Breve introduzione all'uso di Matlab

M.I. Gualtieri, A. Napoli

Dipartimento di Matematica, Università della Calabria - ITALY

Introduzione

Il nome **Matlab** deriva da **Mat**(rix) **Lab**(oratory)

Originariamente il **Matlab** è stato sviluppato come ambiente interattivo per il calcolo matriciale ad alto livello.

Attualmente è utilizzato anche come

- calcolatrice elettronica evoluta
- ambiente grafico
- linguaggio di programmazione.

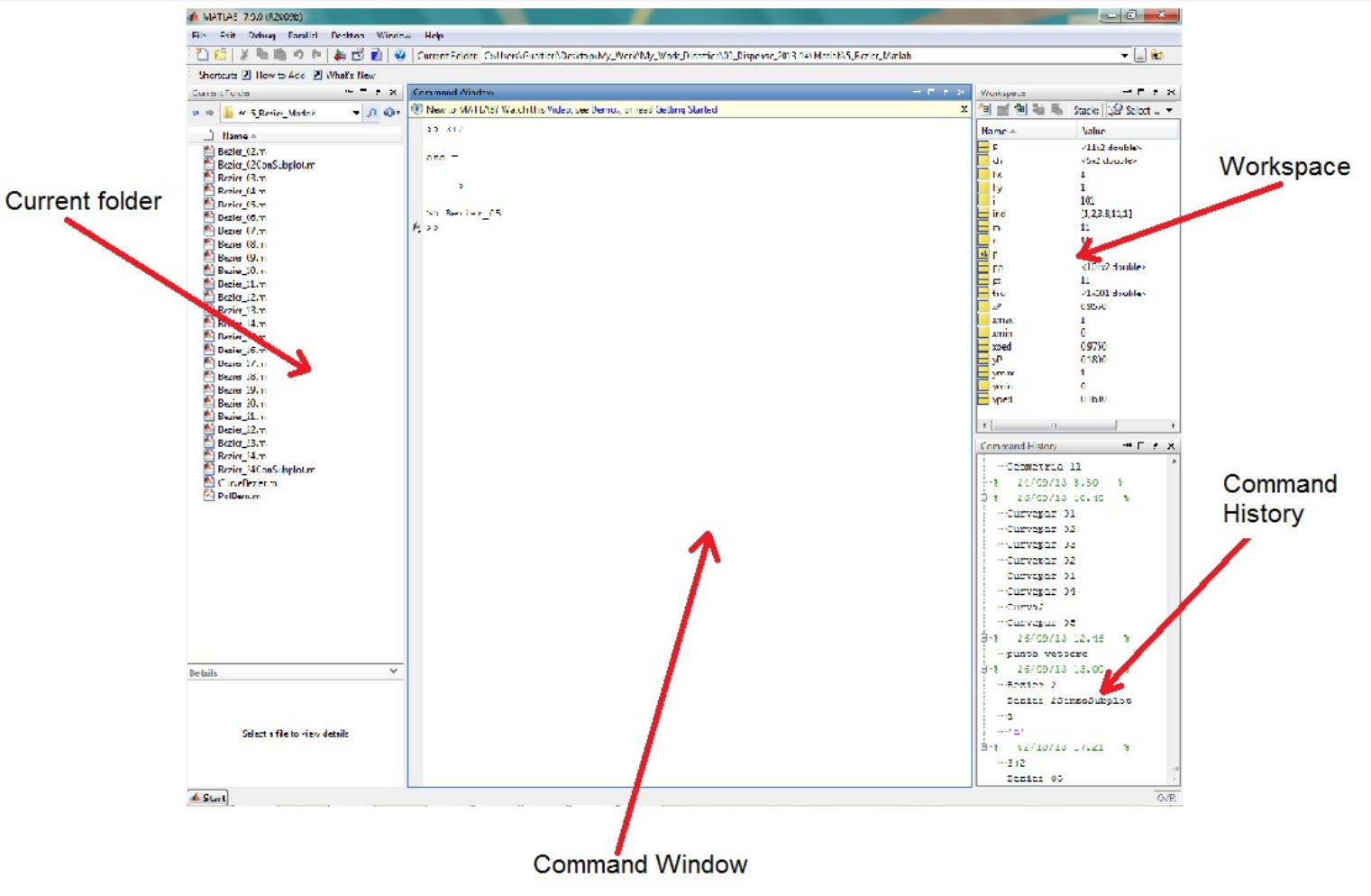
Matlab fa largo uso di librerie di calcolo, in particolar modo di algebra lineare, ampiamente diffuse e validate nell'ambito scientifico.

Desktop **Matlab**

- Command Window (finestra dei comandi): permette all'utente di comunicare con **Matlab**. In questa finestra si digitano i comandi e le istruzioni da eseguire;
- Command History (cronologia dei comandi): mostra tutti i comandi immessi in precedenza nella finestra dei comandi;
- Current folder (directory corrente): elenca i file presenti nella programmi directory di lavoro
- Workspace: elenca le variabili usate e lo spazio utilizzato.

Nella parte superiore del desktop: nome dei menu
barra degli strumenti

Desktop Matlab



Introduzione di comandi e dati

I comandi vengono introdotti in maniera diretta, ossia scritti direttamente sulla linea di comando dopo il prompt (>>) e seguiti dal carattere di invio (return) (↩),

```
>> a = 3
 a =
      3
```

;	al termine di un'istruzione impedisce la visualizzazione dell'output;
%	consente di introdurre un commento alla sua destra;
...	consente di estendere un'istruzione alla riga successiva.

Operazioni elementari

Si usano i simboli standard per le operazioni aritmetiche elementari di addizione (+), sottrazione (-), moltiplicazione (*), divisione (/) più l'operatore di elevamento a potenza (^) e la divisione inversa o a sinistra (\).

```
>> 7-2
ans
     5
```

```
>> b=7-2
b=
     5
```

Durante la sessione di lavoro è possibile richiamare i comandi digitati in precedenza attraverso l'uso delle frecce della tastiera.

Gestione dei numeri complessi

Matlab gestisce in modo automatico l'algebra dei numeri complessi

```
>> c=2+3*i;d=-3+2*i;
>> c+d;
ans =
   -1.0000 + 5.0000i
>> (2+3*i)'                % operazione di coniugazione
ans =
   2.0000 - 3.0000i
>> sqrt(-4)
ans =
   0 + 2.0000i
```

Gestione dei numeri complessi

In **Matlab** il risultato dell'operazione di elevamento a potenza nel caso di esponenti frazionari può non corrispondere a quello naturalmente atteso.
Ad esempio, volendo calcolare la radice cubica di -5, si ottiene

```
>> (-5)^(1/3)
ans =
   0.8550 + 1.4809i
```

e non un numero reale.

L'apparente anomalia è dovuta al fatto che **Matlab**, lavorando in notazione complessa, calcola come prima radice cubica di -5 una delle due con parte immaginaria non nulla.

Help on line

Il comando `help` consente di avere una descrizione immediata di ogni componente **Matlab** (funzione, comando, operatore, ...).

Senza ulteriori indicazioni, `help` visualizza tutti i `toolbox` disponibili, essendo il generico toolbox un pacchetto dedicato ad una specifica applicazione
(es. `polyfun` tratta approssimazione e interpolazione di funzioni).

Per avere un'informazione dettagliata, è sufficiente far seguire il comando `help` dal nome del comando (o del toolbox) desiderato.

Esempio (help) (1)

```
>> help polyfun
```

Polynomial and interpolation functions

Polynomials

roots	Find polynomial roots
poly	Construct polynomial with specified roots
polyval	Evaluate polynomial
polyvalm	Evaluate polynomial with matrix argument
residue	Partial-fraction expansion (residues)
polyfit	Fit polynomial to data
polyder	Differentiate polynomial
conv	Multiply polynomials
deconv	Divide polynomials

Esempio (`help`) (2)

Data interpolation

`interp1`	1-D interpolation (1-D table lookup)
`interp2`	2-D interpolation (2-D table lookup)
`interpft`	1-D interpolation using FFT method
`griddata`	Data gridding

Spline interpolation

`Spline`	Cubic spline data interpolation
`ppval`	Evaluate piecewise polynomial

Esempio (help) (3)

```
>> help spline
```

Spline Cubic spline data interpolation

Given data vectors X and Y, and a new abscissa vector XI, the function YI = SPLINE(X,Y,XI) uses cubic spline interpolation a vector YI corresponding to XI.

Here's an example that generates a coarse sine curve, then interpolates over a finer abscissa:

```
x=0:10;y=sin(x);
xi=0:.25:10;
yi=spline(x,y,xi);
plot(X,Y,'o',xi,yi)
```

`PP = spline(x,y)` returns the pp-form of the cubic spline interpolant instead, for later use with ppval, etc.

See also INTERP1, INTERP2, ..., the Spline Toolbox.

Le espressioni in Matlab

Matlab opera per espressioni, ovvero è realizzato in modo tale da convertire espressioni in variabili.

In particolare, **Matlab** lavora di default con oggetti, classificati come matrici, che possono essere:

- Matrici 1×1 cioè scalari;
- Matrici $1 \times n$ cioè vettori riga;
- Matrici $n \times 1$ cioè vettori colonna;
- Matrici $n \times m$ cioè matrici in senso proprio.

Le espressioni in Matlab

Le espressioni possono coinvolgere più tipi, quali, ad esempio:

- operatori $(+,\ -,\ *,\ /,\ \backslash,\ \ldots)$;
- caratteri speciali $(\%, !, :, ;, \ldots)$;
- funzioni (`help, spline, ...`);
- nomi di variabili.

Le variabili in Matlab

Matlab NON richiede la dichiarazione esplicita del tipo (intero, reale, complesso) e della dimensione delle variabili utilizzate.

