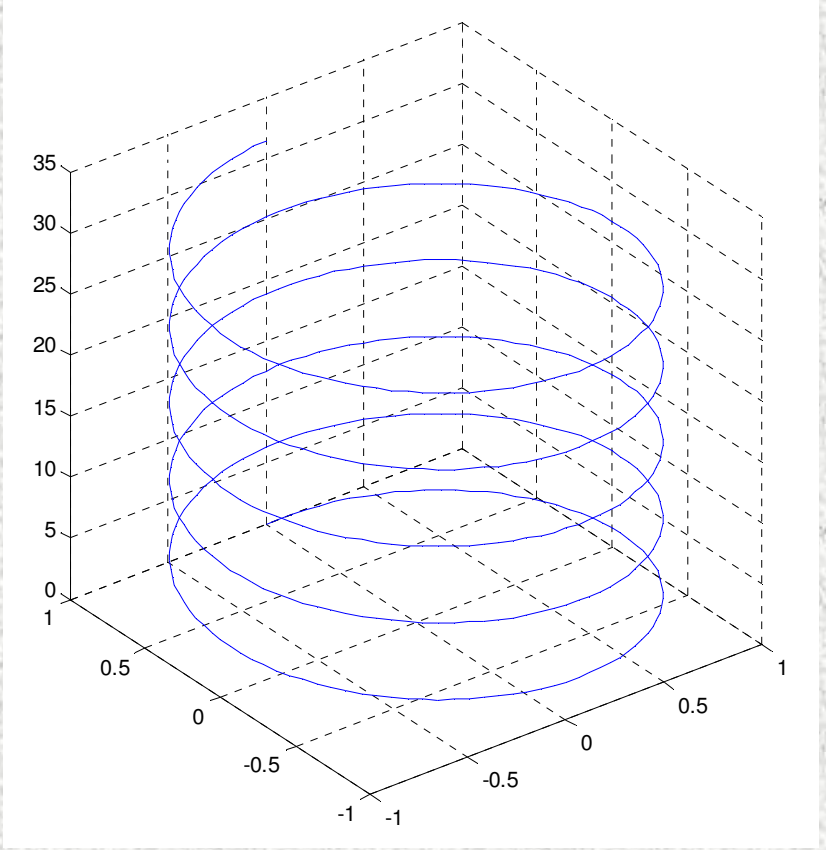
```
>> fplot(inline('2*(sin(x+3)/(x+3))*(4*x^2)'),[-10 10])
```



Potenzialità di MATLAB

Grafici 3D di funzioni

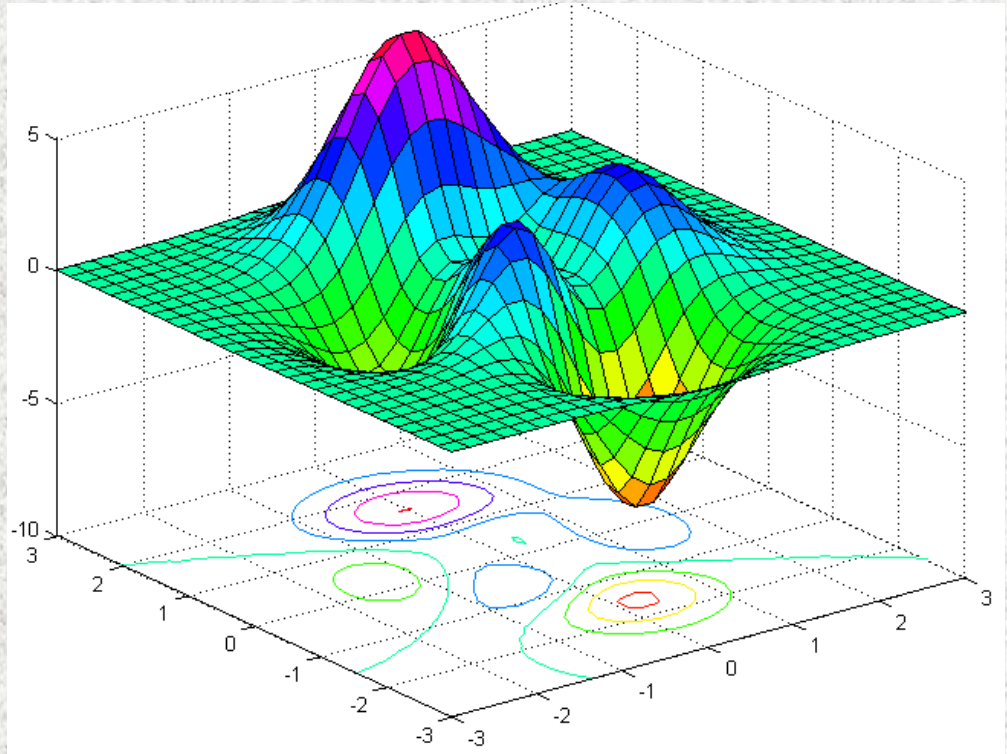
```
>> t = 0:pi/50:10*pi
>> plot3(sin(t),cos(t),t)
>> grid on
>> axis square
```



