Introduzione a MATLAB

Fabio Solari e Manuela Chessa

ATTUALE TENDENZA DEL SOFTWARE

- Semplificare e velocizzare lo sviluppo delle applicazioni
- Utilizzo di paradigmi di *programmazione visuale* (ambienti RAD, *Rapid Application Development*):
  - linguaggi tradizionali
    - Basic, C/ C++
  - nuovi linguaggi
    - gestione di strumentazione (LabVIEW)
    - gestione web (Java)

ATTUALE TENDENZA DEL SOFTWARE

- Utilizzo di paradigmi di *programmazione classica* con apporto di *funzioni specifiche* precompilate
  - linguaggi tradizionali
    - C, C++ (NAG library)
  - nuovi *ambienti di sviluppo* (con interfaccia utente interattiva di tipo **prompt**)
    - calcolo simbolico (Mathematica)
    - calcolo numerico (MATLAB)

CALCOLO SIMBOLICO VS. CALCOLO NUMERICO

- Mathematica

In[1]:=Expand[(a+b)^2]

Out[1]=$a^2 + 2ab + b^2$

se a=1/2 e b=1/3

In[4]:=Expand[(a+b)^2]

Out[4]= 25/36

- MATLAB

» (a+b)^2

??? Undefined function or variable 'a'.

se a=1/2 e b=1/3

» (a+b)^2

ans = 0.6944

ATTUALE TENDENZA DEL SOFTWARE

- Vantaggi
  - apprendimento veloce
  - semplice anche per non esperti
  - algoritmi trasparenti
  - pensare al proprio problema
  - ambienti completi :
    - editing, compilazione e debugging
    - gestione codice e gestione dati
- Svantaggi
  - minore flessibilità
  - codice non ottimizzato
  - tendenza ad una eccessiva semplificazione

MATLAB

- Nel corso degli anni MATLAB ha beneficiato del feedback degli utenti.
- Il programma è largamente impiegato sia in *campo educativo* sia in *campo applicativo*:
  - nelle *università* è utilizzato sia come strumento per la ricerca sia come strumento di apprendimento e di esercitazione con il quale implementare le nozioni apprese durante i corsi teorici
  - nell'*industria* è lo strumento preferito per la realizzazione di progetti di ricerca, sviluppo e analisi.

MATLAB

- È utilizzato in tutti i tipi di calcolo perché
  - si impara ed usa velocemente
  - è un linguaggio ad alto livello per le operazioni matriciali
  - incoraggia a trovare *soluzioni vettoriali*
  - può essere utilizzato interattivamente
  - fornisce molte *funzioni grafiche*
  - può essere interfacciato con C/C++ e Fortran
  - presenta diversi tipi di interfaccia per le diverse aree di applicazione
  - nasconde all'utente i *dettagli architetturali o algoritmici* non necessari, oppure ne permette la modifica

PSE

- Queste caratteristiche rendono MATLAB
  - un PSE (*Problem Solving environment*): sistemi software che forniscono tutti gli strumenti per risolvere problemi in una particolare area (e.g. Microsoft Word).
  - un "*Rapid Prototyping Environment*": sistemi software che permettono di testare algoritmi ed idee rapidamente e facilmente.

TESTI DI RIFERIMENTO

- La documentazione fornita con MATLAB:
  - I manuali online
  - L'help online disponibile in MATLAB
- Materiale disponibile in rete:
  - http://www.mathworks.com

SOFTWARE DISPONIBILE

- Altri prodotti simili a MATLAB, anche per SO diversi (e.g. Linux), sono:
  - **gnuplot** (http://www.gnuplot.info/)
  - **Octave** (http://www.gnu.org/software/octave/)
  - **Scilab** **(**http://www.scilab.org/)

MATLAB

- MATLAB (*MATrix LABoratory*) è un ambiente di sviluppo interattivo per il calcolo scientifico
- Originariamente sviluppato come interfaccia per il software matriciale LINPACK e EISPACK

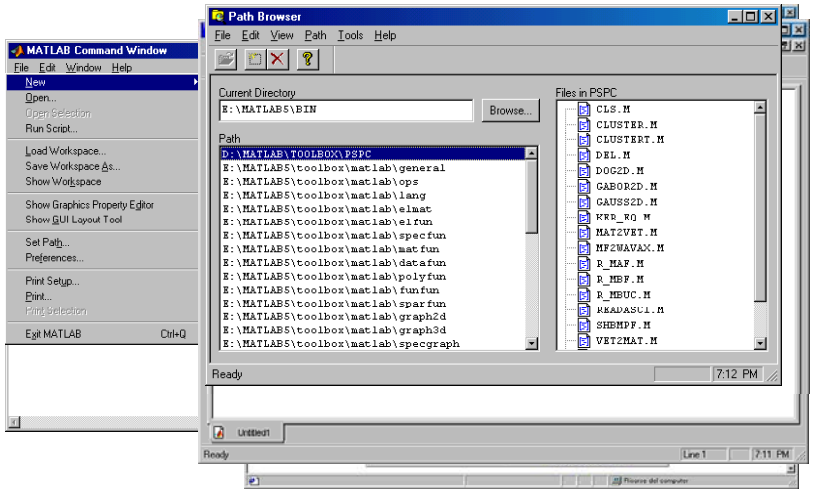
→ L'elemento base è la **matrice**, che non richiede dimensionamento

→ Ideale per risolvere problemi con formulazione matriciale o vettoriale

MATLAB

- MATLAB è composto di 6 parti principali:
  - L'ambiente di lavoro
  - Le librerie di funzioni matematiche
  - Il sistema grafico
  - Il linguaggio
  - API
  - Toolbox

AVVIO DI MATLAB



L'AMBIENTE MATLAB

- Sono forniti generici comandi di sistema per manipolare i file: ls, cd, delete, type …
- MATLAB esegue funzioni che sono nel path o nella directory corrente. Si aggiungono percorsi al path attraverso il menu File e Set Path
- Si possono eseguire programmi esterni tramite il carattere !

>>!notepad

Con il simbolo >> indico il prompt di MATLAB

L'AMBIENTE MATLAB

- Il comando help

>>help sqrt

Caratteri maiuscoli per evidenziare, ma tutte le funzioni devono essere chiamate in minuscolo.

SQRT Square root.

SQRT(X) is the square root of the elements of X. Complex results are produced if X is not positive.

See also SQRTM.

L'AMBIENTE MATLAB

- *help* fornisce una lista delle categorie delle funzioni
- *help categoria* mostra le funzioni di quella categoria
- *helpwin* genera una finestra di help
- *lookfor keyword* cerca le funzioni che corrispondono alla parola chiave fornita
- *helpdesk* fornisce la finestra di aiuto in html
- *doc funzione* richiama helpdesk per la data funzione

ALCUNI CARATTERI SPECIALI

- % è il commento
- … è la continuazione sulla riga successiva
- = è l'operatore di assegnamento
- == è l'operatore di uguaglianza
- ; impedisce l'echo su monitor
- , separa argomenti o comandi
- Ctrl-C termina l'esecuzione di un comando

L'AMBIENTE MATLAB

- Lo *workspace* è l'area di memoria accessibile dal prompt, dove si lavora.
- Per visualizzare i dati si utilizzano
  - *who* e *whos*
- Per cancellare dati in memoria
  - *clear nome_variabile*

```
>> a=[1 2 3];  ii=1;
>>who
Your variables are:
a       ii
>> whos
  Name      Size          Bytes  Class
  a         1x3              24  double array
  ii        1x1               8  double array
Grand total is 4 elements using 32 bytes
>>clear a,  whos
  Name      Size          Bytes  Class
  ii        1x1               8  double array
Grand total is 1 elements using 8 bytes
```


L'AMBIENTE MATLAB

- Per salvare l'intero workspace si usa *save nomefile*, per caricarlo si usa *load nomefile*
- Per salvare variabili in formato binario si usa *save nomefile variabili*, per caricarle *load nomefile*

```
>>a=[1 2];  b=[3;4];
>>save ab
>>ls ab
  ab.mat
>>clear a b
>>a
??? Undefined function or variable 'a'.
```

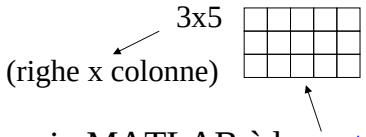
estensione .mat

```
>>load ab
>>a
  a = 1  2
>>b
   b=  1
       2
```

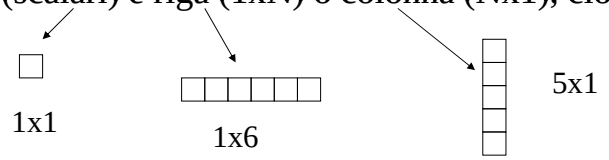

DEFINIZIONE DI VARIABILI

- MATLAB non usa *definizione di tipo* o *dichiarazione di dimensioni*
- Crea automaticamente la variabile digitata
- I nomi delle variabili sono *case sensitive*
- Devono iniziare con una *lettera* e possono contenere *31 caratteri* (lettere, numeri e underscore, ma non punti)