I nomi delle variabili sono costituiti da

- caratteri alfabetici maiuscoli e minuscoli
- cifre
- _ (*underscore*)

con i vincoli

- il primo carattere deve essere un carattere alfabetico
- la lunghezza del nome non deve superare i 31 caratteri.

Le variabili in Matlab

Matlab è case sensitive, cioè distingue tra lettere maiuscole e lettere minuscole.

Per modificare tale caratteristica basta digitare il comando

```
>> cases off
```

per abilitarla nuovamente

```
>> cases on
```

Variabili predefinite in Matlab

`eps`	precisione macchina $\approx 2.22 * 10^{-16}$
`pi`	π
`i, j`	$\sqrt{-1}$
`realmax`	$\approx 1.798 * 10^{308}$ massimo numero di macchina positivo
`realmin`	$\approx 2.225 * 10^{-308}$ minimo numero di macchina positivo
`Inf`	∞ (numero $>$ `realmax`, risultato di una divisione per 0, $\log 0, \ldots$)
`NaN`	Not a Number $(0/0, Inf/Inf, \ldots)$

Le variabili predefinite possono essere modificate dall'utente durante la sessione di lavoro.

Funzioni predefinite in Matlab

Matlab mette a disposizione dell'utente un certo numero di funzioni matematiche predefinite. Molte di esse possono essere applicate sia a variabili scalari che matriciali.

`round(x)`	arrotondamento: $x = 3.6 \Rightarrow round(x) = 4$;
`fix(x)`	troncamento: $x = 3.6 \Rightarrow fix(x) = 3$;
`sign(x)`	segno di x (vale 1, 0 o -1);
`rem`	resto nella divisione intera;
`real(z)`	parte reale di z;
`imag(z)`	parte immaginaria di z;
`conj(z)`	z' (complesso coniugato di z);

Funzioni predefinite in Matlab

`sin(x)`	funzione seno;
`cos(x)`	funzione coseno;
`tan(x)`	funzione tangente;
`sinh(x)`	funzione seno iperbolico ($\sinh x$);
`cosh(x)`	funzione coseno iperbolico ($\cosh x$);
`tanh(x)`	funzione tangente iperbolica ($\tanh x$);
`asin(x)`	funzione arcoseno ($\arcsin x$);
`acos(x)`	funzione arcocoseno ($\arccos x$);
`atan(x)`	funzione arcotangente ($\arctan x$).

Funzioni predefinite in Matlab

`sqrt(x)`	$\sqrt{x}$;
`abs(x)`	$\lvert x \rvert$ (valore assoluto);
`exp(x)`	e^x;
`log(x)`	$\ln x$ (logaritmo naturale di x);
`log10(x)`	$\log x$ (logaritmo in base 10 di x);
`log2(x)`	$\log_2 x$ (logaritmo in base 2 di x).

Per una lista più ampia `help elfun`.

Assegnazione delle variabili

L'assegnazione di uno scalare viene effettuata semplicemente ponendo il nome della variabile uguale al valore prescelto

```
>> nome_variabile=1.54
nome_variabile= 1.5400
```

Se il nome della variabile viene omesso, **Matlab** crea automaticamente la variabile `ans` (answer, risposta) contenente il valore assegnato

```
>> 1.54
ans= 1.5400
```

Nel caso si termini il comando con il carattere `;` l'output viene soppresso.

Gestione del Workspace (variabili)

`who`	elenca tutte le variabili definite durante la sessione di lavoro;
`whos`	elenca le funzioni e il loro tipo;
`clear` *nome_vars*	cancella la variabile o le variabili specificate;
`clear all`	cancella tutte le variabili dell'area di lavoro.

Il comando who

Il comando who elenca tutte le variabili definite durante la sessione di lavoro.

```
>> s=5;
>> who
Your variables are:
s
>> t=3;
>> who
Your variables are:
s   t
```

Il comando whos

Il comando whos dà informazioni sulla quantità di memoria allocata

```
>> s = 10;
>> u = 'ciao'
>> whos
```

Name	Size	Bytes	Class
s	1×1	8	double array
u	1×4	8	char array

```
Grand total is 5 elements using 16 bytes
```

Il comando `whos`

- **Name**: nome della variabile
- **Size**: numero di righe e colonne
- **Bytes**: occupazione della memoria
- **Class**: classe o tipo della variabile

  Può assumere valore double, sparse, struct, char, ...

  (`help class` per una trattazione più dettagliata)

Si può osservare che una variabile di tipo char array è un vettore riga ad n componenti, ciascuna delle quali occupa uno spazio di 2 bytes.

Il comando `clear`

Matlab memorizza tutte le variabili definite nel corso della sessione di lavoro, quindi si rischia facilmente di saturare rapidamente la memoria disponibile; in tal caso può comparire un messaggio di errore del tipo `out of memory`.

Per ovviare a ciò è opportuno controllare periodicamente la quantità di memoria occupata tramite il comando `whos` ed, eventualmente, cancellare le variabili inutili.

Il comando `clear`

Per cancellare una o più variabili e liberare le corrispondenti aree di memoria, basta utilizzare il comando `clear`, seguito dai nomi delle variabili da "*ripulire*".

Per cancellare tutte le variabili definite dall'utente si usa il comando `clear` semplicemente.

Il comando `clear` è usato anche per ripristinare i valori di default delle variabili predefinite ad eccezione di `eps` per la quale è necessario riavviare **Matlab**.

Output

Matlab lavora generalmente con 14 cifre decimali.

Un intero viene presentato in formato privo di punto decimale

```
>> nome_variabile=2
nome_variabile= 2
```

Un decimale viene presentato con solo 4 cifre significative (`format short`)

```
>> nome_variabile=1.322356
nome_variabile= 1.3224
```

Si effettua, solo a livello di output, un arrotondamento alla quarta cifra decimale.

Output

È possibile visualizzare le restanti cifre decimali, attivando `format long`.

In questo modo sono mostrate tutte le 14 cifre decimali impiegate da **Matlab**.

```
>> nome_variabile=1.322356
nome_variabile= 1.3224

>> format long
>> nome_variabile
nome_variabile= 1.32235600000000
```

Format

```
>> help format
```

FORMAT Set output format.

All computations in **Matlab** are done in double precision.
FORMAT may be used to switch between different output
display formats as follows:

FORMAT	Default. Same as SHORT.
FORMAT SHORT	Scaled fixed point format with 5 digits.
FORMAT LONG	Scaled fixed point format with 15 digits.
FORMAT SHORT E	Floating point format with 5 digits.
FORMAT LONG E	Floating point format with 15 digits.
FORMAT SHORT G	Best of fixed or floating point format with 5 digits.
FORMAT LONG G	Best of fixed or floating point format with 15 digits.
FORMAT HEX	Hexadecimal format.

Format

FORMAT +	The symbols +, - and blank are printed for positive, negative and zero elements. Imaginary parts are ignored.
FORMAT BANK	Fixed format for dollars and cents.
FORMAT RAT	Approximation by ratio of small integers.

Spacing:

FORMAT COMPACT	Suppress extra line-feeds.
FORMAT LOOSE	Puts the extra line-feeds back in.

La modifica della visualizzazione di un risultato tramite `format` non ha nulla a che vedere con l'effettiva precisione con cui **Matlab** effettua il calcolo.

Format

- `short` *virgola fissa con 4 cifre decimali (formato di default)*

```
>> pi
ans =
   3.1415
```

- `long` *virgola fissa con 15 cifre decimali*

```
>> format long
>> pi
ans =
   3.14159265358979
```

Format

- `short e` *virgola mobile con 4 cifre decimali*

```
>> format short e
>> pi
ans =
   3.1416e+00
```

- `long e` *virgola mobile con 15 cifre decimali*

```
>> format long e
>> pi
ans =
3.14159265358979e+00
```

Format

- `rat` *razionale (rapporto di due numeri interi)*

```
>> format rat
>> pi
ans =
   355/113
```

```
>> 6/4
ans =
   3/2
```

- `bank` *formato per operazioni bancarie (solo due cifre decimali)*

```
>> format bank
>> pi
ans =
   3.14
```

Array (Assegnazione)

Per assegnare le componenti di un array (matrice o vettore) si ricorre ai caratteri speciali [ ] (o anche ( ) nel caso dei soli vettori) che rappresentano i delimitatori di tali variabili.

I singoli elementi saranno separati dal carattere

, virgola (elementi sulla stessa riga)

o da

; punto e virgola (elementi su righe diverse).

Array (Assegnazione)

```
>> r=[1, 2, 3, 4]          % (vettore riga)
>> r=[1 2 3 4]             % (vettore riga)
r =
    1    2    3    4
```

L'uso delle virgole all'interno della dichiarazione di matrice (o di vettore) è facoltativo.

```
>> c=[1; 2; 3]             % (vettore colonna)
>> c=[1                    % (vettore colonna)
        2
        3]
>> c=[1, 2, 3]'            % (vettore colonna)
c= 1
    2

    3
```

Array (Assegnazione)

```
>> A=[1, 2; 3, 4]                % (matrice quadrata 2 × 2)
A=
    1    2
    3    4
>> B=[1, 2, 3; 4, 5, 6]          % (matrice rettangolare 2 × 3)
B=
    1    2    3
    4    5    6
>> B=[1 2 3                      % (matrice rettangolare 2 × 3)
       4 5 6]
B=
    1    2    3

    4    5    6
```

Array (Assegnazione)

Vettori di elementi ordinati equispaziati possono essere costruiti automaticamente specificando la prima componente, l'ultima e

- l'**incremento** `step`

  ```
  >> vet=[vi:step:vf]
  ```

  se `step` non è specificato, è assunto per default pari ad **1**;

- il **numero di componenti** `nc`

  ```
  >> vet=linspace(vi,vf,nc)
  ```

  se `nc` non è specificato, è assunto per default pari a **100**

  se `nc<2` viene restituito un vettore di una sola componente uguale a `vf`

Array (Assegnazione)

Riepilogo dei comandi di assegnazione per vettori

Vettori riga

```
>> x=[1 2 3 4]
>> x=[1:5]
>> x=1:5
>> x=[1:.1:5]
>> x=[5:-1:1]
>> x=linspace(a,b)
>> x=linspace(a,b,n)
```

Vettori colonna

```
>> x=[1; 2; 3]
>> x=[1
      2
      3]
>> x=[1 2 3]'
```

Matrice vuota

Per creare una matrice vuota

```
>> A=[ ]
```

Permette di rimuovere righe e/o colonne di una matrice. Ad esempio, per eliminare la seconda colonna della matrice 2×3

```
>> A=[1,2,3;4,5,6]
A= 1   2   3
     4   5   6
```

si scriverà

```
>> A(:,2)=[ ]
A= 1   3
     4   6
```

Il carattere `:` è usato per indicare tutti gli elementi di una riga o di una colonna.