Potenzialità di MATLAB

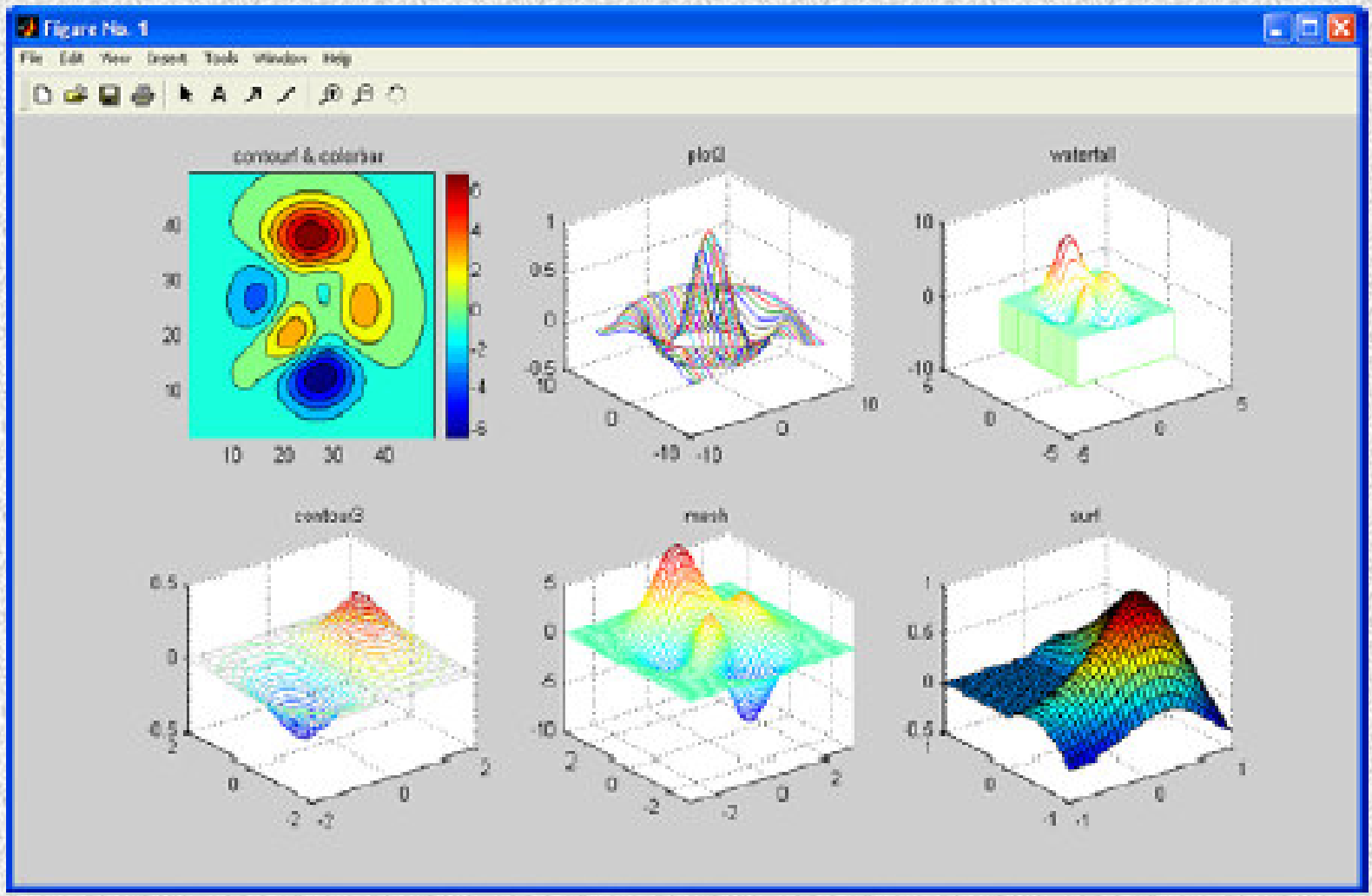
Grafici 3D di funzioni

```
>> [X,Y,Z]= peaks(30)
>> surfc(X,Y,Z)
>> colormap hsv
>> axis([-3 3 -3 3 -10 5])
>> grid on
```