STRUTTURE DATI: MATRICI



- L'elemento base in MATLAB è la *matrice rettangolare*: particolare interesse per quella 1x1 (scalari) e riga (1xN) o colonna (Nx1), cioè vettori



Non c'è *dichiarazione* di variabili, non si usa la *definizione* di tipo e non è richiesto *dimensionamento*

CREARE MATRICI

- Si possono creare matrici in diversi modi:
  - Scrivere esplicitamente gli elementi
  - Caricare gli elementi da file esterni di dati
  - Generare gli elementi utilizzando le *built-in function*
  - Generare gli elementi utilizzando i propri **M-file** (*file che contengono il codice sorgente dei programmi MATLAB ed hanno estensione .m*)

CREARE MATRICI

- Per scrivere esplicitamente gli elementi:
  - Separare gli elementi di una riga con *spazi* o *virgole ,*
  - Finire le righe con *punto e virgola ;*
  - Tutti gli elementi devono essere racchiusi tra *parentesi quadrate* [ ]

```
>>A = [ 16 3 2 13 ; 5 10 11 8 ; 9 6 7 12 ; 4 15 14 1 ]
A = 16   3   2  13
     5  10  11   8
     9   6   7  12
     4  15  14   1
```


ELEMENTI DI UNA MATRICE

- Per accedere all'elemento nella *riga i* e *colonna j* si usa la notazione *A(i,j)*

```
>>A(2,3)
  ans=  11
```

- Se a=[1 2 ; 3 4] ;

N.B.
Gli indici devono essere interi positivi (non 0)

```
>> t=a(2,3)
      ??? Index exceeds matrix dimensions.
>>a(2,3)=5
     a = 1   2   0
         3   4   5
```

Possibili errori legati al non dimensionamento delle matrici

SCALARI E VETTORI

```
>>a=1
a =
    1

>>b=[1  2  3  4  5]
b =
    1    2    3    4    5

>>b(3)
ans =
    3
```

Scalare 1x1

vettore riga 1x5

```
>>c=[1; 2; 3]
c =
    1
    2
    3

>> c(2)
ans =
    2
```

vettore colonna 3x1

variabile predefinita

DIMENSIONI

- Per conoscere le dimensioni di una matrice si usa la funzione *size()*: *[r,c]=size(M)*

```
>>a=[1  2  3; 4  5  6];
>> [r,c] = size(a)
r  =  2
c  =  3
```

Notazione usata per gestire output multipli da una funzione

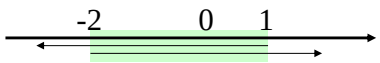
- Per conoscere il numero di elementi di un vettore si usa la funzione *length()*: *n=length(M)*

```
>>a=[1 2 3 4];  length(a)
ans  = 4
```


L'OPERATORE :

- Vediamo come rappresentare *insiemi numerici*: insiemi di numeri in cui l'informazione è contenuta nel valore iniziale, nel valore finale e nel passo di "campionamento" usato (il valore della differenza tra due numeri contigui dell'insieme considerato)
- Matematicamente si considera un *insieme infinito di valori* compresi tra due estremi:

$$\sqrt{(1-x)(2+x)}$$



$$x \in [-2,1]$$

L'OPERATORE :

- A causa della precisione finita dei numeri in un calcolatore, si considera un *insieme finito e quindi discreto* di valori compresi tra due estremi (e.g. [-2, 1] con incrementi di 0.1: quindi l'insieme di numeri {-2.0, -1.9, -1.8 … 0.8, 0.9, 1.0})

```
>>1:10
   ans = 1    2    3    4    5    6    7    8    9  10
>>10: -2 :5
    ans = 10   8   6
>>0:pi/4:pi
    ans = 0    0.7854    1.5708    2.3562    3.1416
```

L'OPERATORE :

- È di fondamentale importanza per creare *vettori*
- Per creare *tabelle di valori di funzioni* (dominio e condominio) si utilizza l'operatore :

Rappresentazione a precisione finita

```
>> x=0 : pi/4 : pi;
>> sin(x)
ans =  0   0.7071   1.0000   0.7071   0.0000
```

Formulazione vettoriale: equivalente a ciclo for e sin(x(i))

MATRICI

- 4 funzioni base
  - zeros
  - ones
  - rand
  - randn

```
>> a=2*ones(2,3)
   a =2   2   2
      2   2   2
>>b=rand(3,3)
  b = 0.9501   0.4860   0.4565
      0.2311   0.8913   0.0185
      0.6068   0.7621   0.8214
```

- Per eliminare righe o colonne utilizzare []

```
>>a=[1  2  3; 4  5  6]
   a =1   2   3
      4   5   6
>>a(1,:)=[]
   a=4   5   6

>>a=[1  2  3; 4  5  6];

>>a(:,1)=[]
    a = 2   3
        5   6
```


MATRICI

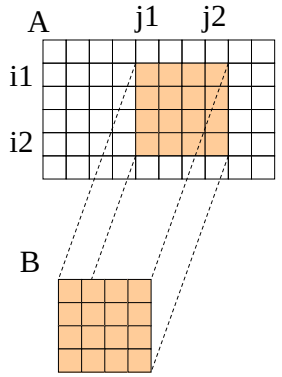
- Per gestire le matrici si fa riferimento all'algebra delle matrici. Si considerano solo l'estrazione di sottomatrici e le operazioni di addizione e sottrazione

- Si considera l'operazione elemento per elemento: funzioni (come quella vista: *sin(x)*) e operazioni in cui le variabili sono trattate come "scalari". Cioè la funzionalità viene applicata ai singoli elementi della matrice

MATRICI

- La gestione delle sottomatrici avviene per mezzo di indici vettoriali

```
>>a=[1 2 3 4; 5 6 7 8; 9 10 11 12]
a =  1   2   3   4
     5   6   7   8
     9  10  11  12
>>a(2:3,2:4)
ans =
   6   7   8
  10  11  12
```



B=A(i1:i2,j1:j2);

MATRICI

- Somma e sottrazione:

```
>>a=[1  2; 3  4];b=ones(2,2);
>>c=a+b
   c= 2   3
      4   5
>>c=a-3
   c= -2  -1
       0   1
>>a+ones(3,3)
??? Error using ==> +
Matrix dimensions must agree.
```

- Moltiplicazione

```
>>A=a*a
   A=   7   10
       15   22
>>(2*ones(1,2)) * a
     ans =  8   12
>>a*pi
    ans =3.1416    6.2832
         9.4248   12.5664
```


MATRICI

- Per operare elemento per elemento si pone il . prima del corrispondente operatore:

```
>>a=[1  2; 3  4]; a.*a
     ans=  1    4
           9   16
>>a./(2*ones(2,2))
      ans =0.5000   1.0000
           1.5000   2.0000
>> a.\(2*ones(2,2))
      ans =2.0000   1.0000
           0.6667   0.5000
```


ESPRESSIONI

- I blocchi costituenti le espressioni sono (ricordarsi che agiscono su intere matrici):

- Variabili *(appena viste)*
- Numeri
- Operatori
- Funzioni

NUMERI E OPERATORI

- Si usa la notazione decimale, e per la notazione scientifica e i o j per i numeri immaginari

3	-99
0.0120	-9.1233
1.34512 e 23	6.34567 e -20
-2.3 i	2.1 + 3.2 i

- Sono memorizzati in formato *long* IEEE

- \+ somma
- – sottrazione
- * moltiplicazione
- / divisione
- \ divisione sinistra
- ^ esponente
- ' trasposta (compl. con.)
- . operatore elemento per elemento
- ( ) ordine di valutazione

NOMI BULT-IN

- È utile non sovrascrivere i seguenti nomi built-in:
  - ans
  - pi (3.1415)
  - eps (2.2204e-016)
  - realmin (2.2251e-308)
  - realmax (1.7977e+308)
  - i, j ($\sqrt{-1}$)
  - nargin
  - nargout
  - Inf
  - NaN
  - flops

- computer
- date
- clock
- cputime

FUNZIONI

- Arrotondamento

```
>>round(2.3)
ans =
    2

>>floor(2.3)
ans =
    2

>>ceil(2.3)
ans =
    3
```

```
>>round(2.7)
ans =
    3

>>floor(2.7)
ans =
    2

>> ceil(2.7)
ans =
    3
```


FUNZIONI

- Approssimazioni razionali e fattorizzazione intera

```
>>rem(11,3)
ans =
    2

>> rats(1/2 -1/3 +1/5)
ans =
    11/30

>>gcd(27,72)
ans =
    9
```


FUNZIONI

- Aritmetica complessa

```
>>z1=1+3*j;z2=2-5*j;
>> z1*z2
ans =
  17.0000 + 1.0000i

>>real(z1)
ans =
    1

>> imag(z1)
ans =
    3
```

```
>> j=1;
>>z1=1+3*j
z1 =
    4

>> j=sqrt(-1)
j =
       0 + 1.0000i

>> z1=1+3*j
z1 =
   1.0000 + 3.0000i
```