Dimensione di una matrice

Il comando `size` restituisce le dimensioni di una matrice. L'output può essere memorizzato in un vettore

```
>> A=[1,2,3,4;5,6,7,8];
```

```
>> size(A)
ans=
   2   4
```

```
>> [n,m]=size (A)
n=
   2
m=
   4
```

```
>> n=size(A,1),m=size(A,2)
n=
   2
m=
   4
```

Il comando `length` restituisce la dimensione di una vettore.

```
>> b=[1,2,3,4];
```

```
>> length(b)
ans=
   4
```

`length` applicato ad una matrice restituisce la dim. più grande della matrice.

```
>> length(A)
ans=
   4
```

Manipolazione di matrici

```
>>   c=[1 2 3;4 5 6]        (assegnazione di matrice)
c=
     1   2   3
     4   5   6

>>   d=c(2,3)               (elemento di posto (2,3))
d=
     6

>>   d=c(1,:)               (estrazione della prima riga)
d=
     1   2   3
```

Manipolazione di matrici

```
>>   d=c(:,2)
d=
     2   5
```

(*estrazione della seconda colonna*)

```
>>   d=c(:,1:2)
d=
     1   2
     4   5
```

(*estrazione delle prime due colonne*)

```
>>   d=c(:)
d=
     1   2   3   4   5   6
```

(*trasferisce gli elementi di* `c` *in un vettore* `d`)

Manipolazione di matrici

Nei comandi precedenti si noti l'uso del carattere speciale `:`.

Esso rappresenta la versione più semplice del comando

```
vet=[vi:step:vf]
```

in cui

- `vi`=1
- `vf`=dimensione riga (o colonna)
- `step`=1.

Così il comando

```
>> d=c(:,1:2)
```

trasferisce in `d` le colonne 1 e 2 di `c` per tutta la loro lunghezza.

Manipolazione di matrici

Si possono costruire matrici combinando opportunamente altre matrici già assegnate.

```
>> c=[1 2 3;4 5 6];

>> d=[c;c]
d=
     1   2   3
     4   5   6
     1   2   3
     4   5   6
```

In tal modo `d` è costruita come una matrice a due righe, ciascuna delle quali è data dall'intera matrice `c`.

Operazioni con array

Le operazioni elementari si estendono naturalmente all'algebra dei vettori, con l'eccezione delle operazioni di divisione e di elevamento a potenza.

```
>> a=[1:4];                % (vettore [1 2 3 4])
>> b=[1:3];                % (vettore [1 2 3])
>> c=[3 2 6 -1];
>> a+c                     % (somma di vettori riga)
ans=
     4   4   9   3
>> a-c                     % (differenza di vettori riga)
ans=
     -2   0   -3   5
```

Operazioni con array

```
>> a+b
???  Error using ==> + Matrix dimension must agree
>> a*c
???  Error using ==> * Inner matrix dimension must agree
```

Le operazioni fra vettori possono essere effettuate solo se le dimensioni sono consistenti.

Dato un vettore riga `r` di dimensione n (ossia in **Matlab** una matrice $1 \times n$) ed un vettore colonna `c` di dimensione m (cioè una matrice $m \times 1$), il prodotto scalare `r*c` sarà eseguito solo se $m = n$ (producendo come risultato una matrice 1×1 ossia uno scalare).

Operazioni con array

```
>> r=[1:4];c=[1;2;3;4];  % (vettori riga [1 2 3 4]) e colonna [1 2 3 4]')
>> r*c                   % (prodotto scalare)
ans=
   30
```

Il risultato di `c*r` è una matrice di dimensione $m \times n$

```
>> r=[1:4];c=[1;2;3;4;5];  % (vettori riga [1 2 3 4]) e col. [1 2 3 4]')
>> M=c*r                  % (prodotto riga-colonna)
M=
    1   2   3   4
    2   4   6   8
    3   6   8   12
    4   8   12  16
    5   10  15  20
```

Operazioni con array

Vettori riga

Estrazione di una sottomatrice

```
>> A(1:2,:)
>> A(1:2,2:3)
>> A(:,2)
>> A(3,:)
```

Somma

```
>> A+B      % somma di matrici
>> A+c      % somma con uno scalare
```

Prodotto

```
>> A*B      % prodotto tra matrici
>> c*A      % prodotto per uno scalare
```

Divisione

```
>> X = B/A     % equivale a X = B*inv(A)
>> X = B\ A    % equivale a X = inv(B)*A
```

Divisione tra matrici (esempio)

Esempio

Dati `A` e `b`, risolvere il sistema `Ax=b`.

```
>> A=[2 6 10;6 10 14;10 14 6];
>> b=[44 68 56]';
>> x = A\b
x =
   1.0000
   2.0000
   3.0000
>> b-A*x
ans =
   0
   0
   0
```

Operazioni con array (Matrice trasposta)

L'operazione di trasposizione si effettua posponendo al nome della matrice il carattere '. Con i dati dell'esempio precedente

```
>> c'                    % (traspone in vettore riga)
ans=
   1  2  3  4  5
>> M'                    % (traspone la matrice)
ans=
    1   2   3    4    5
    2   4   6    8    10
    3   6   8    12   15
    4   8   12   16   20
```

Nel caso di matrici ad elementi complessi l'uso del carattere ' comporta anche l'operazione di coniugazione.

Operazioni con array (esempio)

Esempio

```
>> n=5;
>> x=linspace(0,2*pi,n);
>> c=cos(x);s=sin(x);
>> f=2*x.^2-1;
>> [x' c' s' f']
ans =
         0    1.0000         0   -1.0000
    1.5708    0.0000    1.0000    3.9348
    3.1416   -1.0000    0.0000   18.7392
    4.7124   -0.0000   -1.0000   43.4132
    6.2832    1.0000   -0.0000   77.9568
```

Operazioni con array (Matrice inversa)

L'operazione di inversione (per matrici quadrate non singolari) viene effettuata usando il comando `inv`.

```
>> A=[1 2;3 4];
>> inv(A)                    % (calcola l'inversa di A)
ans=
    -2.0000    1.0000
     1.5000   -0.5000
```

Operazioni con array (Matrice inversa)

Se la matrice è singolare, l'output contiene la variabile `Inf`.

```
>> A=[0 0;0 1];
>> inv(A)
Warning:  Matrix is singular to working precision
ans=
    Inf   Inf
    Inf   Inf
```

Essendo il calcolo della inversa effettuato per via numerica, questo può essere affetto da grandi errori nel caso in cui la matrice da invertire risulti **mal condizionata**.

Operatori punto

In **Matlab** è possibile definire operatori che compiono su vettori e matrici operazioni elementari elemento per elemento.

Tali operatori, detti **operatori punto**, si ottengono dagli operatori classici premettendo al simbolo che identifica l'operazione il carattere . (punto, da cui il nome).

Nel caso delle operazioni di addizione e sottrazione gli operatori punto lavorano esattamente come gli operatori classici.

Un interessante utilizzo degli operatori punto si ha nella definizione delle funzioni e nella generazione di grafici. Essi consentono infatti che una funzione lavori contemporaneamente sulle singole componenenti di un array.

Operatori punto su vettori

u, v, w vettori di uguale dimensione

s scalare

>> w=u.±v	equivale a	w(i)=u(i)±v(i)
>> w=u.*v	equivale a	w(i)=u(i)*v(i)
>> w=u./v	equivale a	w(i)=u(i)/v(i)
>> w=u.\v	equivale a	w(i)=u(i)\v(i)
		=v(i)/u(i)
>> w=u.^s	equivale a	w(i)=u(i)^s
>> w=u.^v	equivale a	w(i)=u(i)^v(i)

Operatori punto su vettori

```
>> u=[1:4];
>> v=[3 2 6 -1];
>> w=u.*v          % (non prodotto scalare ma per elementi)
w=
   3   4   18   -4
>> w=u./v          % (divisione elemento per elemento)
w=
   0.3333   1.0000   0.5000   -4.0000
>> w=u.^v          % (potenza elemento per elemento)
w=
   1.0000   4.0000   729.0000   0.2500
```

Operatori punto su matrici

A, B, C vettori di uguale dimensione

s scalare

>> C=A.±B	equivale a	c(i,j)=a(i,j)±b(i,j)
>> C=A.*B	equivale a	c(i,j)=a(i,j)*b(i,j)
>> C=A./B	equivale a	c(i,j)=a(i,j)/b(i,j)
>> C=A.\B	equivale a	c(i,j)=a(i,j)\b(i,j)
		=b(i,j)/a(i,j)
>> C=A.^s	equivale a	c(i,j)=a(i,j)^s
>> C=A.^B	equivale a	c(i,j)=a(i,j)^b(i,j)

Operatori punto su matrici

```
>> a=[1 2;3 4];b=[1 2;-1 1];
>> c=a./b     % (non prodotto per l'inversa ma divisione per elementi)
c=
    1   1
   -3   4
>> a.*b           % (prodotto elemento per elemento)
c=
    1   4
   -3   4
>> a.^b           % (potenza elemento per elemento)
c=
   1.0000   4.0000
   0.3333   4.0000
```

Funzioni speciali per array

- `diag(A)`: applicata ad una matrice ne estrae la diagonale, applicata ad un vettore crea la matrice diagonale;
- `tril(A)`: estrae da una matrice la parte triangolare inferiore; precisando un numero intero positivo o negativo come secondo argomento, è possibile estrarre anche le diagonali soprastanti o sottostanti la diagonale principale;
- `triu(A)`: come `tril` ma estrae la parte triangolare superiore;
- `abs(A)`: se applicata ad una matrice reale, produce la matrice dei valori assoluti, se applicata ad un numero complesso, ne calcola il modulo.