Potenzialità di MATLAB

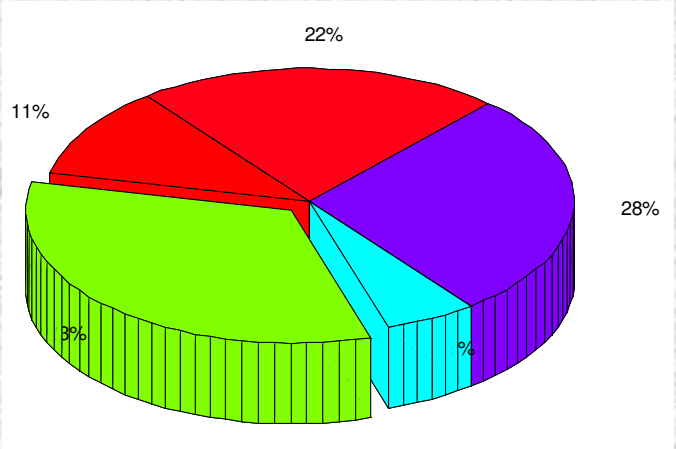
Altri esempi di grafici 3D

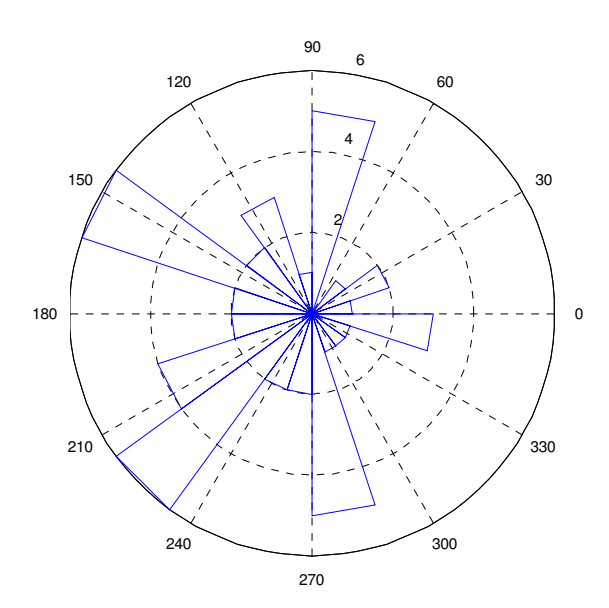


Potenzialità di MATLAB

Altri esempi di grafici 3D

```
>> x = [1 3 0.5 2.5 2]
>> explode = [0 1 0 0 0]
>> pie3(x,explode)
>> colormap hsv
```



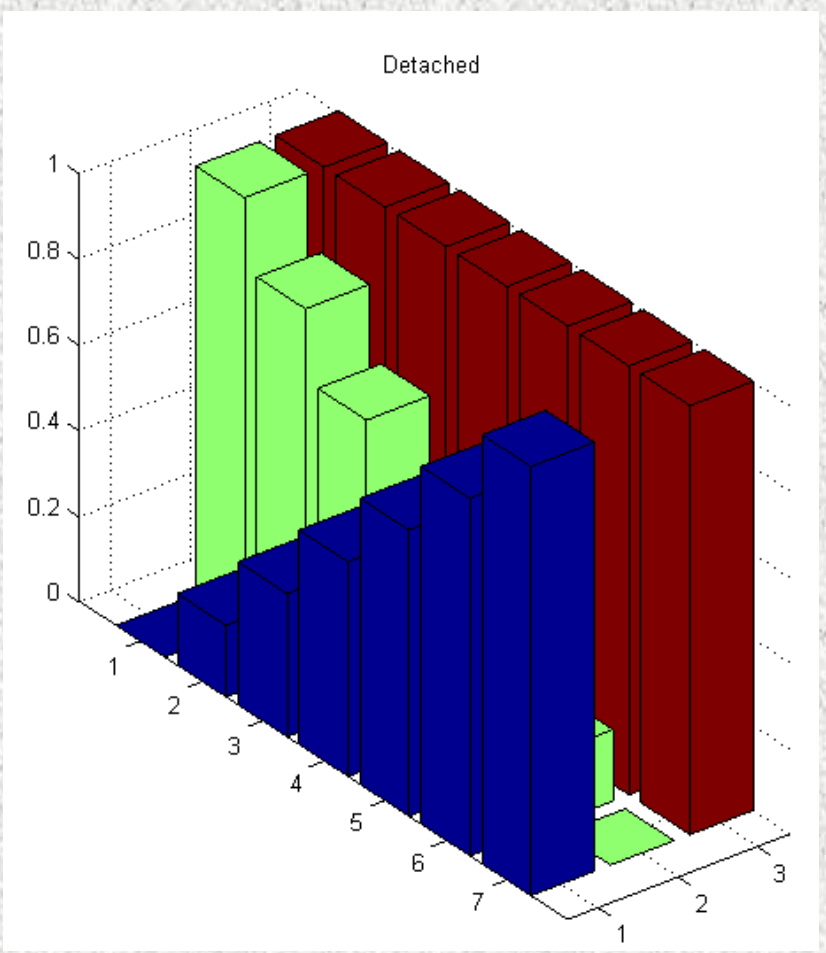



```
>> theta = 2*pi*rand(1,50);
>> rose(theta)
```