FUNZIONI

- Esponenziali, logaritmiche, trigonometriche e specifiche di particolari ambiti scientifici.
- In generale possono operare su dati complessi e matriciali

```
>>pow2(10)
ans =
      1024

>>log10(10)
ans =
   1
```

```
>> asin(0.6)
ans =
0.6435

>>factorial(15)
ans =
 1.3077e+012
```


OPERATORI RELAZIONALI E LOGICI

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

- & and
- | or
- xor or esclusivo
- ~ not

- Il valore *falso* è indicato con 0
- Il valore *vero* con 1

```
>>a=[1 2 3; 2 2 5];
>>A=((a/2)==1)
A =
   0   1   0
   1   1   0
```


POLINOMI

- I *polinom*i si rappresentano come *vettori riga* contenenti i coefficienti in ordine di potenze decrescenti

  – $p(x)=x^3 + 3x^2 + 2x +10$ si rappresenta come p=[1 3 2 10]

  – $p(x)=x^3 + 1$ si rappresenta come p=[1 0 0 1]

POLINOMI

- La costruzione di un polinomio con specifiche radici si esegue con la funzione *poly()*. Radici: -2 -j3, -2 +j3, -5; ovvero il polinomio p(x)=(x+2+j3)(x+2-j3)(x+5)

```
>>r=[-2-j*3,-2+j*3,-5];
>>p=poly(r)
p =
   1    9   33   65
```

POLINOMI

- Il prodotto di polinomi si esegue con la funzione *conv()*

```
>>p1=[1 3 5]; p2=[1 -2 4];
>> p3=conv(p1,p2)
p3 =
    1    1    3    2   20
```

$p3(x)=x^4 + x^3 + 3x^2 + 2x +20$

- L'operazione duale è la divisione di polinomi, ottenuta con *deconv()*. La sintassi è *[q,r]=deconv(a,b)* che fornisce due polinomi q(x) (quoziente) e r(x) (resto) tali che a(x)=q(x)b(x)+r(x)

POLINOMI

- Le radici di un polinomio si calcolano con *roots()*

```
>>p=[1 9 33 65]
p =
    1    9   33   65

>>roots(p)
ans =
  -5.0000
  -2.0000 + 3.0000i
  -2.0000 - 3.0000i
```

- *polyval(p,x)* calcola il valore p(x) con x dato

```
>> polyval(p,-5)
ans =
    0

>>polyval(p,2.3)
ans =
  200.6770
```


POLINOMI

- Per effettuare la derivata di polinomi sono disponibili le seguenti funzioni
  - *q=polyder(p)*: derivata del polinomio $q(x)=\frac{dp(x)}{dx}$
  - *q=polyder(p1,p2)*: derivata del prodotto $p1(x)\,p2(x)$
  - *[q,d]=polyder(p1,p2)*: derivata del rapporto sotto forma di funzione razionale fratta $\frac{q(x)}{d(x)}=\frac{d}{dx}\frac{p1(x)}{p2(x)}$

POLINOMI

```
>> p=[1 2 3 4];
>>q=polyder(p)
q =
    3    4    3

>> p1=[1 2 3];p2=[4 5 6];
>> q=polyder(p1,p2)
q =

   16   39   56   27
```

```
>> p1=[1 1 1];p2=[1 0 4];
>> [q,d]=polyder(p1,p2)
q =
   -1    6    4

d =
    1    0    8    0   16
```

VISUALIZZAZIONE SCIENTIFICA

- L'integrazione delle funzionalità di calcolo numerico con le elevate capacità grafiche è un punto di forza dell'ambiente MATLAB
- La visualizzazione scientifica è la rappresentazione grafica di dati. Utile nel
  - trovare modelli
  - identificare tendenze
  - comparare informazioni complesse
  - esaminare oggetti che non possono essere esaminati fisicamente

VISUALIZZAZIONE SCIENTIFICA

- Sono forniti un ampio insieme di funzioni ad alto livello per visualizzare dati (2-D, 3-D , movies, etc.)
- Inoltre si ha la possibilità di accedere anche alle proprietà a basso livello degli oggetti grafici (*Handle Graphics*) per gestire completamente la visualizzazione dei dati

GRAFICI

- Per default MATLAB traccia grafici sulla finestra 1; si possono aprire altre finestre di visualizzazione con il comando *figure*.
- Si chiude la finestra corrente con il comando *close*
- Con *figure(n)* si rende corrente la figura numero n e con *close(n)* si chiude la figura numero n
- Per visualizzare grafici bidimensionali (2-D) si utilizza la funzione *plot()*

GRAFICI

- Consideriamo come varia la densità dell'aria ρ (in kg/m^3) al variare della quota h (in km):

h	7	10	15	21	27	34	39	43	47	51
ρ	556	369	191	75	26.2	9.9	4.4	2.3	1.4	0.8

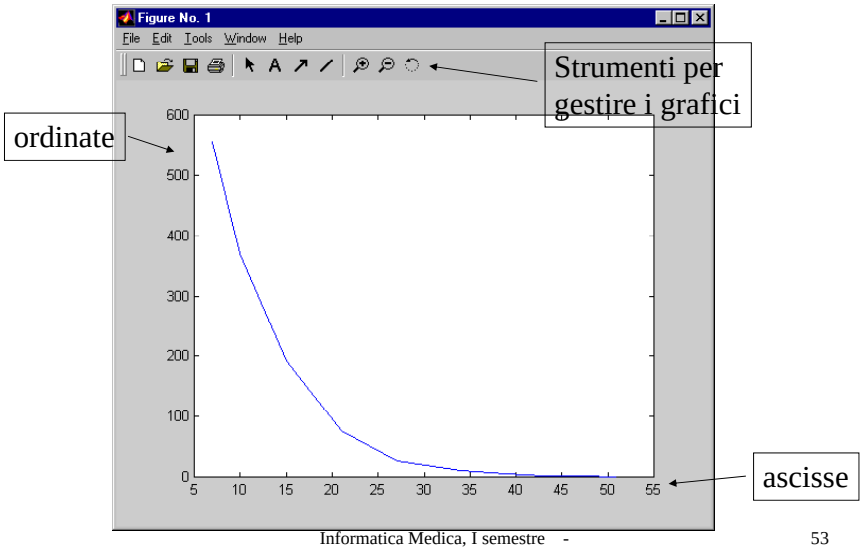
Inseriamo i dati in due vettori

```
>>h = [7   10  15  21  27  34  39  43  47  51];
>>r = [556 369 191 75  26.2 9.9 4.4 2.3 1.4 0.8];
>>plot(h,r)
```

Visualizziamo i dati tramite *plot(ascisse, ordinate)*

GRAFICI

Finestra di visualizzazione



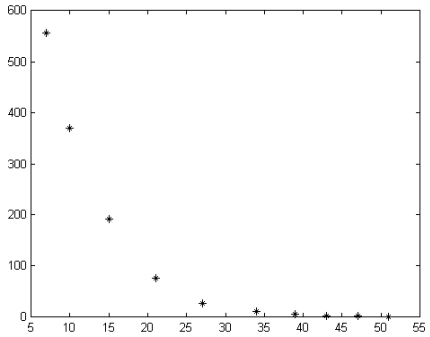
GRAFICI

- La scalatura degli assi è automatica
- Il titolo della finestra (Figure No. 1) indica il numero della finestra stessa
- I punti tracciati sono automaticamente congiunti da linee a tratto pieno
- Non si è dovuto scrivere codice specifico per produrre un output grafico

GRAFICI

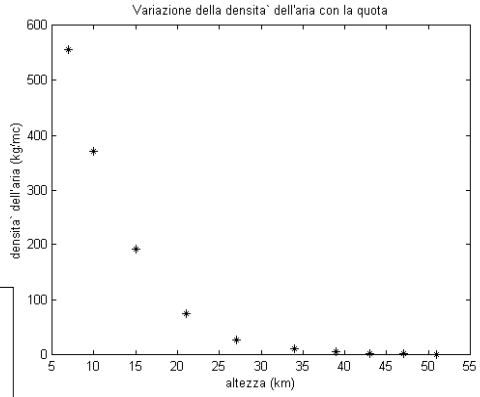
- Congiungere con linee i dati sui grafici è visivamente gradevole, tuttavia è importante indicare anche i dati effettivi: si possono usare parametri opzionali di *plot()*

>>plot(h,r,'')*



GRAFICI

- Si devono sempre indicare ascisse, ordinate e titolo di un grafico

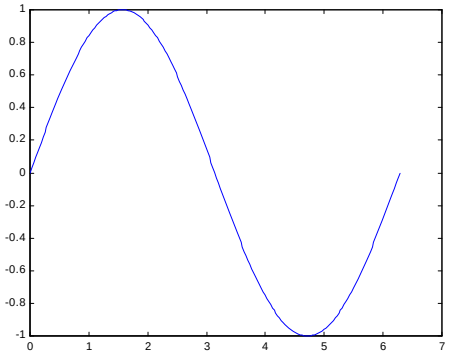



```
>>xlabel('altezza (km)')
>>ylabel('densita` dell''aria
       (kg/mc)')
>>title('Variazione della
      densita` dell''aria con
      la quota')
```

