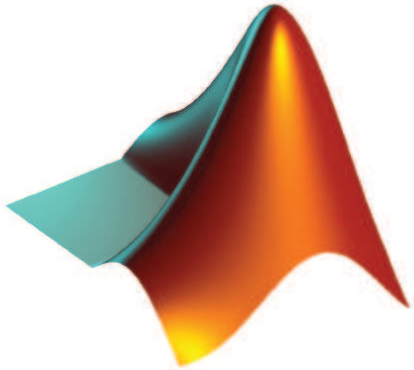
Introduzione a Matlab

Mauro Gaggero



Matlab

- Matlab (MATrix LABoratory) è un ambiente di sviluppo interattivo per il calcolo scientifico. L'elemento base è la matrice, che non richiede dimensionamento. E' lo strumento ideale per risolvere problemi con formulazione matriciale o vettoriale.

- Molte informazioni utili possono essere reperite su:
  - sito ufficiale del produttore: `www.mathworks.com`;
  - programmi sviluppati da utenti: `www.mathworks.com/matlabcentral`;
  - help in linea (`help <nomefunzione>` al prompt dei comandi).

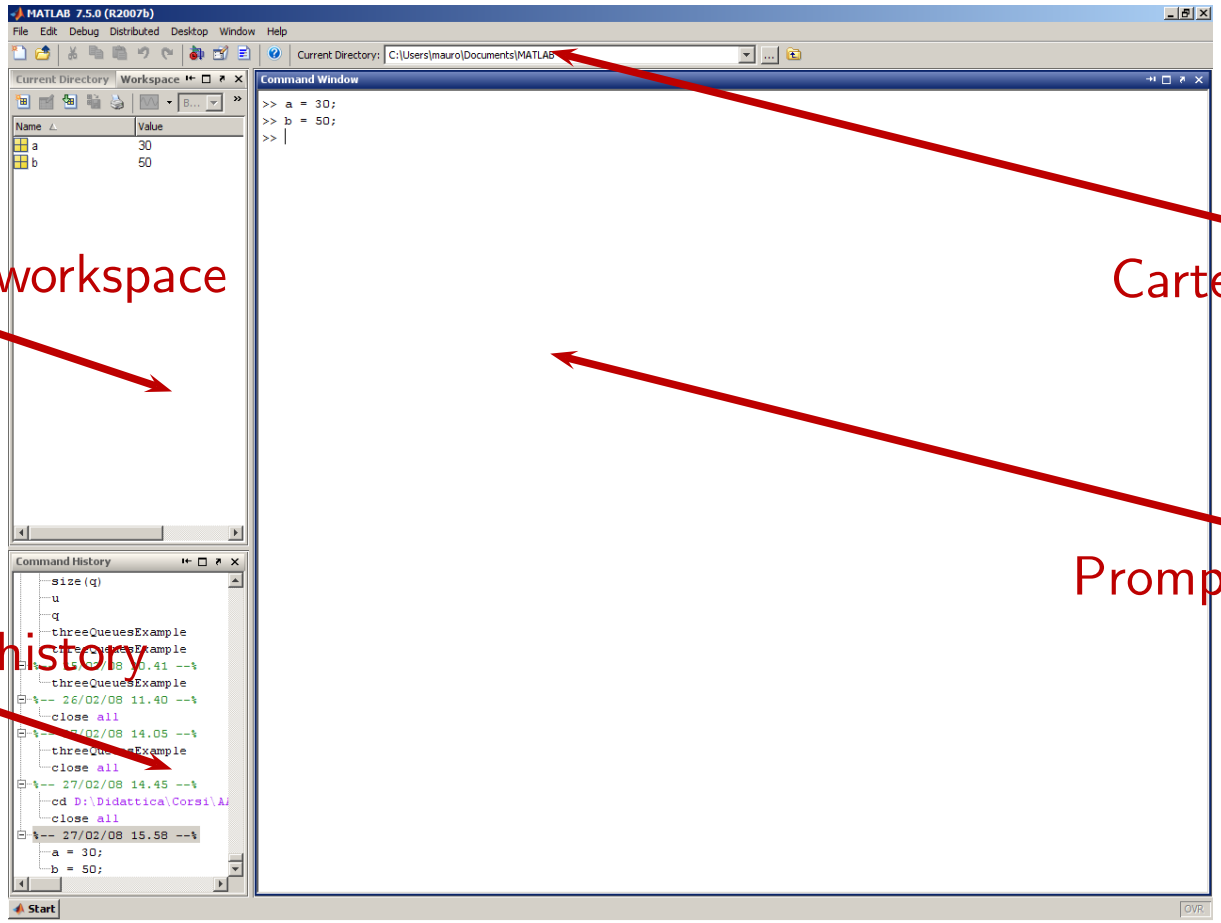
- Brevi ma utili guide per iniziare a utilizzare Matlab:
  - Sigmon, Davis, *Matlab Primer*, 6th Edition, Chapman & Hall, 2001.
  - Cavallo, Setola, Vasca, *Guida operativa a Matlab*, Liguori Editore, Napoli, 1994.