Potenzialità di MATLAB

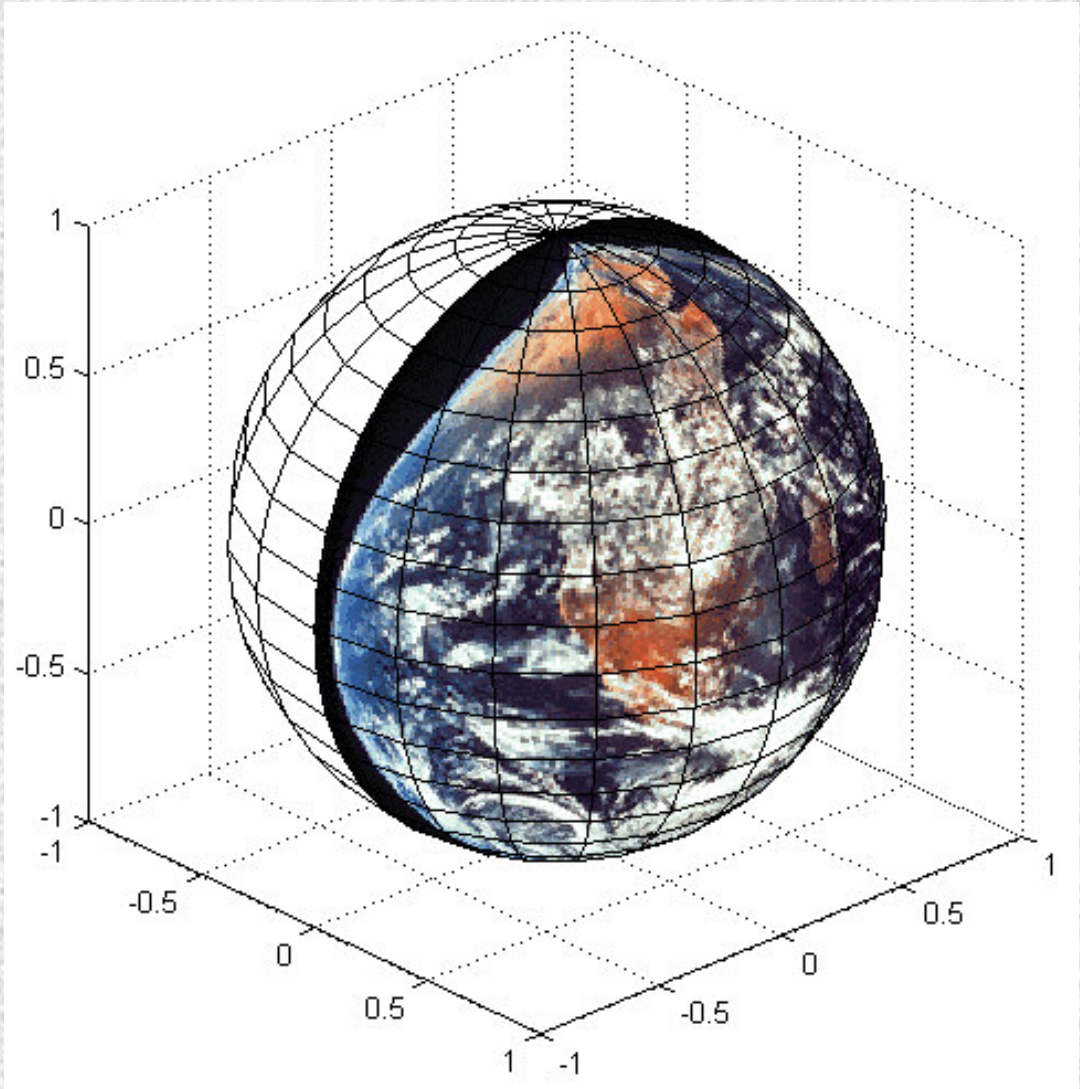
Altri esempi di grafici 3D

```
>> Y = cool(7);
>> bar3(Y,'detached')
>> title('Detached')
```