Funzioni speciali per array

- `sum(A)`: se applicata ad una matrice fornisce un vettore che contiene le somme per colonna degli elementi della matrice, se applicata ad un vettore fornisce uno scalare dato dalla somma degli elementi del vettore;
- `max(A)`: se applicata ad un vettore calcola il massimo degli elementi e la sua posizione, se applicata ad una matrice produce un vettore che contiene il massimo degli elementi della matrice per colonne e le relative posizioni sulle righe;
- `min(A)`: come `max` ma calcola il minimo.

Funzioni speciali per array

- `det(A)`: calcola il determinante di una matrice quadrata non singolare;
- `rank(A)`: calcola il rango di una matrice come massimo numero di colonne (o righe) linearmente indipendenti;
- `trace(A)`: calcola la traccia di una matrice cioè la somma degli elementi diagonali che coincide con la somma degli autovalori.

Funzioni speciali per array

- `norm(A)`: calcola la norma 2 di un vettore o di una matrice (equivalentemente `norm(A,2)`);
- `norm(A,inf)`: calcola la norma infinito di una matrice;
- `norm(A,1)`: calcola la norma 1 di una matrice;
- `cond(A)`: calcola il numero di condizionamento in norma 2 di una matrice;
- `eig(A)`: calcola gli autovalori di una matrice quadrata. Se usata come `[X,D]=eig(A)` produce una matrice diagonale `D` che contiene gli autovalori di A e una matrice `X` le cui colonne sono gli autovettori di A.

Funzioni speciali per array (Esempi)

```
>> A=[1 2 3;4 5 6;7 8 9];

>> v=diag(A)
v=
   1   5   9

>> diag(v)
ans=
   1   0   0
   0   5   0
   0   0   9
```

Funzioni speciali per array (Esempi)

```
>> A=[1 2 3;4 5 6;7 8 9];

>> tril(A)
ans=
   1   0   0
   4   5   0
   7   8   9

>> tril(A,-1)
ans=
   0   0   0
   4   0   0
   7   8   0

>> tril(A,1)
ans=
   1   2   0
   4   5   6
   7   8   7
```

Funzioni speciali per array (Esempi)

```
>> A=[1 2 3;4 5 6;7 8 9];

>> triu(A)
ans=
   1   2   3
   0   5   6
   0   0   9

>> triu(A,-1)
ans=
   1   2   3
   4   5   6
   0   8   9

>> triu(A,2)
ans=
   0   0   3
   0   0   0
   0   0   0
```

Funzioni speciali per array (Esempi)

```
>> B=[-1 2 -3;4 -5 6;7 8 -9];

>> abs(B)
ans=
   1   2   3
   4   5   6
   7   8   9

>> z=2+i*3;

>> abs(z)
ans=
   3.6056
```

Funzioni speciali per array (Esempi)

```
>> A=[10 8 3;-4 5 12;17 4 9];
>> w=sum(A)
w=
  23   17   24
>> max(A)
ans=
   17   8   12
>> [x pmax]=max(A)
x=
   17   8   12
pmax=
   3   1   2
```

```
>> min(A)
ans=
   -4   4   33
>> [y pmin]=min(A)
y=
   -4   4   3
pmin=
   2   3   1
```

Funzioni speciali per array (Esempi)

```
>> w=[12 3 5 32];

>> sum(w)
ans=
   52

>> max(w)
ans=
   32

>> min(w)
ans=
   3
```

Funzioni speciali per array (Esempi)

```
>> M=[10 10 9;10 9 8;9 8 7];C=[0 3 1;1 2 4];
>> det(M)          % (determinante di M)
ans=
   1
>> rank(C)         % (rango di C)
ans=
   2
>> trace(M)        % (traccia di M)
ans=
   26
>> sum(eig(M))     % (traccia di M uguale alla somma degli autovalori)
ans=
   26
```

Funzioni speciali per array (Esempi)

```
>> A=[1 2 3;4 5 6;7 8 9];

>> norm(A)
ans=
   16.8481

>> norm(A,2)
ans=
   16.8481

>> cond(A)
ans=
   8.5796e+016
```

```
>> norm(A,inf)
ans=
   24

>> norm(A,1)
ans=
   18
```

Funzioni speciali per array (Esempi)

```
>> A=[1 2 3;4 5 6;7 8 9];

>> [X,D]=eig(A)
X =
    0.2320    0.7858    0.4082
    0.5253    0.0868   -0.8165
    0.8187   -0.6123    0.4082

D =
   16.1168         0         0
         0   -1.1168         0
         0         0   -0.0000
```

Matrici particolari

I seguenti comandi permettono di creare direttamente particolari tipi di matrici

- `eye(n)`: genera la matrice identità di ordine `n`;
- `zeros(n,m)`: genera una matrice `n`×`m` di elementi tutti `0`; con un solo parametro `n` genera la matrice quadrata `n`×`n`;
- `ones(n,m)`: genera una matrice `n`×`m` di elementi tutti `1`; con un solo parametro `n` genera la matrice quadrata `n`×`n`;
- `rand(n,m)`: genera una matrice `n`×`m` di numeri casuali tra 0 e 1; con un solo parametro `n` genera la matrice `n`×`n`;
- `hilb(n)`: genera la matrice di Hilbert di ordine `n`.

Matrici particolari (Esempi)

```
>> eye(3)
ans =
   1   0   0
   0   1   0
   0   0   1

>> hilb(3)
X =
   1.0000   0.5000   0.3333
   0.5000   0.3333   0.2500
   0.3333   0.2500   0.2000
```

Matrici particolari (Esempi)

```
>> zeros(2)
ans =
   0   0
   0   0
```

```
>> ones(2)
ans =
   1   1
   1   1
```

```
>> zeros(2,3)
ans =
   0   0   0
   0   0   0
```

```
>> ones(2,3)
ans =
   1   1   1
   1   1   1
```

```
>> zeros(2,1)
ans =
   0
   0
```

```
>> ones(2,1)
ans =
   1
   1
```

```
>> zeros(1,3)
ans =
   0   0   0
```

```
>> ones(1,3)
ans =
   1   1   1
```

Matrici particolari (Esempi)

```
>> rand(3)
ans =
   0.5234   0.11235   0.3165
   0.2133   0.93814   0.5456
   0.4199   0.2317    0.8876
>> rand(1,3)
ans =
   0.2190   0.0470   0.6789
>> rand(2,2)
ans =
   0.6793   0.3835
   0.9347   0.5194
```

Matlab come linguaggio di programmazione

In **Matlab** sono disponibili le principali istruzioni che lo rendono un linguaggio strutturato:

- operatori relazionali: $<$, $<=$, $>$, $>=$, $==$, $=$;
- operatori logici: `&` (*and*), `|` (*or*), `~` (*not*);
- cicli controllati da contatore: `for`, `while`;
- strutture condizionali: `if` ... `else` ... `end`;
- uscita incondizionata: `break`.

Operatori relazionali (Esempi)

Esempi

```
>> 3>5
ans=
   0
>> A=[2 1;0 3]
>> B=[2 -1;-2 3]
>> A==B
ans=
   1   0
   0   1
```

Cicli controllati da contatore (for)

Ripetizione di un insieme di istruzioni per un numero predeterminato di iterazioni

```
for var = val_0 :  step :  val_1
   blocco istruzioni
end
```

step, se non specificato, vale 1

Esempi:

```
a=0;
for i=1:5
   a=a+2;
end
disp(a)
```

```
fat=1;
a=5;
for n=1:a
   fat=n*fat;
end
```

Cicli controllati da contatore (while)

Ripetizione di un insieme di istruzioni fintanto che una determinata condizione rimane vera

```
while espressione logica
   blocco istruzioni
end
```

Esempio:

```
n=10;
while n>1
   n=n/2;
end
```

Per terminare forzatamente l'esecuzione di un ciclo `for` o `while` si usa l'istruzione `break`.

Struttura condizionale

```
if espressione logica_(if)
      blocco istruzioni_(if)
elseif espressione logica_1
      blocco istruzioni_1
:
elseif espressione logica_n
      blocco istruzioni_n
else
      blocco istruzioni_(else)
end
```

Struttura condizionale

Esempio

```
n=input('Introdurre un numero');
if n=0
   disp('n = 0')
elseif n<0
   disp('n negativo')
else
   disp('n positivo')
end
```

Strutture di controllo

La struttura condizionale non necessita, come accade in altri linguaggi, dell'apposizione `then` dopo il comando `if`.
I rami `elseif` possono essere più di uno o assenti.
Il ramo `else` può essere uno solo o assente.