GRAFICI: 2-D

- Per tracciare il grafico di una funzione y=f(t), si deve creare una tabella di valori della funzione (come abbiamo visto): specificare un dominio discreto con l'operatore : e calcolare i corrispondenti valori di codominio

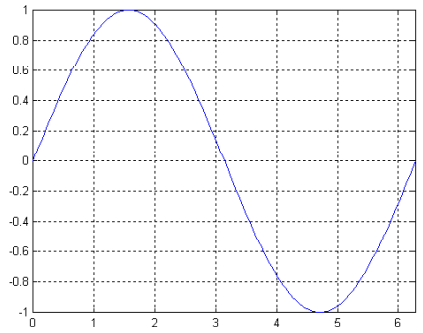
```
>>t=0:pi/100:2*pi;
>> y=sin(t);
>> plot(t,y)
```



GRAFICI : 2-D

- Esistono vari controlli sugli assi
  - axis([xmin xmax ymin ymax])
  - axis square , axis equal
  - axis auto
  - axis on , axis off
  - grid on , grid off

```
>>axis([0  2*pi  -1  1])
>>grid on
```

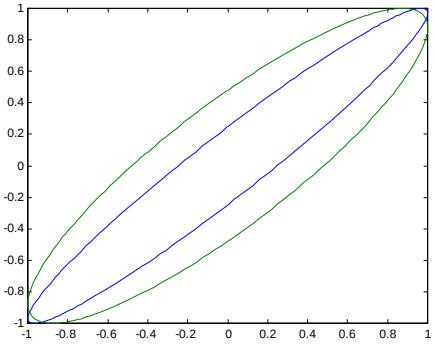


GRAFICI : 2-D

- Si possono visualizzare più grafici insieme

```
>>t=0:pi/100:2*pi;
>> x=sin(t);
>> y1=sin(t+0.25);
>> y2=sin(t+0.5);
>> plot(x,y1,x,y2)
```

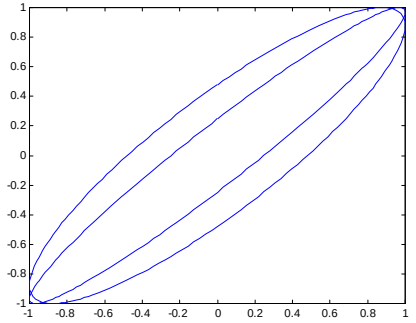
Si devono ripetere ascisse e ordinate



GRAFICI : 2-D

- Si possono aggiungere curve ad un grafico esistente con il comando *hold on*

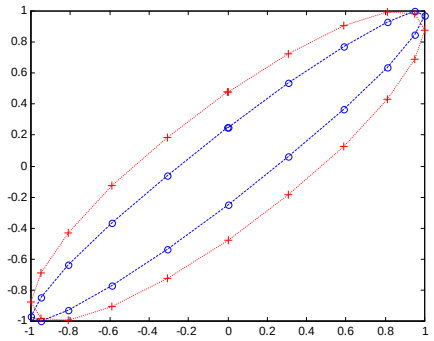
```
>>t=0:pi/100:2*pi;
>> x=sin(t);
>> y1=sin(t+0.25);
>> y2=sin(t+0.5);
>> plot(x,y1)
>>hold on
>> plot(x,y2)
```



GRAFICI : 2-D

- Si possono differenziare i grafici utilizzando i parametri opzionali di plot()

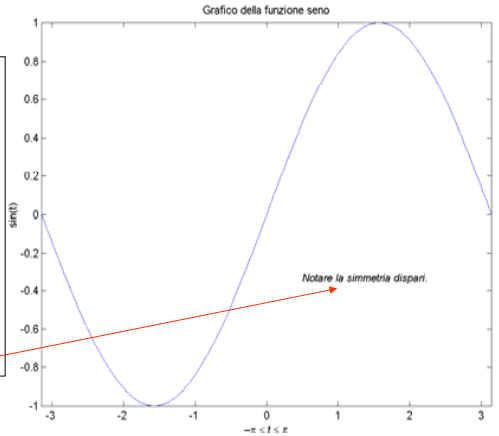
```
>>t=0:pi/100:2*pi;
>> x=sin(t);
>> y1=sin(t+0.25);
>> y2=sin(t+0.5);
>> plot(x,y1,'b--o')
>>hold on
>> plot(x,y2,'r:+')
```



GRAFICI : 2-D

- Si possono inserire titoli, testi ed etichette (LaTeX)

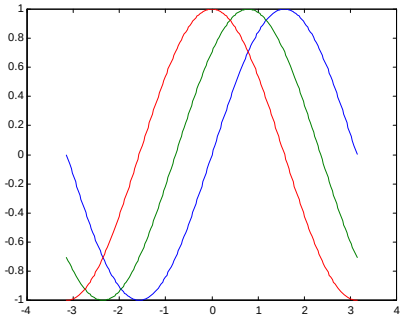
```
>>t = -pi:pi/100:pi;
>> y = sin(t);
>> plot(t,y)
>> axis([-pi pi -1 1])
>> xlabel('-\pi \leq \itt \leq \pi')
>> ylabel('sin(t)')
>> title('Grafico della funzione seno')
>> text(0.5,-1/3,'\it{Notare la simmetria dispari.}')
```



GRAFICI : 2-D

- plot() visualizza le colonne di matrici in ingresso

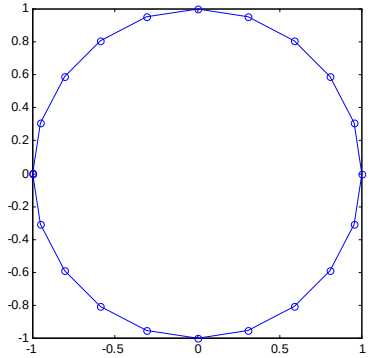
```
>> t = -pi:pi/100:pi;    t=t';
>> a=[sin(t) sin(t+pi/4) sin(t+pi/2)];
>> plot(t,a)
```



GRAFICI : 2-D

- plot() visualizza anche numeri complessi: traccia una curva nel piano complesso (Re{},Im{}). Per esempio la funzione complessa $f(t)= e^{jt}$

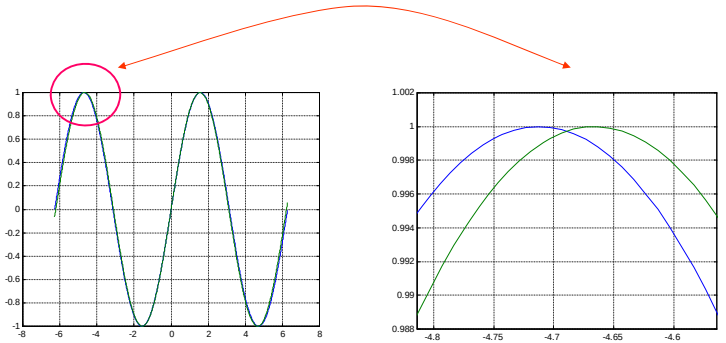
```
>>t = -pi:pi/10:pi;
>>plot(exp(i*t),'-o')
>>axis square
```



GRAFICI : 2-D

- Utile funzione è zoom:

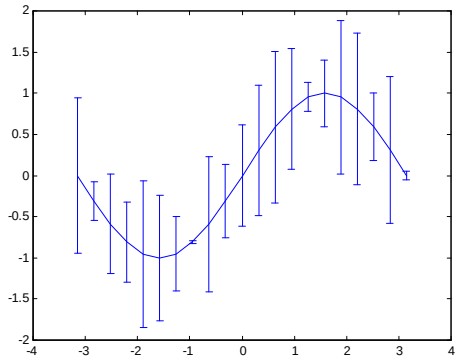
```
>>x=-2*pi:0.01:2*pi;
>>plot(x,sin(x),x,sin(x*1.01))
>> zoom xon, grid on
```



GRAFICI : 2-D

- Vi sono molte altre funzioni specifiche per visualizzare grafici: g.e. errorbar(), bar()

```
>>t = -pi:pi/10:pi;
>>y=sin(t);
>>e=rand(size(t));
>>errorbar(t,y,e)
```

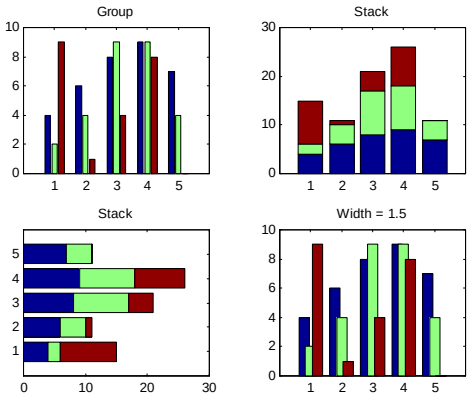


GRAFICI : 2-D

- Per visualizzare più grafici sulla stessa finestra si usa *subplot()*

```
>>figure
>> Y = round(rand(5,3)*10);

>> subplot(2,2,1)
>> bar(Y, 'group')
>> title 'Group'
>> subplot(2,2,2)
>> bar(Y, 'stack')
>> title 'Stack'
>> subplot(2,2,3)
>> barh(Y, 'stack')
>> title 'Stack'
>> subplot(2,2,4)
>> bar(Y,1.5)
>> title 'Width = 1.5'
```