Potenzialità di MATLAB

Altri esempi di grafici 3D



Avvio del programma

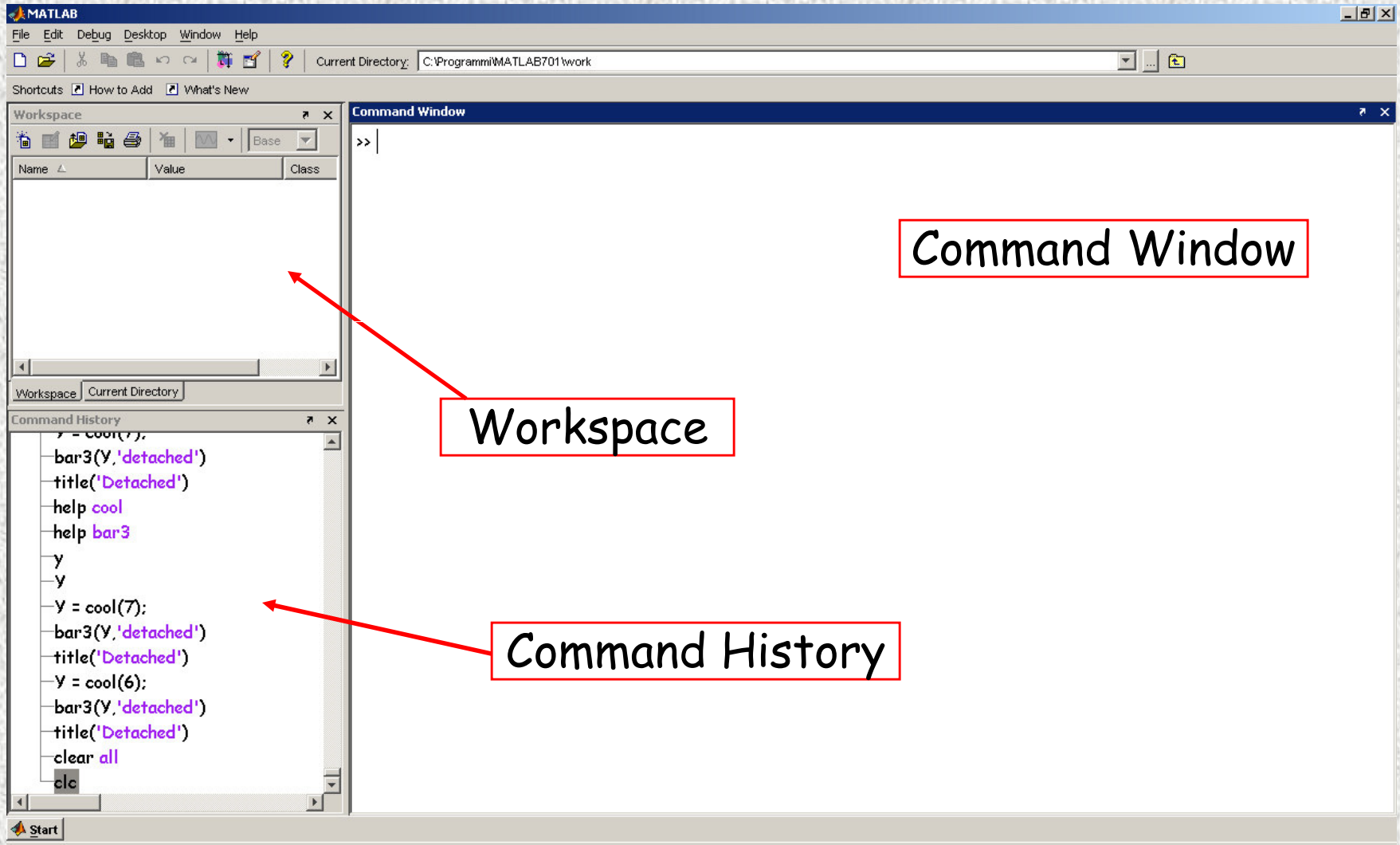
Avvio di MATLAB (Windows)

Start → Programmi → Matlab (o icona sul desktop)

Avvio di MATLAB (Linux)

Terminale `matlab` INVIO

Schermata MATLAB



La linea di comando

La linea di comando è indicata da un prompt come in DOS: >>

Accetta dichiarazioni di variabili, espressioni e chiamate a tutte le funzioni disponibili nel programma. Tutte le funzioni di MATLAB non sono altro che file di testo, simili a quelli che l'utente può generare con un text editor, e vengono eseguite semplicemente digitandone il nome sulla linea di comando

MATLAB permette inoltre di richiamare le ultime righe di comandi inseriti usando le frecce in alto e in basso

Help in linea

Esempio: >> help cos

Comandi di uso generale

who: elenco delle variabili definite in memoria

whos: informazioni su tutte le variabili in memoria

clear: cancella tutte le variabili in memoria o una in particolare se specificata (clear nome_variabile)

save: salva tutte le variabili in memoria sul file specificato, in vari formati

load: richiama in memoria le variabili salvate sul file specificato

what: elenco di tutte le funzioni MATLAB nell'area di lavoro (estensione .m) e dei file di dati che sono stati salvati (estensione .mat)

Assegnazione di variabili costanti

```
>> a = 1.54
```

"a" è il nome della costante, 1.54 il valore

```
>> a = 1.54;
```

";" non visualizza la risposta sullo schermo

```
>> 5
ans = 5
```

"ans" è il nome della variabile di default

Di default MATLAB lavora in doppia precisione. Ogni numero memorizzato in doppia precisione occupa 8 bytes