Ad ogni istruzione `if`, `for`, `while` deve *necessariamente* corrispondere un comando `end`.

Tutte le istruzioni sopra introdotte si possono digitare su linee diverse oppure sulla stessa linea di comando e, in tal caso, separate da virgole.

Matlab come linguaggio di programmazione

Esempio

Si costruisca una matrice di Hilbert

$$\{H_{ij}\} = \frac{1}{i+j-1}$$

di dimensione $n = 5$.

Successivamente si dimostri che si tratta di una matrice malcondizionata, utilizzando la funzione intrinseca `hilb` per generare la matrice H e quindi il comando `cond` per calcolare il numero di condizionamento in norma 2 al crescere della dimensione n.

Matlab come linguaggio di programmazione

```
>> n=5;
>> for i=l:n,for j=l:n,H(i,j)=1/(i+j-l);end,end
>> H
 H=
    1.0000   0.5000   0.3333   0.2500   0.2000
    0.5000   0.3333   0.2500   0.2000   0.1667
    0.3333   0.2500   0.2000   0.1667   0.1429
    0.2500   0.2000   0.1667   0.1429   0.1250
    0.2000   0.1667   0.1429   0.1250   0.1111
```

Matlab come linguaggio di programmazione

Usando le funzioni di libreria

```
>> hilb(5)
 ans=
      1.0000   0.5000   0.3333   0.2500   0.2000
      0.5000   0.3333   0.2500   0.2000   0.1667
      0.3333   0.2500   0.2000   0.1667   0.1429
      0.2500   0.2000   0.1667   0.1429   0.1250
      0.2000   0.1667   0.1429   0.1250   0.1111
```

Matlab come linguaggio di programmazione

```
>> cond(ans)
ans=
      4.7661e+005
>> H10=hilb(10);
>> cond(H10)
ans=
      1.6025e+013
```

Per calcolare l'andamento del numero di condizionamento al variare di n si può costruire un array in cui si memorizzano i numeri di condizionamento delle matrici fino all'ordine massimo desiderato.

Matlab come linguaggio di programmazione

```
>> n=input('Dimensione max matrici >>');
Dimensione max matrici >> 10
>>for i=1:n,H=hilb(i);cond_h(i)=cond(H);end
>> cond_h
 cond_h=     1.0e+13 *

            Columns 1   through 4
              0.0000      0.0000    0.0000   0.0000
            Columns 5   through 8
              0.0000      0.0000    0.0000   0.0015
            Columns 9   through 10
              0.0493      1.6024
```

Istruzioni di Input/Output

Per l'introduzione dell'input **Matlab** prevede il comando

`input`

Per la visualizzazione dell'output, **Matlab** prevede tre diversi comandi:

`disp`	output non formattato;
`fprintf`	output formattato;
`sprintf`	output formattato.

Istruzioni di Input (`input`)

Var=`input(`*Stringa di caratteri*`)`

permette di assegnare alla variabile *Var* il valore di un'espressione introdotta da tastiera. Può essere introdotto un valore di tipo scalare, vettore oppure matrice, usando la sintassi standard di **Matlab**.

Per l'assegnazione in ingresso di stringhe di caratteri, si può consultare `help input`.

Stringa di caratteri contiene un messaggio che viene mostrato sulla finestra dei comandi.

```
>> n=input('Inserisci il numero di valori:  ');
>> x=linspace(0,pi,n);
```

costruisce il vettore `x` di `n` elementi equispaziati tra 0 e π.

Output non formattato (`disp`)

`disp(`*var*`)`

mostra il contenuto della variabile *var* senza stampare il nome della variabile stessa

```
>> a=3;
>> fdisp(a)
     3
```

Output non formattato (`disp`)

`disp('`*messaggio*`')`

visualizza sullo schermo il messaggio racchiuso tra apici

```
>> disp('Oggi e'' lunedi''')
    Oggi e' lunedi'
```

Per utilizzare i simboli di apostrofo/accento all'interno della stringa si utilizza il simbolo '' come delimitatore, per evitare conflitti con il segnale di inizio e fine della stringa.

Output non formattato (disp)

Per visualizzare più stringhe, si utilizza come argomento un vettore riga di stringhe, che vengono concatenate

```
>> disp(['Gennaio ','Febbraio ','Marzo'])
    Gennaio Febbraio Marzo
```

Per costruire vettori colonna è necessario che le stringhe abbiano tutte la stessa dimensione, cosa che può essere realizzata inserendo un opportuno numero di spazi.

```
>> disp(['Gennaio ';'Febbraio';'Marzo   '])
    Gennaio
    Febbraio
    Marzo
```

Output non formattato (disp)

Per rappresentare in uscita sia valori numerici che stringhe di caratteri si usa più volte il comando disp

```
>> a=2; b=3; c=2+3;
>> disp('Somma di 2 e 3 ='), disp(c)
Somma di 2 e 3 =
     5
```

oppure si usa la funzione num2str che converte in stringa il valore numerico assegnato ad una variabile

```
>> a=2; b=3; c=2+3;
>> stringa = ['c = ', num2str(c)];
>> disp(stringa)
     c = 5
```

Output formattato (`fprintf`)

`fprintf(`*Fid, Formato, Variabili*`)`

- ***Fid*** (opzionale) è un identificatore del file al quale l'output è inviato;
- ***Formato*** è una stringa di testo che contiene caratteri speciali indicanti il tipo di formato dell'output;
- ***Variabili*** è una lista opzionale di variabili separate da una virgola e che hanno un corrispondente all'interno della stringa *Formato*.

Output formattato (fprintf)

```
>> x=10;y=5.5;
>> fid=fopen('prova.txt','w+');
>> fprintf(fid,'x = %d e y= %f\n',x,y);
>> fclose(fid);
```

`fopen` e `fclose` consentono rispettivamente di aprire e chiudere il file *prova.txt*.

L'opzione `w+` consente la scrittura e la lettura del file.

La stringa di *Formato* contiene codici che specificano il tipo di variabile che deve essere convertita in stringa e stampata sul file.

Output formattato (fprintf)

`fprintf` può essere usato anche solo per formattare l'output su video

```
>> x=10;y=5.5;
>> fprintf('x = %d e y= %f\n',x,y);
      x = 10  e  y= 5.500000
```

Output formattato (sprintf)

stringa =sprintf(*Formato*, *Variabili*)

L'output viene indirizzato su una stringa di testo che può essere poi visualizzata mediante il comando disp.

```
>> stringa=sprintf('x = %d e y= %f\n',x,y);
>> disp(stringa)
      x = 10   e   y= 5.500000
```

Formato e *Variabili* hanno lo stesso significato illustrato per il comando fprintf.

Descrittori di formato

%s	formato stringa
%d	formato senza parte frazionaria (intero)
%f	formato numero decimale
%e	formato in notazione scientifica
%g	formato in forma compatta usando %f o %e
\n	inserisce carattere di nuova linea
\r	inserisce il "carriage return" (l'invio-stesso effetto di \n)
\t	inserisce carattere di tabulazione
\b	cancella un carattere a sinistra
\f	inserisce il carattere form-feed, carattere di avanzamento di pagina (inserisce un quadratino bianco)
%%	inserisce il simbolo %
\\	inserisce il backslash(\)

Descrittori di formato

I descrittori di formato numerico richiedono l'indicazione esplicita del numero di cifre totale del numero e di quelle della parte decimale

```
fprintf('%m.ng',numero)
fprintf('%m.nf',numero)
```

- `m` indica "lo spazio totale" che *numero* occupa
- `n` indica "lo spazio riservato" alla parte decimale.

Descrittori di formato

Esempio

Azione di alcuni descrittori di formato in output

Valore	**%6.3f**	**%6.3e**	**%6d**
2	2.000	2.000e+00	2
0.02	0.020	2.000e-02	2.000000e-02
200	200.000	2.000e+02	200
sqrt(2)	1.414	1.414e+00	1.414214e+00
sqrt(0.02)	0.141	1.414e-01	1.414214e-01

Se il numero di cifre del numero è superiore al numero di cifre previste, il campo si allunga di conseguenza senza troncare il numero stesso;

se il numero di cifre effettive è inferiore al numero di cifre previste, il campo viene "allungato" a sinistra con spazi bianchi.

Descrittori di formato (Esempi)

`fprintf('%0.nf',numero)`	*numero* è stampato in notazione decimale con n cifre decimali
`fprintf('%15.5f',1/eps)`	4503599627370496.00000
`fprintf('%12.8f',pi)`	3.14159265
`fprintf('%1.6f',pi)`	3.141593
`fprintf('%8.6f',pi)`	3.141593
`fprintf('%9.6f',pi)`	_3.141593
`fprintf('%10.6f',pi)`	_ _3.141593

Descrittori di formato (Esempi)

`fprintf('%0.ng',numero)`	*numero* è stampato in forma compatta con n-1 cifre decimali
`fprintf('%0.5g',sqrt(5))`	2.2361
`fprintf('%0.5g',1/eps)`	4.5036e+15
`fprintf('%0.8g',1/eps)`	4.5035996e+015

`fprintf('%d',round(pi))`	3
`fprintf('%s','hello')`	hello
`fprintf('Array is %dx%d.',2,3)`	Array is 2x3.

Descrittori di formato (Esempi)

Esempio

Stampare una tabella di valori per le funzioni seno e coseno

```
>> n=input('Inserisci il numero di valori:  ');
>> x=linspace(0,pi,n);
>> c=cos(x);
>> s=sin(x);
>> disp('-----------------------------------------');
>> fprintf('k\t x(k)\t cos(x(k))\t cos(x(k))\n');
>> disp('-----------------------------------------');
>> fprintf('%d\t %3.2f\t %6.5f\t %6.5f\n',[1:n;x;c;s]);
```

Da notare l'uso vettoriale di `fprintf`.
Se la variabile in output è una matrice, questa è letta per colonne, e ad ogni elemento è applicato il corrispondente descrittore di formato.