L'ambiente Matlab

- Matlab è composto da sei parti principali:
  - ambiente di lavoro (prompt dei comandi dove inserire istruzioni);
  - librerie di funzioni matematiche (sin, cos, log, ecc.);
  - sistema grafico (per creare grafici di funzioni);
  - linguaggio di programmazione (con sintassi intuitiva e simile al C++);
  - Application Program Interface (per l'interfacciamento con altri linguaggi e creazione di interfacce grafiche);
  - Toolbox (pacchetti software e funzioni per risolvere problemi specifici).

L'ambiente Matlab

- Qui sotto è riportata la finestra di lavoro di Matlab:



Perché Matlab

- C'è una serie di buoni motivi per cui usare Matlab:
  - è semplice e veloce da usare e imparare;
  - è un linguaggio ad alto livello per le operazioni matriciali;
  - fornisce molte funzioni grafiche;
  - permette di arrivare velocemente alla definizione di un problema e alla verifica di nuovi algoritmi e nuove idee (rapid prototyping);
  - esistono funzioni già sviluppate in moltissimi campi dell'ingegneria, della matematica, della fisica, dell'economia, della finanza, e altro ancora.

- Si tratta di un linguaggio interpretato, portatile su diverse piattaforme.

- Le sue prestazioni in termini di velocità di calcolo sono però molto basse se confrontate con linguaggi compilati quali C++ o Fortran.

Chi usa Matlab

- Matlab è largamente impiegato sia in campo educativo sia in campo applicativo:
  - nelle università è utilizzato sia come strumento di ricerca sia come strumento di apprendimento e di esercitazione;
  - nelle industrie è lo strumento preferito per la realizzazione di progetti di ricerca, sviluppo, e analisi.
- Può essere definito un "Problem Solving Environment", ossia un sistema software che fornisce tutti gli strumenti per risolvere un problema in una determinata area.

Come si impara Matlab?

- La risposta migliore a questa domanda è: usandolo!
- Per questo motivo questa presentazione non è e non vuole essere una guida esaustiva all'uso di Matlab.
- Familiarità con il linguaggio si ottiene solo utilizzandolo realmente e consultando di volta in volta la documentazione che accompagna ogni funzione.

Alcuni caratteri speciali

- Qui di seguito è riportato un elenco di alcuni caratteri speciali untilizzati da Matlab:

%	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
+ − * / \ ^	sono operatori algebrici (funzionano in modo "matriciale")
.* ./ .\ .^	sono operatori di moltiplicazione e divisione elemento per elemento
′	indica il trasposto complesso coniugato (trasposto per matrici reali)
< <= > >=	sono operatori di disuguaglianza
&& || ∼	sono operatori logici

Prompt dei comandi

- Il prompt dei comandi è l'interfaccia offerta da Matlab per inserire comandi e definire variabili. Esso è indicato con >>.
- Lo workspace è l'area di memoria accessibile dal prompt dei comandi, dove si lavora.
- Per visualizzare i dati disponibili nello workspace si usano i comandi `who` e `whos`. Per cancellare i dati in memoria si usano i comandi `clear <nomevariabile>` o `clear all`.

```
>> a = [1 2 3];
>> b = 5;
>> who
Your variables are:
a  b

>> whos
  Name        Size              Bytes  Class     Attributes
  a           1x3                  24  double
  b           1x1                   8  double
```

Prompt dei comandi

- Sono forniti generici comandi di sistema per manipolare i file:
  - `ls`, `cd`, `del`, ecc.
- Matlab esegue comandi e funzioni che sono nel suo percorso predefinito (path) o nella cartella corrente. Si possono aggiungere percorsi al path attraverso il menu File.
- Si possono eseguire programmi esterni (del sistema operativo oppure propri file eseguibili) premettendo al nome del programma il carattere "!".