Operazioni aritmetiche

+	addizione
-	sottrazione
/	divisione
*	moltiplicazione
^	potenza

$$x = \frac{3 + 5^3 - 2/3}{4(5 + 2^4)}$$

ATTENZIONE: l'intero calcolo va scritto in riga. E' necessario un uso adeguato delle parentesi () per le precedenze aritmetiche

```
>> x = (3 + 5^3 - 2/3)/(4*(5 + 2^4))
x = 1.5159
```

Operazioni aritmetiche

Visualizzazione dei numeri sul display:

```
>>  format type
```

valori di type	risultato	pi
short	5-digit scaled fixed point	`3.1416`
short e	5-digit floating point	`3.1416e+000`
short g	Best of 5-digit fixed or floating point	`3.1416`
long	15-digit scaled fixed point	`3.14159265358979`
long e	15-digit floating point	`3.14159265358979e+000`
long g	Best of 15-digit fixed or floating point	`3.14159265358979`

Variabili predefinite

pi	π
i , j	unità immaginaria
Inf	Infinito (1/0)
NaN	Not a Number (0/0)
eps	2.2204e-16 precisione di macchina

Assegnazione di matrici e array

Modi equivalenti di generare un vettore riga:

```
>> v = [1 5 8 12]
>> v = [1,5,8,12]
```

Modi equivalenti di generare un vettore colonna:

```
>> v = [1;5;8;12]
>> v = [1 5 8 12]'
```

Generazione di una matrice di dimensione 2 x 3:

```
>> m = [1 6 2; 3 9 1]
m =
     1     6     2

     3     9     1
```

Gli intervalli

MATLAB permette di definire intervalli numerici in modo semplice ed automatico. Esistono per tale scopo specifici operatori e funzioni

L'operatore ":" consente la generazione di intervalli equispaziati
Sintassi → valore iniziale : incremento : valore finale
N.B.: l'incremento di default è pari a 1

```
x = 1:5      => x = [1 2 3 4 5]
x = 0:2:10   => x = [0 2 4 6 8 10]
x = 0:3:10   => x = [0 3 6 9]
x = 0:1.5:9  => x = [0 1.5 3.0 4.5 6.0 7.5 9.0]
x = 0:-1:-5  => x = [0 -1 -2 -3 -4 -5]
```

Gli intervalli

MATLAB permette di definire intervalli numerici in modo semplice ed automatico. Esistono per tale scopo specifici operatori e funzioni

L'operatore "linspace" crea un intervallo numerico prefissando il numero di punti piuttosto che l'incremento
Sintassi → linspace(valore iniziale, valore finale, numero punti)
N.B.: il numero di punti di default è 100

```
>> s = linspace(1,10,6)

s =

   1.0000    2.8000    4.6000    6.4000    8.2000   10.0000
```

Operazioni sulle matrici

Accedere agli elementi di un vettore: `v(i)`

Accedere agli elementi di una matrice: `m(i,j)`

Estrarre una riga della matrice: `row = m(i,:)`

Estrarre una colonna da una matrice: `col = m(:,j)`

Modificare elementi: `m(i,j) = #`

Estrazione di sottomatrici

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 0 & 6 \end{pmatrix}$$

Estrarre la colonna 2

```
v = A(:,2)
```

$$v = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

Estrarre una matrice 2 x 2

```
M = A(:,2:3)
```

$$M = \begin{pmatrix} 2 & 3 \\ 0 & 6 \end{pmatrix}$$

Matrici a blocchi

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

Costruire una matrice 2 x 4 da *A* → `B = [A,A]`

$$B = \begin{pmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{pmatrix}$$

Costruire una matrice 4 x 2 da *A* → `C = [A;A]`

$$C = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Funzioni utili lavorando con le matrici

eye(n)	matrice identica n x n
zeros(m,n)	matrice m x n con tutti elementi uguali a 0
ones (m,n)	matrice m x n con tutti elementi uguali a 1
size(x)	restituisce le dimensioni dell'array x
length(v)	restituisce il numero di componenti del vettore v

Funzioni utili lavorando con le matrici

diag(M)	restituisce un vettore contenente la diagonale della matrice M
diag(v)	restituisce una matrice quadrata con il vettore v sulla diagonale e zero altrove
tril(M,i)	estrae dalla matrice M tutti gli elementi sulla e sotto la i-esima diagonale Di default si ha i = 0 (diagonale principale)
triu(M,i)	estrae dalla matrice M tutti gli elementi sulla e sopra la i-esima diagonale Di default si ha i = 0 (diagonale principale)

Esercizio

Creare una matrice in cui la prima riga sia composta dai numeri da 1 a 10, la seconda riga composta dai numeri da 11 a 20 e la e la terza dai numeri da 21 a 30. Modificare la seconda riga in modo da annullarne gli elementi.

```
>> m = [1:10;11:20;21:30]

m =

     1     2     3     4     5     6     7     8     9    10
    11    12    13    14    15    16    17    18    19    20
    21    22    23    24    25    26    27    28    29    30

>> m(2,:) = 0

m =

     1     2     3     4     5     6     7     8     9    10
     0     0     0     0     0     0     0     0     0     0
    21    22    23    24    25    26    27    28    29    30
```