La visualizzazione avviene per righe il che porta ad una sorta di "trasposizione" della matrice.

Descrittori di formato (Esempi)

Esempio

Le istruzioni

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

creano un file di testo contenente una tabella di valori della funzione esponenziale

0.00	1.00000000
0.10	1.10517092
⋮	⋮
1.00	2.71828183

I files in Matlab

Gli esempi mostrati mettono in evidenza l'esigenza di salvare su file la sequenza di istruzioni programmata, per poterla poi eseguire, eventualmente con un passaggio di parametri di input/output.

Un'altra possibile applicazione che richiede l'utilizzo di file esterni, è la gestione di dati.

I files in **Matlab**

Matlab permette la gestione di tre tipi di files

- M-file: hanno estensione .m e sono utilizzati per i programmi e qualche funzione di **Matlab**;
- MAT-file: hanno estensione .mat e sono utilizzati per salvare i nomi e i valori delle variabili create durante una sessione di lavoro;
- file di dati (ASCII American Standard Code for Information Interchange): contengono dati organizzati in righe e colonne, separati da spazi o da virgole. Sono scritti in un formato appositamente ideato per essere accettato da una grande varietà di software.

M-files

Un M-file è un file di testo che contiene le istruzioni del programma scritto dall'utente, caratterizzato da un nome.

Il nome del file, indicato nel seguito con `nome-file.m`, identifica il progammamma contenuto nel file a tutti gli effetti come un nuovo comando **Matlab**.

Gli M-files possono essere creati utilizzando un qualunque word-processor o editor di testo e sono scritti nel linguaggio **Matlab**.

M-files (script files)

Ci sono due tipi di M-files:

- script file: sequenza di istruzioni **Matlab**. Gli script files si usano, ad esempio, per introdurre matrici con molti elementi o una serie di istruzioni che si vuole salvare;
  non hanno variabili in entrata e in uscita e operano sulle variabili del workspace.

M-files (function files)

- function-file: permettono di definire funzioni che non sono standard;

  sono caratterizzati da un'istruzione iniziale che contiene la parola `function`;
  presentano una struttura nella quale si distinguono:
  - una lista (opzionale) di `n` parametri di output (il risultato dell'elaborazione della function);
  - una lista (anch'essa opzionale) di `m` parametri di input.

M-files (script files)

Esempio

```
n = 5;
x = linspace(0,2*pi,n);
c = cos(x); s = sin(x); f = 2*x.^2-1;
[x' c' s' f']
```

Le istruzioni vengono salvate in un file con un nome, ad es. `esempio.m`, e il nome, senza estensione, viene usato per far sì che le istruzioni siano eseguite, come un comando **Matlab**.

```
>> esempio
 ans=
                0    1.0000e+000                0   -1.0000e+000
      1.5708e+000    6.1232e-017    1.0000e+000    3.9348e+000
      3.1416e+000   -1.0000e+000    1.2246e-016    1.8739e+001
      4.7124e+000   -1.8370e-016   -1.0000e+000    4.3413e+001
      6.2832e+000    1.0000e+000   -2.4493e-016    7.7957e+001
```

M-files (script files)

Esempio

File radice.m

```
% Questo file calcola la radice degli elementi
% di una matrice a, se a>=0,
% altrimenti da' un messaggio di errore
  a=input('Introdurre a')
  if a>=0
    a=sqrt(a);
  else
    display('errore')
  end
```

M-files (script files)

Il file `condhilb.m` calcola il numero di condizionamento delle matrici di Hilbert

```
% Programma per il calcolo del numero di
% condizionamento delle matrici di Hilbert
% fino ad una dimensione pari ad n
n=input('Dimensione massima delle matrici >> ');
for i = 1:n
   H = hilb(i);
   cond_hilb(i) = cond(H);
end
```

Per eseguire il programma basta digitare il comando:

```
>> condhilb
```

M-files (script files)

Tutte le variabili definite in uno script file sono trattate come variabili globali e quindi contribuiscono ad aumentare la quantità di memoria occupata, anche se non servono in seguito (nell'esempio precedente vengono conservate in memoria anche le variabili `i` e `H`).

Per evitare ciò, conviene mantenere distinte le variabili globali (da conservare in memoria) dalle variabili di servizio.

A tale scopo vengono definiti in **Matlab** i `function files`, file di comandi, che hanno parametri in entrata e in uscita.
Le variabili interne a questi programmi non influenzano le variabili del workspace.

M-files (function files)

```
function[y_1,...,y_n]=nome_function(x_1,...,x_m)
% Commenti
   blocco istruzioni
% Assegnazione valore ai parametri di output
   y_1=val_1;y_2=val_2;...y_n=val_n;
   return
```

Il comando `return` è opzionale.

L'interazione tra programma chiamante e function avviene tramite le liste di parametri di input/output, con la peculiarità che le variabili definite all'intemo della function sono variabili locali e quindi eventuali assegnazioni o modifiche su di esse non producono effetti definitivi sul loro valore al momento del ritorno al programma chiamante.

M-files (function files)

Esempio

```
function A = matrice(m,n)
A = rand(m,n);
```

Esempio

```
function y = effe(x);
y=x.^2;
```

La sequenza di istruzioni che definiscono la funzione deve essere salvata in un file che conviene chiamare con lo stesso nome della function ed estensione `.m`.

Se si usa un nome diverso la function deve essere richiamata usando il nome dato al file.

M-files (function files)

Esempio

```
function [xmin,xmax]=minmax(A,m,n)
% minmax(A,m,n) calcola l'elemento minimo, xmin,
% e l'elemento massimo, xmax,
% della matrice A con m righe ed n colonne
xmin=Inf; xmax=-Inf;
for i=1:m
  for j=1:n
    if A(i,j) > xmax
      xmax = A(i,j);
    end
    if A(i,j) < xmin
      xmin = A(i,j);
    end
  end
end
```

M-files (function files)

Per il calcolo del condizionamento delle matrici di Hilbert

```
function [cond_hilb]= condhilb2(n)
% Numero di condizionamento delle matrici di Hilbert fino a dim. n
% Parametri d'ingresso:  n              max dim. matrici di Hilbert
% Parametri d'uscita:    cond_hilb  vettore contenente il nr. di cond.
   for i=1:n
     H=hilb(i);
     cond_hilb(i)=cond(H);
   end
```

Il calcolo effettivo del numero di condizionamento delle matrici di Hilbert di dimensione $n = 1, \ldots, 10$ può essere ottenuto digitando

```
>> [condiz]=condhilb2(10)
```

M-files

Negli esempi presentati si osservi l'uso del carattere %, per inserire dei commenti all'interno del programma.

Le righe di commento di un M-file poste in testa ai files script o immediatamente di seguito alla riga di intestazione dei files function diventano parte dell'`help online` e vengono visualizzate richiamando `help`seguito dal nome del file.

Rappresentazione e salvataggio dei dati

Per memorizzare e/o leggere dati da disco, sotto forma di files, **Matlab** utilizza i comandi

- `diary`
- `save`
- `load`

Il comando `diary`

`diary` *Nomefile*

è usato per salvare una intera sessione di lavoro **Matlab**, trasferendo su un file tutto ciò che compare nella finestra di comando di **Matlab**.

`diary` attiva la copia sul file di testo chiamato *Nomefile* di tutto ciò che comparirà sulla finestra di comando fino a quando non verrà introdotto il comando

`diary off`

per ripristinare la situazione originale.

Il comando diary

Esempio

Lo script

```
diary tabella.txt
a=linspace(0,1);
b=exp(a);
diary off
```

crea il file di testo chiamato *tabella.txt* che contiene le istruzioni `a=linspace(0,1);` e `b=exp(a);`

Il comando `save`

Il comando `save` rappresenta un modo più versatile per registrare dati su file.

L'uso del comando `save` senza parametri

```
>> save nomefile
```

opera il salvataggio di tutte le variabili presenti in memoria al momento dell'operazione.

Se il file *nomefile* non ha estensione a questo è aggiunta l'estensione *.mat*.

Il comando `save`

`save` permette inoltre di salvare il contenuto solo delle variabili indicate

```
>> save Nomefile Elenco Variabili Formato
```

dove *Formato* è un parametro opzionale.

Se tale parametro è omesso il file è salvato in formato binario.

`save` permette diverse opzioni per salvare i dati in un formato diverso da quello binario.
Ciò semplifica il passaggio delle informazioni da elaborare tra **Matlab** e un eventuale programma esterno.

Il formato *-ascii* consente di salvare file in modalità testo.

Il comando save

Esempio

```
n=input('Inserisci il numero di valori:  ');
x=linspace(0,pi,n);
c=cos(x);
s=sin(x);
save tabella.dat 1:n x c s -ascii
```

costruisce un file di testo *tabella.dat* contenente solo i valori di una tabella di valori delle funzioni seno e coseno

Il comando load

Il comando

```
>> load nomefile
```

permette di *leggere* un file dati (in genere in formato *ascii*).

Esempio

Per caricare in memoria la tabella costruita nell'esempio precedente con save

```
load tabella.dat
A=tabella;
disp('-----------------------------------------');
fprintf('k\t x(k)\t cos(x(k))\t cos(x(k))\n');
disp('-----------------------------------------');
fprintf('%d\t %3.2f\t %6.5f\t %6.5f\n',A');
```

Il comando `load`

Per leggere un file dati (in genere in formato ascii) è necessario cancellare prima l'eventuale header (intestazione, commenti, nome del file, data, ...) del file stesso attraverso un editor di testo e sostituire le virgole con uno o più spazi.