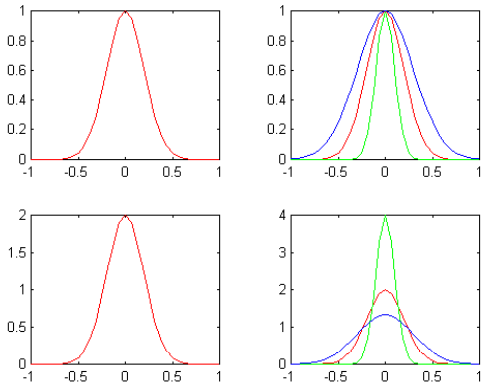
GRAFICI : 2-D

Consideriamo la funzione Gaussiana $\begin{cases} g(x) = \exp(-\frac{x^2}{2\sigma^2}) \\ g(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{x^2}{2\sigma^2}) \end{cases}$

```
x=-1:2/32:1;
figure
subplot(2,2,1)
s=0.2, plot(x,exp(-(x.^2)/(2*s^2)),'r')
subplot(2,2,2)
s=0.2;, plot(x,exp(-(x.^2)/(2*s^2)),'r'),hold on
s=0.1; plot(x,exp(-(x.^2)/(2*s^2)),'g')
s=0.3; plot(x,exp(-(x.^2)/(2*s^2)),'b'),hold off
subplot(2,2,3)
s=0.2; plot(x,(1/(sqrt(2*pi)*s))*exp(-(x.^2)/(2*s^2)),'r')
subplot(2,2,4)
s=0.2; plot(x,(1/(sqrt(2*pi)*s))*exp(-(x.^2)/(2*s^2)),'r')
hold on
s=0.1; plot(x,(1/(sqrt(2*pi)*s))*exp(-(x.^2)/(2*s^2)),'g')
s=0.3; plot(x,(1/(sqrt(2*pi)*s))*exp(-(x.^2)/(2*s^2)),'b')
hold off
```

GRAFICI : 2-D

$$g(x) = \exp(-\frac{x^2}{2\sigma^2})$$



red σ=0.2
green σ=0.1
blue σ=0.3

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{x^2}{2\sigma^2})$$

GRAFICI: 3-D

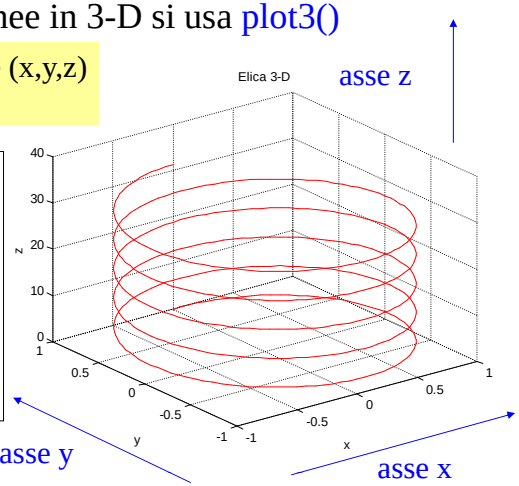
- La *grafica tridimensionale* permette di visualizzare
  - traiettorie (linee) nello spazio tridimensionale
  - superfici generate da funzioni di due variabili f(x,y) o parametriche
  - dati che dipendono da due variabili (e.g. mappe)
  - immagini
  - solidi di rotazione

GRAFICI: 3-D

- Per visualizzare linee in 3-D si usa plot3()

Si specificano le coordinate (x,y,z) dei punti della traiettoria

```
>>t=0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t,'r')
>> grid on
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
>> title('Elica 3-D')
```



GRAFICI: 3-D

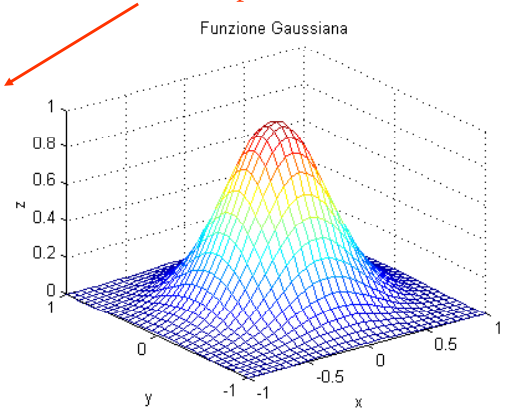
- Anche per funzioni di 2 variabili ($z=f(x,y)$) si deve creare una tabella di valori:
  - si specifica un dominio *discreto* bidimensionale (una griglia su un piano *(x,y)*) con la funzione *meshgrid()*
  - si valuta la funzione *f()* nei valori del dominio *(x,y)*
  - infine si visualizzano i valori calcolati con la funzione *mesh()*

GRAFICI: 3-D

$$z = g(x,y) = \exp\left(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2}\right)$$

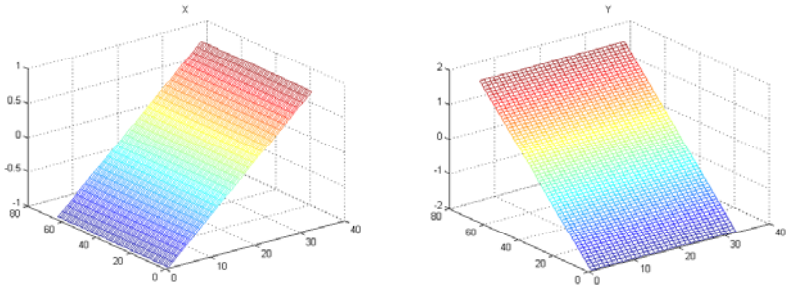
Dominio di campionamento

```
>>sx=0.35; sy=0.35;
>>[X,Y]=meshgrid(-1:2/32:1);
>> Z=exp(-(X.^2)/(2*sx^2) -…
(Y.^2)/(2*sy^2));
>> mesh(X,Y,Z)
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
>> title('Funzione Gaussiana)
```



GRAFICI: 3-D

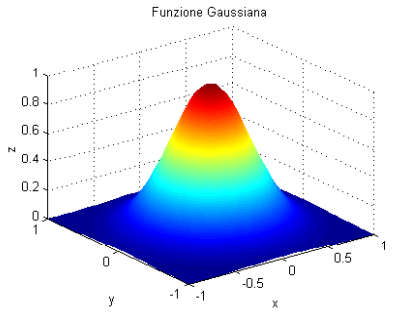
```
>>[X,Y]=meshgrid(-1:2/32:1,-2:2/32:2);
>>figure, mesh(X),title('X')
>>figure, mesh(Y), title('Y')
```



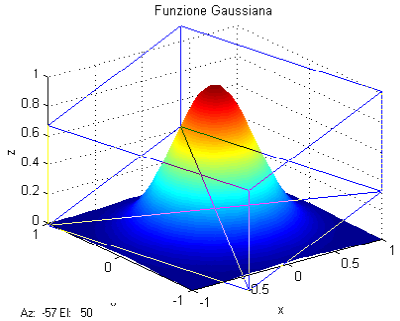
GRAFICI: 3-D

- Esistono molte altre funzioni di visualizzazione (e.g. surf) e di manipolazione (e.g. rotate3d)

```
>>surf(X,Y,Z), shading interp
```

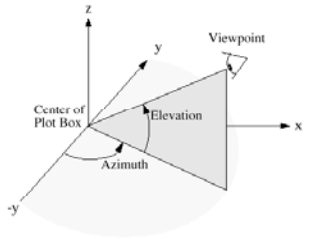



```
>>rotate3d
```

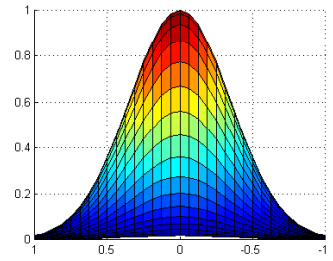


GRAFICI: 3-D

- Si cambia vista di un grafico con view
  - view(azimuth,elevation) o view([x y z])




```
>>surf(X,Y,Z), shading faceted, view([0 1 0])
```

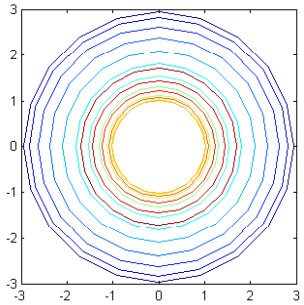


GRAFICI: 3-D

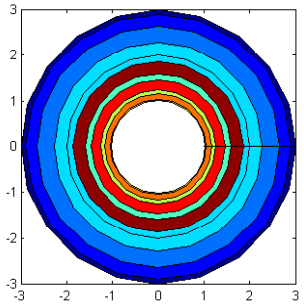
- Consideriamo funzioni che non producono visualizzazioni nello spazio tridimensionale, ma che agiscono comunque su matrici
  - contour(X,Y,Z,n) visualizza curve di livello
  - image(x,y,Z) visualizza dati come immagini
  - pcolor(X,Y,C) visualizza dati come matrici di celle
- Immagini: una griglia di punti il cui valore è codificato con un colore o livello di grigio. Si può considerare una matrice: le righe e colonne formano la struttura spaziale dell'immagine e il valore degli elementi costituiscono il colore