Operazioni tra matrici

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \qquad C = \begin{pmatrix} 0 & 3 & 1 \\ 1 & 2 & 4 \end{pmatrix}$$

Somma / differenza

A + B $\longrightarrow$ $\begin{pmatrix} 2 & 2 \\ 2 & 5 \end{pmatrix}$

A - B $\longrightarrow$ $\begin{pmatrix} 0 & 2 \\ 4 & 3 \end{pmatrix}$

A + C

??? Error using = => +

Matrix dimensions must agree.

Operazioni tra matrici

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$$

Prodotto

matriciale A * B → $\begin{pmatrix} -1 & 2 \\ -1 & 4 \end{pmatrix}$

elemento per elemento A .* B → $\begin{pmatrix} 1 & 0 \\ -3 & 4 \end{pmatrix}$

Determinante / rango / inversa

$$B = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \qquad D = \begin{pmatrix} 3 & 4 & -1 \\ 5 & 2 & 3 \\ 0 & 1 & -1 \end{pmatrix}$$

Determinante →
- det(B) ⟶ 1
- det(D) ⟶ 0

Rango → rank(D) ⟶ 2

Inversa →
- inv(B) ⟶ $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$
- inv(D) ⟶ ?

anche B^(-1)

Stringhe

In MATLAB una stringa è un vettore di caratteri:

```
>> s = 'oste';
>> s(1)
ans =
o
>> s(1) = 'a'
s =
aste
>> s = ['c',s]
s =
caste
```

Operazioni sulle stringhe

str2num	trasforma un array di caratteri numerici in un array di numeri (in formato double)
num2str	trasforma un array di numeri in un array di caratteri numerici
str2mat	trasforma una sequenza di stringhe in una matrice (ogni stringa della sequenza è una riga della matrice)
mat2str	converte una matrice numerica in una stringa contenente la sintassi MATLAB per la creazione di tale matrice
disp('text')	visualizza su schermo la stringa 'text'

La programmazione

In MATLAB si possono realizzare degli m-file, ovvero file di testo contenenti sequenze di comando e strutture di controllo che vengono interpretate

I file prodotti mediante un editor di testo devono essere salvati in un file con estensione .m, in una directory contenuta nel path

Le macro

Gli m-file di tipo macro operano sulle variabili contenute in memoria, e non esistono variabili locali

Contengono una serie di comandi che vengono automaticamente eseguiti quando si esegue la macro

Per eseguire un m-file basta digitarne il nome (senza l'estensione) dalla riga di comando

Per creare una macro:

1. aprire un file nuovo (File → New)
2. scrivere il codice della macro
3. salvare il file `filename.m`
4. eseguire da linea di comando `>> filename + INVIO`

Esempio

```
% ESEMPIO DI MACRO: calcola la matrice trasposta
% di una matrice A (presente in memoria) e ne
% visualizza l'output a schermo

Atrasp = A';
disp('La trasposta della matrice A è')
Atrasp
```

Le function

MATLAB permette di definire file di tipo funzioni. Tali funzioni vanno scritte in modo identico agli m-file, tranne per l'intestazione che è del tipo

```
function [variabili di uscita] =
                    nomefunzione(variabili di ingresso)
```

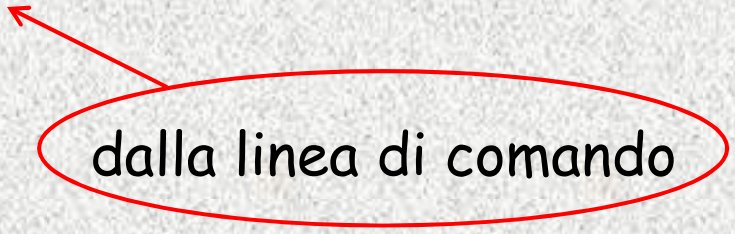
N.B.: le funzioni vanno salvato in un file avente lo stesso nome della funzione stessa. Tutte le variabili sono locali alla funzione, per cui dopo la sua esecuzione non restano in memoria

La function viene chiamata da linea di comando con

```
nomefunzione(valore_variabile_ingresso)
```

Esempio

```
% ESEMPIO DI FUNCTION: trasformare la macro
% precedente in una funzione

function [Atrasp] = trasposta(A)
Atrasp = A';


>> mat_trasp = trasposta(A)
```



Funzioni matematiche intrinseche

sqrt(x)	radice quadrata di x
round(x)	arrotondamento di x all'intero più vicino
fix(x)	parte intera di x
floor(x)	intero 'sinistro' più vicino a x
ceil(x)	intero 'destro' più vicino a x

cos(x), sin(x), tan(x)
cosh(x), sinh(x), tanh(x)
acos(x), asin(x), atan(x)
exp(x), log(x), log10(x)