Se si vuole leggere da un file di testo che non ha estensione è obbligatorio l'uso del formato *-ascii* anche in lettura.

Se il file contiene `m` righe e `n` dati per ogni riga, **Matlab** crea una matrice `m`$\times$`n` contenente i valori registrati ed a questa assegna il nome del file in lettura senza estensione.

Il comando `load`

Nell'esempio precedente l'istruzione `load tabella.dat` legge i valori presenti nel file `tabella.dat` e li assegna ad una matrice chiamata `tabella`.

La riga successiva `A=tabella` assegna la matrice di valori alla variabile `A`.

Il comando `load` permette di leggere solo alcune variabili, con la sintassi

```
>> load nomefile variabili
```

Il comando load

Esempio

```
>> x=100;w=[0:0.2:1];
>> save pippo.mat w
>> clear all
>> x
???  Undefined function or variable 'x'.
>> w
???  Undefined function or variable 'wx'.
>> load pippo
>> x
???  Undefined function or variable 'x'.
>> w
w=
    0   0.2000   0.4000   0.6000   0.8000   1.0000
```

Con il comando `save` è stata salvata nel file `pippo.mat`, la sola variabile `w`; il comando `load pippo` consente quindi di caricare in memoria soltanto `w`.

Rappresentazione e salvataggio dei dati

È possibile salvare i grafici prodotti in **Matlab** in file utilizzando diversi formati grafici. Il comando

```
>> print Opzioni Nomefile
```

consente di salvare il contenuto della attuale finestra grafica sul file `Nomefile` utilizzando un particolare formato grafico specificato in `Opzioni`. Ad esempio

```
>> plot(x,y);
>> print -dpsc prova1
```

salva il grafico generato nel file Postscript `prova1.ps`

Per l'elenco completo delle opzioni consultare `help print`.

Funzioni definite dall'utente

In **Matlab** ci sono diversi modi per definire una funzione f(x) da valutare all'interno di espressioni.

- `@`
- `inline`
- `eval`
- `feval`

Funzioni definite dall'utente: @

Per dichiarare una funzione di nome f si usa il comando

```
f=@(arglist) <espressione>
```

dove espressione è una **stringa** rappresentante la funzione e arglist è la lista dei nomi delle variabili da cui dipende f.

Esempio

```
>> f=@(x)  x^2;
f =
  @(x)  x^2
>> g=@(x,y)  sqrt(x^2+y^2);
g =
  @(x,y)  sqrt(x^2+y^2)
```

Funzioni definite dall'utente: @

Per dichiarare una funzione a valori in $\mathbb{R}^k$ con $k > 1$.

```
f=@(arglist) <array espressioni>
```

Esempio

```
>> p=@(x)  [x^2,x^3];
p =
  @(x)  [x^2,x^3]
>> q=@(x,y)  [x^3,y^3];
q =
  @(x,y)  [x^3,y^3]
```

Funzioni definite dall'utente: @

In ogni caso per valutare la funzione definita tramite @

```
[lista_output]=nome_fun [lista_input]
```

```
>> zf=f(3);
zf =
   9
>> zp=p(3);
zp =
   9    27
```

```
>> zg=g(3,4);
 zg =
    5
>> zq=q(3,2);
 zq =
    27    8
```

Funzioni definite dall'utente: eval

La funzione

```
eval(<espressione>)
```

dove espressione è una stringa rappresentante una espressione **Matlab**, valuta l'espressione nel valore assegnato alla variabile.

Esempio

```
>> x = 2;
>> eval('x^2-1')
ans =
     3
>> t = 3;
>> eval('t^2-1')
ans =
     8
```

Funzioni definite dall'utente: eval

La funzione eval si usa in function aventi come argomenti di input funzioni matematiche definite tramite espressioni.

Ad esempio, la seguente function riceve in ingresso una funzione data tramite un'espressione nella variabile x e restituisce il valore di questa funzione in a.

```
function y = valuta1(fun,a)
x = a;
y = eval(fun);
return
```

```
>> f = 'x^2-1';
>> y = valuta1(f,2);
```

Funzioni definite dall'utente: feval

La funzione

```
feval(<nomeM-file>,<lista-punti>)
```

dove nomeM-file è una stringa, restituisce il valore della funzione nei punti lista-punti.

La funzione da valutare può essere una funzione predefinita **Matlab**, ad esempio, sin(x), e in questo caso

```
>> feval('sin',pi/2)
```

oppure una funzione definita tramite la function nomeM-file.m

```
>> feval('nomeM-file',2)
```

Funzioni definite dall'utente: feval

La funzione feval si usa in funzioni aventi come argomenti di input un function file.

Ad esempio, la seguente funzione riceve in ingresso un function file fun.m e restituisce il valore in a della funzione definita mediante tale function.

```
function y = valuta2(fun,a)
y = feval(fun,a);
return
```

```
>> y = valuta2('nomefunz',2);
```

Funzioni definite dall'utente: `inline`

Un altro modo per definire una funzione da passare in function come input, fa uso del comando `inline`

```
inline(<espressione>)
```

dove espressione è una stringa rappresentante una espressione **Matlab**

Esempio

```
>> f = inline('x^2-1');
>> valuta2(f,2);
```

La grafica in Matlab

Matlab è un valido strumento per la maggior parte delle esigenze di rappresentazione grafica di dati o funzioni.
Consente infatti di rappresentare

- curve bidimensionali `plot`;
- curve tridimensionali `plotz`;
- superfici in tre dimensioni `mesh, surf`;
- isolinee `contour, meshc,surfc`;
- grafici di quantità scalari nello spazio `slice`.

La grafica in Matlab: Grafici bidimensionali

Il grafico bidimensionale di una funzione f di variabile x, può essere realizzato mediante il comando `plot`, la cui sintassi nella forma più semplice è

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

essendo `x` ed `y` due vettori di uguale lunghezza in cui sono memorizzate rispettivamente ascisse e ordinate di punti appartenenti al grafico della funzione f.

La grafica in Matlab: Grafici bidimensionali

Per rappresentare il grafico di $f(x) = \sin x$ fra $-\pi$ e π

```
>> x = [-pi : 0.01 : pi];
>> y = sin(x);
>> plot(x,y);
```

o in maniera equivalente, utilizzando il comando `eval`,

```
>> x = [-pi : 0.01 : pi];
>> f = 'sin(x)';
>> y = eval(f);
>> plot(x,y)
```

oppure utilizzando il comando `feval`

```
>> x = [-pi : 0.01 : pi];
>> y = feval('sin',x);
>> plot(x,y)
```

La grafica in Matlab: Grafici bidimensionali

Diverse curve possono essere rappresentate nella stessa finestra grafica:
se $x1, \ldots, xN$ sono N vettori, non necessariamente della stessa lunghezza, e $y1, \ldots, yN$ sono i corrispondenti vettori dei valori da rappresentare, il comando `plot` assume la forma

```
plot(x1,y1,...,xN,yN)
```

Le varie curve possono essere rappresentate in stile diverso (diversi colori, tratteggi, ...) facendo seguire a ciascuna coppia di dati xi, yi una opportuna stringa.

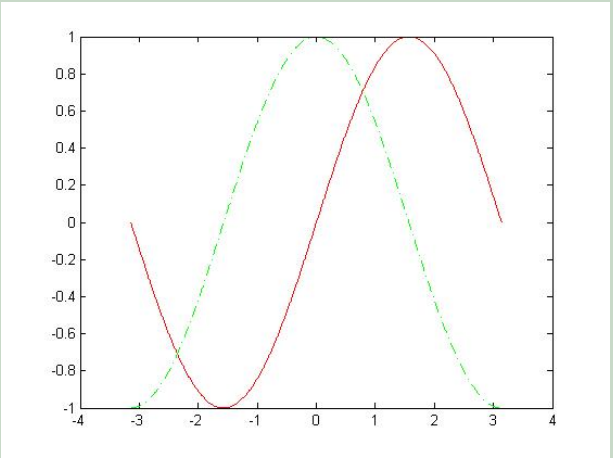
Per l'elenco completo delle stringhe valide `help plot`.

La grafica in Matlab: Grafici bidimensionali

Esempio

Rappresentare le funzioni $\sin x$ e $\cos x$ fra $-\pi$ e π, rispettivamente in linea rossa tratteggiata ed in linea verde tratto-punto.

```
>> x=[-pi:0.01:pi];
>> y=sin(x);z=cos(x);
>> plot(x,y,'r--',x,z,'g-.')
```



La grafica in Matlab: Grafici bidimensionali

Se una finestra grafica è già aperta, un nuovo grafico cancella il disegno preesistente, sostituendolo con il nuovo.
In Matlab è possibile mantenere aperte più finestre grafiche: il comando

```
figure(n)
```

attiva la finestra grafica numero `n`.

A volte può essere utile sovrapporre ad un grafico preesistente un nuovo grafico: ciò può essere realizzato con

```
hold on
```

(`hold off` disattiva questa modalità).

La grafica in Matlab: Grafici bidimensionali

I grafici realizzati possono essere completati da

- titolo `title`;
- etichette degli assi `xlabel, ylabel`;
- didascalie `text, gtext`.

Il tipo di grafico stesso può essere modificato usando al posto di `plot`:

- `semilogy` (`semilogx`) grafico in scala logaritmica sull'asse delle ordinate (ascisse);
- `loglog` grafico in scala logaritmica su entrambi gli assi.