GRAFICI: 3-D

```
>>contour(X,Y,Z,15)
>>axis square
```

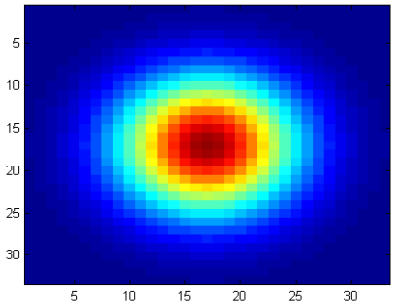


```
>>contourf(X,Y,Z,10)
>>axis square
```

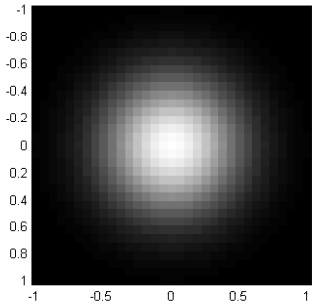


GRAFICI: 3-D

```
>>imagesc(Z)
```

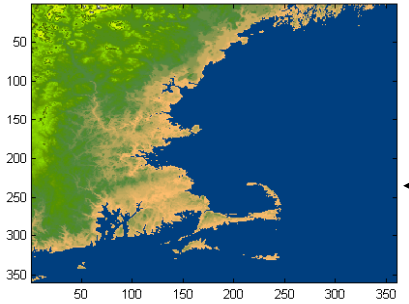


```
>>imagesc(Z)
>>axis square
>>colormap(gray)
```



GRAFICI: 3-D

```
>>load clown
>>imagesc(X)
>>colormap(map)
>>axis off
```



```
>>load cape
>>imagesc(X)
>>colormap(map)
```

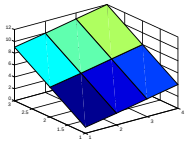
GRAFICI: 3-D

- Attenzione a come le funzioni visualizzano i dati

```
>>a=[1 2 3 4 ;5 6 7 8; 9 10 11 12 ];
>>figure,  surf(a)
>>figure,  imagesc(a)
>>colormap(gray)
>>figure,  pcolor(a)
>>colormap(gray)
>>a= 1   2   3   4
      5   6   7   8
      9  10  11  12
```

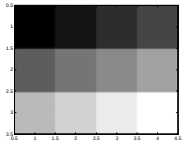
C

R



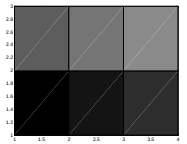
C

R



C

R



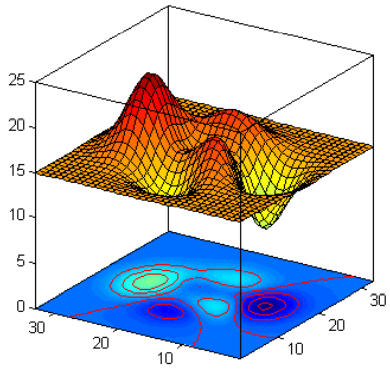
C

R

GRAFICI: 3-D

- Si possono usare insieme le funzioni viste per ottenere grafici complessi

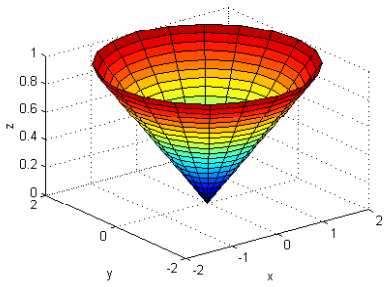
```
>>a=peaks(33);
>> pcolor(a)
>> shading interp
>> axis square
>> hold on
>> contour(a,'r')
>> surf(a+15)
>> view(-57,22)
```



SOLIDI DI ROTAZIONE

- La funzione *cylinder(r)* disegna la superficie di rotazione che ha per generatrice la curva descritta dal vettore *r*: per esempio disegnare il cono generato dalla rotazione della retta $y = x,\ x \in [0,2]$

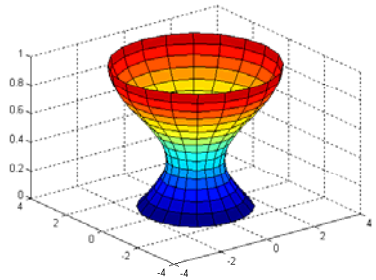
```
>>x=0:0.1:2;
>>y=x;
>>cylinder(y)
>>xlabel('x')
>>ylabel('y')
>>zlabel('z')
```

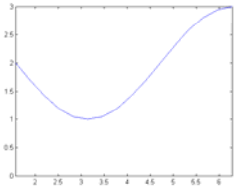


SOLIDI DI ROTAZIONE

- Si può usare una curva per descrivere il profilo

```
>>t = pi/2:pi/10:2*pi;
>>y= 2+cos(t);
>>cylinder(y)
```



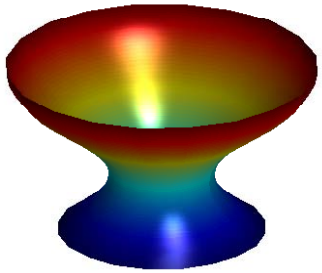


```
>>plot(t,y)
>>axis([pi/2 2*pi 0 3]);
```

SOLIDI DI ROTAZIONE

- Vi sono molte funzioni per manipolare la visualizzazione dei solidi (materiale, luce, angolo di vista)

```
>>axis off
>>shading interp
>>material metal
>>lightangle(45,30)
>>lighting phong
>>view([26 38])
```



HANDLE GRAPHICS

- MATLAB fornisce un insieme di funzioni a basso livello per manipolare elementi grafici: questo sistema è chiamato Handle Graphics
- Gli oggetti grafici sono le primitive di tale sistema
- Ci sono 11 tipi di oggetti Handle Graphics, organizzati in una struttura gerarchica
- Per creare un oggetto è necessario richiamare la funzione corrispondente: *figure, axes, line ...*

GUI

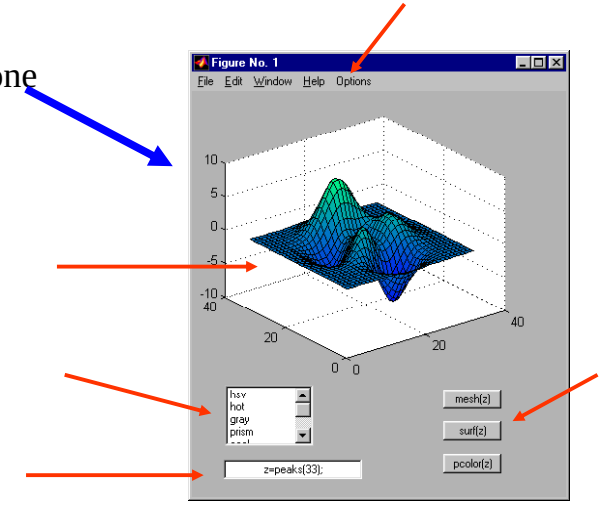
È possibile sviluppare in MATLAB strumenti basati su Graphical User Interface (GUI)

- I principi di un buon progetto di GUI sono generali, anche se si utilizzano strumenti specifici
- In particolare in MATLAB si sviluppa una GUI attraverso Guide (Graphical User Interface Development Environment)

GUI: ESEMPIO

- Esempio di un'applicazione completa.

>>guide



STAMPA DEI GRAFICI

- Con l'opzione Print dal menu File
- Importare in altra applicazione con Copy Figure dal menu Edit
- Salvare in diversi formati con print

24-bit RGB TIFF with packbits compression

```
>>print  -dtiff   nome_file
>>print   -depsc2 nome_file
```

Level 2 color Encapsulated PostScript

STRUTTURE DATI

Vediamo alcune strutture dati diverse dalle matrici

- Caratteri e testo
- Cell array
- Strutture
- Array multidimensionali
- Classi

CARATTERI E TESTO

- Per inserire testo si usano gli apici ' '

```
>>s='Ciao'
s =
  Ciao
>>whos s
  Name      Size          Bytes  Class
  s         1x4               8  char array
Grand total is 4 elements using 8 bytes
```

- È un vettore di caratteri

CARATTERI E TESTO

- Il testo è memorizzato come interi (codice ASCII)

```
>>a=double(s)
a =
    67   105    97   111
>>a(1,1)=99;
>>s=char(a)
s =
  ciao
```

Funzioni di conversione

CARATTERI E TESTO

- Si possono concatenare stringhe

```
>>a=[s '  mondo']
a =
  ciao mondo
>>a=[s ;'  mondo']
???  All rows in the bracketed expression must have the same
number of columns.
```


CARATTERI E TESTO

- Le righe di una matrice devono avere la stessa dimensione: si usa la funzione char()

```
>>s=char('Queste','sono' ,'prove')
s =
Queste
sono
prove
>> size(s)
ans =
    3     6
```