Per un numero complesso z:

real(z)	parte reale di z
imag(z)	parte immaginaria di z
conj(z)	complesso coniugato di z

Istruzione if… else… elseif

if valuta un'espressione logica ed esegue una serie di istruzioni a seconda del valore dell'espressione logica

```
if  espressione logica 1
     istruzioni 1
elseif espressione logica 2
     istruzioni 2
else
     istruzioni 3
end
```

Operatori di relazione:

>	maggiore
<	minore
>=	maggiore o uguale
>=	minore o uguale
==	uguale
~=	diverso

Esempio

Aprire un file nuovo e salvarlo con nome dispar.m
La function prende in input un numero e controlla se è pari o dispari

```
function [] = dispar(x)

if rem(x,2) == 0
   disp('Il numero è pari')
else
   disp('Il numero è dispari')
end
```

N.B.: La funzione rem restituisce il resto di una divisione

Esempio

Aprire un file nuovo e salvarlo con nome inversa.m
La function prende in input una matrice 2x2 e restituisce, quando possibile, la sua matrice inversa

```
function B = inversa(A)

detA = A(1,1) * A(2,2) - A(1,2) * A(2,1);

if (detA == 0)
   disp('La matrice non è invertibile')
   B = [];
else
   B = 1 / detA * [A(2,2),-A(1,2);A(1,1),-A(2,1)];
end
```

Ciclo for

for esegue un numero di istruzioni per un numero fissato di volte

```
for  indice = inizio : incremento : fine
     istruzioni
end
```

Esempio: valor medio di un vettore

```
x = [1 2 3 4 5 6 7];
somma = 0;
for i=1:length(x)
    somma = somma + x(i);
end
media = somma / length(x);
```

N.B.: Il comando **break** forza l'uscita dal ciclo

Ciclo while

while esegue un numero di istruzioni finché l'espressione di controllo rimane vera

```
while espressione di controllo
     istruzioni
end
```

Esempio: dividere un numero per 2 finché il risultato non sia inferiore a 0.005 (contando il numero di divisioni effettuate)

```
a = 2390;  % dividendo
count = 0;
while (a/2 > 0.005)
     c = a/2;
     a = c;
     count = count + 1;
end
```

N.B.: Il comando **break** forza l'uscita dal ciclo

Istruzione switch

switch valuta un'espressione ed esegue un unico caso (**case**) possibile di istruzioni in base al valore di tale espressione

```
switch espressione_switch
   case espressione_case 1
         istruzioni 1
   case espressione case 2
         istruzioni 2
   …
   case espressione case n
         istruzioni n
   otherwise
         istruzioni n+1
end
```

Esempio

Creare una macro che chiede all'utente di inserire il grado di un polinomio e far visualizzare il nome della curva corrispondente

```
grado = input('Inserire un numero naturale');

switch grado
   case 0
        disp('Tipo di curva: retta orizzontale')
   case 1
        disp('Tipo di curva: retta obliqua')
   case 2
        disp('Tipo di curva: parabola')
   case 3
        disp('Tipo di curva: cubica')
   otherwise
        disp('Grado maggiore di 3')
end
```

Esercizio

Costruire una funzione fatt.m che restituisca il fattoriale di n. Inserire controlli sul numero n in ingresso (se è negativo, non intero,…)

```
function fn = fatt(n)
if (n < 0)
   disp('Errore: n negativo')
elseif (floor(n)-n ~= 0)
   disp('Errore: n non intero')
elseif (n == 0)
   fn = 1;
else
   fn = 1;
   for it = 1 : n
       fn = fn * it;
   end
end
```

Esercizio

Costruire una funzione hilbert.m che restituisca la matrice di Hilbert di ordine n H(i,j) = 1 / (i+j-1) (i,j = 1,…,n).
Inserire controlli sul numero n in ingresso (n deve essere un numero intero positivo)

```
function H = hilbert(n)
if (floor(n)==n & n > 1)
   for i = 1 : n
      for j = 1 : n
         H(i,j) = 1/(i + j - 1);
      end
   end
else
   disp('Errore: n deve essere un numero…
      intero positivo')
end
```

Operatori logici:

&	and
|	or
~	not

Operatori logici

Oltre agli operatori &, |, ~, in MATLAB sono presenti:

isempty	determina se un array è vuoto
isequal	determina se due array sono uguali
ismember	trova i membri di un insieme dentro un array
ischar	trova i caratteri dentro un array
isfloat	determina se un array è fatto di numeri reali
isinteger	determina se un array è fatto di numeri interi
find(expr)	valuta l'espressione 'expr' avente al proprio interno un array e ritorna le componenti dell' array in cui expr è verificata

Esempio:

```
>> find([4 3 2 1] >= 2)
ans =
     1   2   3
```