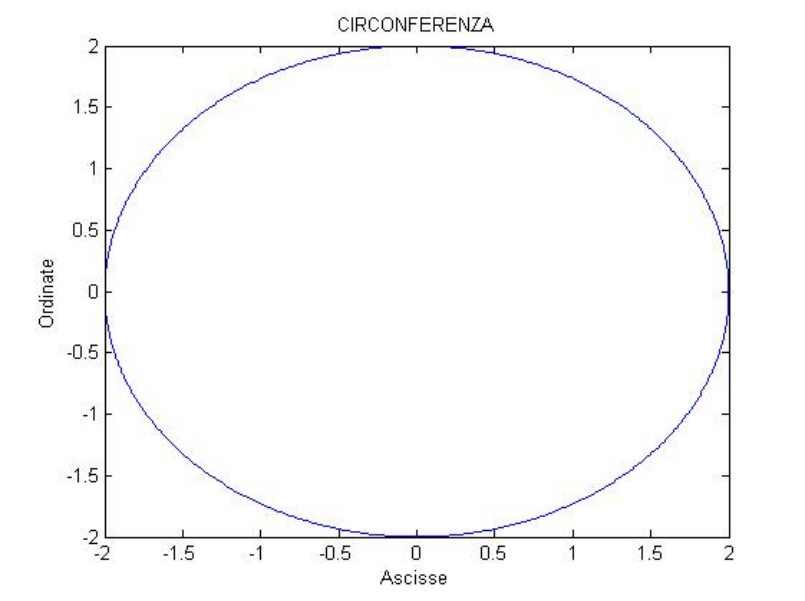
La grafica in Matlab: Grafici bidimensionali

Esempio

Plot di una circonferenza

```
i = 0;
for t=-pi :  0.01 :  pi
      i=i+1;
      x(i)=2*cos(t);
      y(i)=2*sin(t);
end;
plot(x,y)
title('Circonferenza')
xlabel('Ascisse')
ylabel('Ordinate')
```

La grafica in Matlab: Grafici bidimensionali



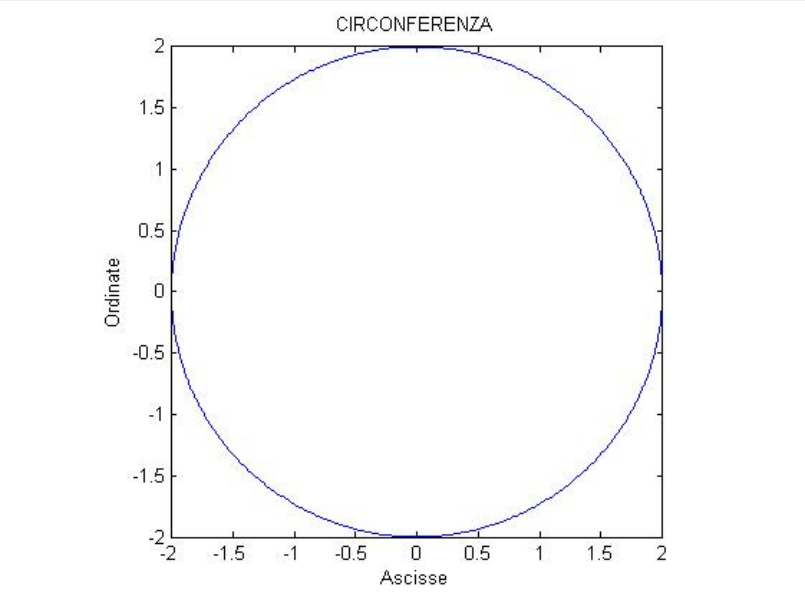
Il grafico ottenuto sembra non rappresentare un cerchio, ma un'ellisse.

L'effetto è dovuto alla differente scalatura degli assi x e y.

La grafica in Matlab: Grafici bidimensionali

Per ottenere un grafico in cui le scale siano rispettate si utilizza

```
axis('square')
```



La grafica in Matlab: Grafici bidimensionali (subplot)

Il comando

```
>> subplot(r,c,i)
```

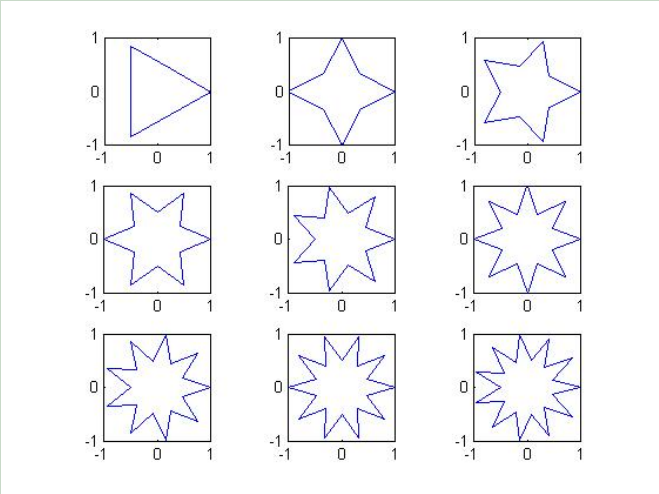
suddivide la finestra grafica corrente in una matrice di sottografici a r righe e c colonne e attiva il sottografico i-esimo (contando per righe)

La grafica in Matlab: Grafici bidimensionali (subplot)

Esempio

Il seguente M-file plotta 9 stelle ad n punte con $n = 3, \ldots, 11$ in una finestra grafica divisa in 9 sottofinestre, una per ogni plot.

```
a=0.5;
for n=3:11
  t=linspace(0,2*pi,n+1);
  s=linspace(pi/n,2*pi-pi/n,n);
  x(1:2:2*n+1)=cos(t);
  x(2:2:2*n)=a*cos(s);
  y(1:2:2*n+1)=sin(t);
  y(2:2:2*n)=a*sin(s);
  subplot(3,3,n-2);
  plot(x,y);
  axis('square');
end
```



La grafica in Matlab: Grafici tridimensionali

I grafici tridimensionali di funzioni reali nelle variabili x, y, sono facilmente ottenibili su domini di forma rettangolare.

Mediante il comando

```
meshgrid
```

Matlab genera una griglia opportuna (matrice) di nodi (xi, yi) nei quali valutare la funzione f che si intende plottare.

Per valutare la funzione f sul rettangolo

$$R = \{(x, y) : x_0 < x < x_1, y_0 < y < y_1\}$$

utilizzando n nodi lungo le direzioni coordinate si usa il comando

```
>> hx=(x1-x0)/n; hy=(y1-y0)/n;
>> [x,y]=meshgrid(x0:hx:x1,y0:hy:y1);
```

La grafica in Matlab: Grafici tridimensionali

La funzione f verrà valutata in corrispondenza delle matrici x e y ed il risultato memorizzato in una matrice z. Tramite il comando `mesh(z)` è possibile poi visualizzare il grafico tridimensionale cercato

Esempio

Disegnare la funzione $f = e^{-0.3(x^2+y^2)} \sin(x^2+y^2)$ sul quadrato $[-4,4] \times [-4,4]$,

```
>> [x,y]=meshgrid(-4:.1:4,-4:.1:4);
>> z=exp(-0.3*(x.^2+y.^2).*sin(x.^2+y.^2);
>> mesh(z);
```

La grafica in Matlab: Grafici tridimensionali

Altri comandi permettono di rappresentare grafici tridimensionali

- `contour` grafico che rappresenta le linee di livello di f
- `meshc` grafico tridimensionale sovrapposto alle isolinee.

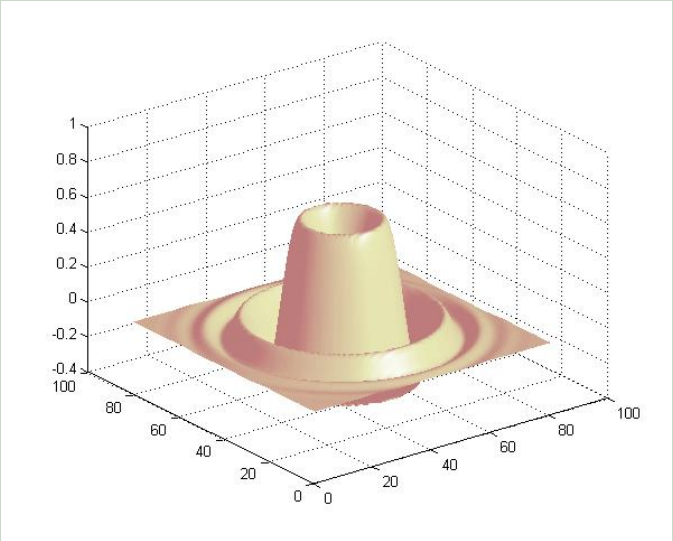
Opportune funzioni permettono di ottenere elaborazioni grafiche più sofisticate

- `surfl` 3-D shaded surface with lighting
- `shading` color shading mode
- `colormap` color look-up table
- ⋮

La grafica in Matlab: Grafici tridimensionali

Esempio

```
>> [x,y] = meshgrid(-4 :  .1 :  4, -4 :  .1 :  4);
>> z=exp(-0.3*(x.^2+y.^2)).*sin(x.^2+y.^2);
>> colormap(pink);
>> surfl(z);
>> shading interp;
```



La grafica in Matlab: Campi vettoriali

La rappresentazione bidimensionale di un campo vettoriale si ottiene utilizzando opportunamente il comando quiver

```
>>  [x,y] = meshgrid(-2:.2:2,-2:.2:2);
>>  zx=y;  zy=-x;
>>  colormap (pink);
>>  quiver (x,y,zx,zy);
>>  axis('square');
```

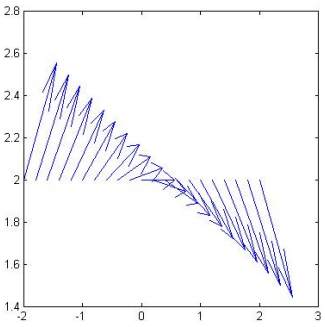


Grafico del campo vettoriale di componenti y, -x.