CARATTERI E TESTO

- Si possono convertire stringhe in numeri e viceversa

```
>> a=str2num('123')
a =  123
>> s=num2str(345)
s =  345
>>whos
  Name      Size          Bytes  Class
  a         1x1               8  double array
  s         1x3               6  char array
Grand total is 4 elements using 14 bytes
```


CONTROLLO FLUSSO DATI

- In MATLAB si usano 5 costrutti
  - if
  - switch
  - for
  - while
  - break / return

CONTROLLO FLUSSO DATI: if

```
>>x=1.2;
>>if  x < 0
>>y = -1;
>>elseif  x > 0
>>y = 1;
>>else
>>y = 0;
>>end
>>y
y =
    1
```

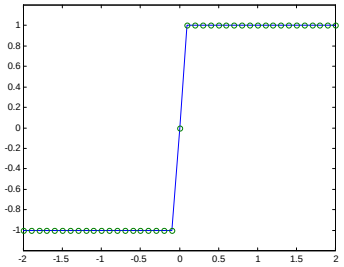
- Ogni istruzione if è terminata da un end
- Non sono necessarie parentesi
- Gli operatori relazionali agiscono sugli array elemento per elemento, quindi per lavorare con array utilizzare le funzioni is* (forniscono lo stato)

CONTROLLO FLUSSO DATI: if

```
>>x=[1.2 1.2];
>>y=[1.2 1.2];
>>x==y
ans =
    1    1

>>isequal(x,y)
ans =
    1
```

```
>>x=-2:0.1:2;   y=zeros(size(x));
>>y =y -1*(x<0);   y =y+ 1*(x>0);
>>y =y+ 0*(x==0);
>>plot(x,y,x,y,'o'),   axis([-2 2 -1.2 1.2])
```



CONTROLLO FLUSSO DATI : switch

```
>>ii=1;
>>switch ( ii )
>>case 0
>>disp('Opzione  0');
>>case 1
>>disp('Opzione  1');
>>otherwise
>>disp('Opzione ???');
>>end
 Opzione  1
```

- Ogni istruzione switch è terminata con end
- Il valore di default è trattato con otherwise
- Quando un case è vero, gli altri sono saltati: non è necessario break.
- La funzione disp permette di visualizzare testi o array

CONTROLLO FLUSSO DATI : for

```
>>for ii=1:3
>>for jj=1:3
>>a(ii,jj)=(ii-1)*3 +jj;
>>end
>>end
>>a
a =
    1    2    3
    4    5    6
    7    8    9
```

- Ogni for è terminato con un end
- Non sono necessarie parentesi
- Si può utilizzare un incremento qualsiasi for ii = 1 : -0.3 : -2, end

N.B In MATLAB è meglio utilizzare codice vettorizzato

CONTROLLO FLUSSO DATI : while

```
>>b=10;  a= -3;
>>while b-a > 0
>>b=b-1;
>>end
>>b
b =
    -3
```

- Ogni while è terminato con un end
- Non sono necessarie parentesi

CONTROLLO FLUSSO DATI : break

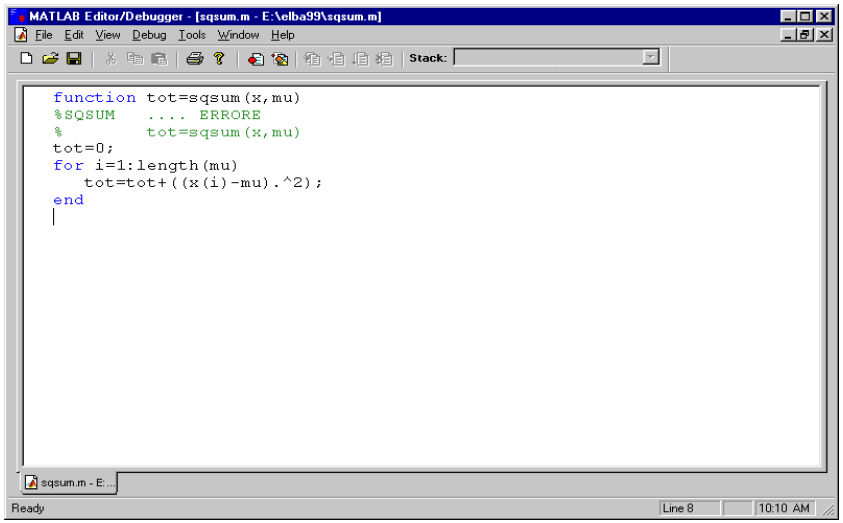
```
>>b=10;  a= -3;
>>while b-a > 0
>>if b<=0
>>break
>>end
 >>b=b-1;
>>end
>>b
b =
    0
```

- break permette di uscire dai loop for e while
- Permette di uscire solo dal loop più interno
- return permette di tornare alla funzione chiamante

M-file

- File che contengono codice in linguaggio MATLAB sono chiamati M-file.
- Devono avere estensione .m
- Sono file di testo: possono essere creati con qualsiasi editor ma è preferibile usare quello built-in
- Evidenzia con colori diversi la sintassi e contiene un debugger

M-file



M-file

Script	Function
• Non ha argomenti di input ed output	• Ha argomenti di input ed output
• Opera sui dati nello workspace	• Le variabili interne sono locali
• Utile per automatizzare una serie di passi ripetitivi	• Utile per estendere le funzionalità di MATLAB

M-file: SCRIPT

Nome del file → script_2.m

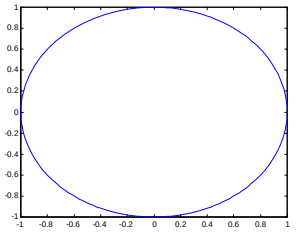
Linea di commento → *%Esempio di script*

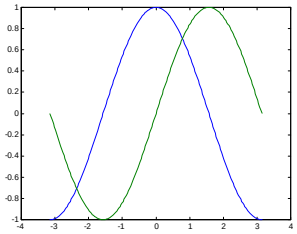
Calcoli →
```
x=-pi:pi/100:pi;
ys=sin(x);
yc=cos(x);
```

Controllo flusso dati e visualizzazione →
```
if stato
   plot(x,yc,x,ys)
else
   plot(yc,ys)
end
```


M-file: SCRIPT

```
>>script_2
??? Undefined function or variable
    'stato'.
Error in ==> e:\elba99\script_2.m
On line 5  ==> if stato
>>stato=0;
>>script_2
>>stato=1;
>>script_2
>>who
Your variables are:
stato    x        yc       ys
```





M-file: FUNCTION

Nome del file → function_2.m

Definizione di funz. →
```
function ys = function_2(stato)
```

Linea di help1 →
```
%FUNCTION_2  Esempio
```

Testo di help →
```
%          ys = function_2(stato) : ...
```

Corpo della funzione:

Calcoli →
```
x=-pi:pi/100:pi;
ys=sin(x);
yc=cos(x);
```

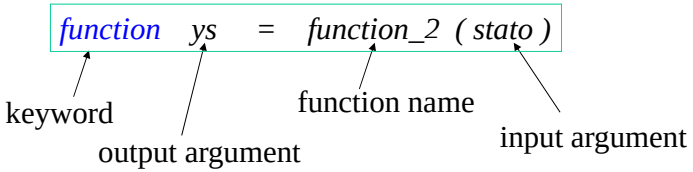
Controllo flusso visualizzazione →
```
if stato
   plot(x,yc,x,ys)
else
   plot(yc,ys)
end
```

M-file: FUNCTION

```
>>script_2
??? Undefined function or variable
    'stato'.
Error in ==> e:\elba99\script_2.m
On line 5  ==> if stato
>>stato=0;
>>script_2
>>stato=1;
>>script_2
>>who
Your variables are:
stato    x        yc       ys
```

```
>>function_2
??? Input argument 'stato' is undefined.
Error in ==> e:\elba99\function_2.m
On line 7  ==> if stato
>> stato=0;
>>function_2
??? Input argument 'stato' is undefined.
Error in ==> e:\elba99\function_2.m
On line 7  ==> if stato
>>a=function_2(0);
>>a=function_2(1);
>> who
Your variables are:
a        stato
```


M-file: FUNCTION



- Se si hanno molti argomenti
  function [x,y,z] = sphere(theta,phi,rho)

- Se non si hanno argomenti di output
  function sphere(theta,phi,rho)

M-file: FUNCTION

- Quando si chiama una funzione, MATLAB esegue il parsing del codice e produce uno pseudocodice, che alloca in memoria

- Ogni funzione ha un proprio workspace

- Ma possono essere definite variabili globali

M-file: VARIABILI GLOBALI

```
>>global ALPHA
>>ALPHA=3;
>>myp_2([1 2 3])
ans =
   1    8    27
>> ALPHA=2.3;
>>myp_2([1 2 3])
ans =
   1.0000    4.9246   12.5135
```

myp_2.m

```
function y = myp_2(x)
%MYP_2    ....
%         .....
global ALPHA
y=x.^ALPHA;
```

M-file: USER INPUT

- Per ottenere un input da utente durante l'esecuzione di un M-file
  - Visualizzare un prompt e attendere un input (input)
  - Attendere la pressione di un tasto (pause)
  - Costruire una GUI

M-file: USER INPUT

```
>>myp_a_2([1 2 3])
Inserisci l`esponente:3
ans =
    1     8    27
```

myp_a_2.m

```
function y = myp_a_2(x)
%MYP_A_2    ....
%           .....
n=input('Inserisci l`esponente:');
y=x.^n;
```

```
>>myp_a_2([1 2 3])
Inserisci l`esponente:ciao
ans =
ciao
```

```
n=input('Inserisci l`esponente:','s');
y=s;
```


M-file: VALUTAZIONE DI STRINGHE

- La funzione eval permette di eseguire una stringa

```
>>eval('a=[1 2 3].^2')
a =
    1     4     9
```

```
>>for ii=1:3
eval([ 'p' , int2str(ii) , '=ii.^2' ])
end

p1 =
     1
p2 =
     4
p3 =
     9
```


M-file: OTTIMIZZAZIONE

- Principalmente ci sono due tecniche
- Vettorizzazione dei loop: cioè sostituire for e while con equivalenti operazioni matriciali
- Preallocazione degli array: allocare la memoria prima di utilizzare gli array

M-file: OTTIMIZZAZIONE

```
>>mybench_2
elapsed_time =
  204.1000
elapsed_time =
    0.0500
```

mybench_2.m

```
%esempio
N=30000;   tic
i=0;  c=zeros(1,N);
for t=0:pi/N:2*pi
   i=i+1;
   c(i)=sin(t);
end
toc,  tic
t=0:pi/N:2*pi;
c=sin(t);  toc
```


M-file: OTTIMIZZAZIONE

```
>>mybench_1_2
elapsed_time =
   5.4300

elapsed_time =
  14.7800
```

Mybench_1_2.m

```
%esempio
N=500;  tic,   a=zeros(N);
for ii=1:N
   for jj=1:N
      a(ii,jj)=ii+jj;
   end
end
toc,  tic
for ii=1:N
   for jj=1:N
      b(ii,jj)=ii+jj;
   end
end,  toc
```


DEBUGGING

Il debugging è il processo che permette di trovare errori nel proprio codice

- Errori di sintassi: li indica MATLAB al prompt
- Errori runtime: errori algoritmici, quindi rilevabili solo da risultati inattesi, perchè interni allo workspace delle funzioni

DEBUGGING

Per isolare gli errori runtime

- Togliere selettivamente i ;
- Rendere la funzione uno script
- Usare l'istruzione keyboard
- Usare il MATLAB debugger

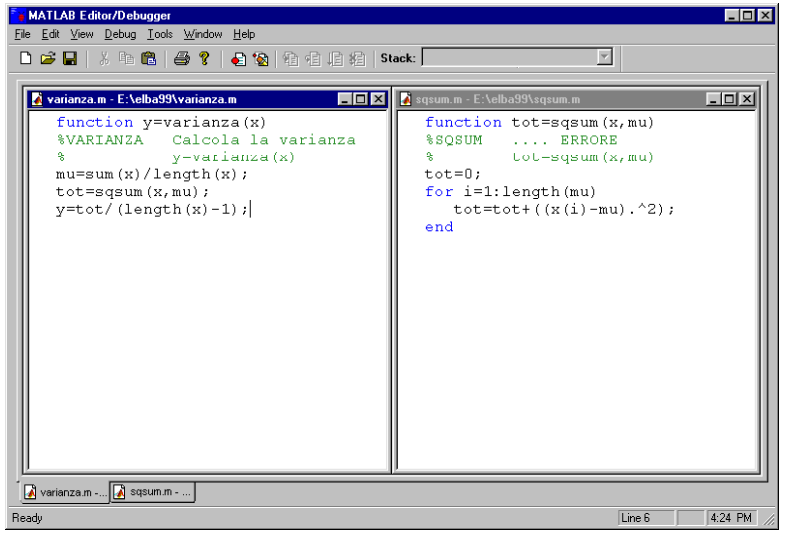
DEBUGGING

```
>>who
>>niente
K» who
Your variables are:
x        y
K» y
y =  4
K» x=3;
K» return
y =   9
>>
```

niente.m

```
function niente()
%       ....
%       ....
x=2;
y=x.^2;
keyboard
y=x.^2
```

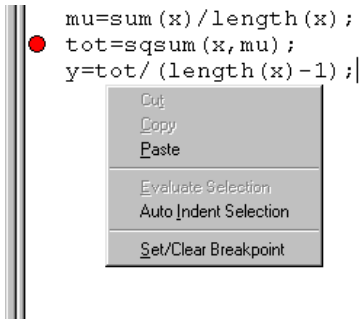

DEBUGGING



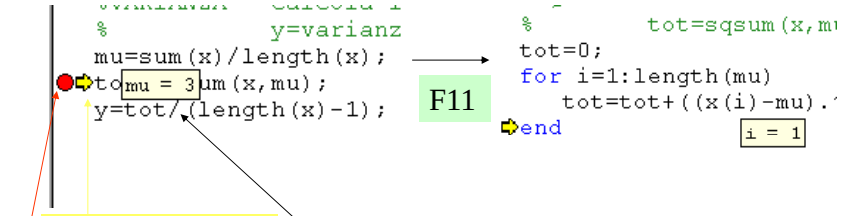
DEBUGGING

```
>> v=[1 2 3 4 5];
>>var1=std(v).^2
var1 =
   2.5000

>>var2=varianza(v)
var2 =
    1
```



DEBUGGING



F11 Step in

F10 Single step

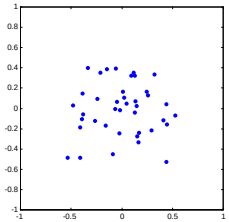
APPLICAZIONI

- Vediamo come MATLAB lavora con particolari tipi di dati e funzioni
  - animazioni
  - I/O dati
  - funzioni di funzione

APPLICAZIONI: ANIMAZIONI

anim_2.m

```
%Brownian motion
n=40; s=.02;
x=rand(n,1)-0.5;  y=rand(n,1)-0.5;
h=plot(x,y,'.');
axis([-1 1 -1 1]),  axis square,  grid off
set(h,'EraseMode','xor','MarkerSize',18)
while 1
  x=x + s*randn(n,1);
  y=y + s*randn(n,1);
  set(h,'XData',x,'YData',y)
  drawnow
end
```



- Adatta per lunghe sequenze di semplici plot
- Per fermare la simulazione digitare <ctrl>-c

I/O

- Oltre alle funzioni save e load MATLAB fornisce funzioni di I/O C-like:
  - fopen, fclose : apertura e chiusura file
  - fscanf, fprintf: lettura e scrittura dati formattati
  - fread, fwrite: lettura e scrittura binaria di dati

I/O FORMATTATO

```
%...
x = 0:.1:1;y = [x; exp(x)];
fid =fopen('exp.txt','w');
fprintf(fid,'%6.2f%12.8f\n',y);
fclose(fid)
```

File su disco

exp.txt

```
0.00     1.00000000
0.10     1.10517092
…
1.00     2.71828183
```

```
%...
fid = fopen('exp.txt');
a = fscanf(fid,'%g %g',[2 inf]);
   % Ora sono due righe
a = a';
fclose(fid)
```

```
>>a
a = 0.00     1.00000000
    0.10     1.10517092
    …
    1.00     2.71828183
```

I/O BINARIO

```
%...
a=[1 2 3 4; 5 6 7 8];
fid =fopen('a.bin','wb');
fwrite(fid,a,'float');
fclose(fid)
```

```
%...
fid =fopen('a.bin','rb');
a=fread(fid,[2,4],'float');
fclose(fid)
```

a.bin

File su disco

```
>>a
a =
   1   2   3   4
   5   6   7   8
```

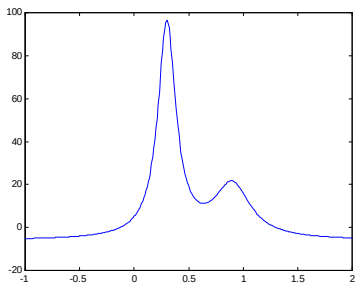

FUNZIONI DI FUNZIONE

Consideriamo un esempio che può essere generalizzato come modello di interfaccia a particolari funzioni di MATLAB

- Integrazione numerica
- Minimo di una funzione
- Equazioni differenziali ordinarie

FUNZIONI DI FUNZIONE

```
>>x=-1:.01:2;
>>plot(x,es_2(x))
```



es_2.m

```
function y=es_2(x)
%ES_2    Esempio
%        y=es_2(x)
y=1./((x -.3).^2 + .01)+...
   1./((x - .9).^2 + .04) -6;
```


INTEGRAZIONE NUMERICA

```
>>quad('es_2',0,1)
ans =
   29.8583

>>quad8('es_2',-10,10)
ans =
  -73.2779
```

Stringa contenente il nome di una funzione

API

- Sebbene MATLAB è un ambiente completo per programmare, esiste la possibilità di interfacciare MATLAB con programmi esterni attraverso la Application Program Interface (API)
- Si può
  - utilizzare MATLAB da programmi C, MATLAB engine
  - utilizzare proprie applicazioni C come funzioni buil-in, MEX-file