Il comando help

- Digitando `help <nomefunzione>` al prompt si ottiene la descrizione delle funzionalità di un comando o di una funzione.

```
>> help sin
 SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded methods:
       darray/sin

    Reference page in Help browser
       doc sin
```

- Il comando `doc <nomefunzione>` richiama una documentazione grafica più estesa, con indice e possibilità di ricerca.

- Il comando `lookfor <parolachiave>` mostra un elenco delle funzioni che riguardano la parola chiave specificata.

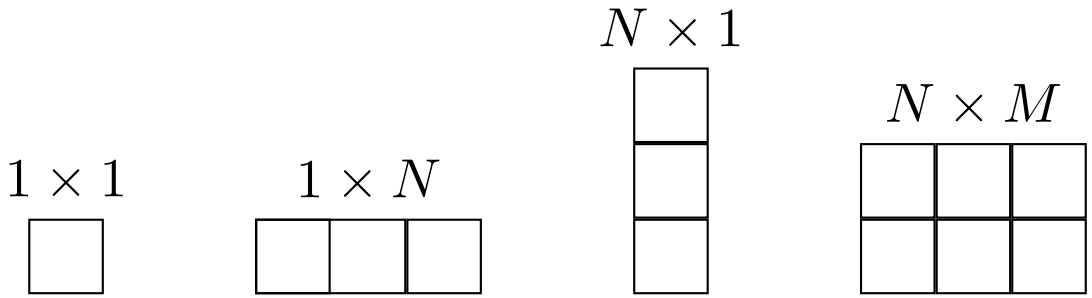
Definizione di variabili

- Matlab non usa definizione di tipo o dichiarazione di dimensioni delle variabili.
- 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).
- Ad esempio, per creare due variabili chiamate `a` e `b` al prompt e sommarle salvando il risultato in una terza variabile chiamata `c`, occorre scrivere:

```
1 >> a = 1;     % Assegna alla variabile 'a' il valore 1
2 >> b = 2;     % Assegna alla variabile 'b' il valore 2
3 >> c = a+b;   % Assegna alla variabile 'c' la somma di 'a' e 'b'
```

Strutture dati: matrici

- L'elemento base di Matlab è la matrice rettangolare di numeri reali a doppia precisione.
- Particolare interesse per le matrici 1×1 (scalari), $1 \times N$ (vettori riga), $N \times 1$ (vettori colonna).



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

Creazione di matrici

- Per scrivere esplicitamente gli elementi di una matrice occorre:
  - separare gli elementi di una riga con spazi o virgole;
  - finire le righe con punto e virgola;
  - racchiudere tutti gli elementi tra parentesi quadrate.

```
>> a = 5  % Costruzione di uno scalare
a =
     5

>> A = [4 32 5 10]  % Costruzione di un vettore riga
A =
     4    32     5    10

>> B = [1; 3]  % Costruzione di un vettore colonna
B =
     1
     3

>> C = [3 6 9; 0 45 2]  % Costruzione di una matrice
C =
     3     6     9
     0    45     2
```

Dimensioni delle matrici

- Per conoscere le dimensioni di una matrice si utilizza il comando `size`:

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

>> [righe, colonne] = size(a)
righe =
     2
colonne =
     3

>> size(a,1)  % Estrazione numero di righe
ans =
     2

>> size(a,2)  % Estrazione numero di colonne
ans =
     3
```

- Per i vettori è possibile anche utilizzare il comando `length`.

Dimensioni delle matrici

- `ans` (abbreviazione di "answer") è la variabile predefinita in cui Matlab memorizza i risultati di un'espressione.

- Il punto e virgola alla fine delle istruzioni impedisce la visualizzazione del risultato dell'espressione. Senza punto e virgola il risultato viene mostrato a video.

Elementi di matrice

- Per accedere all'elemento nella riga i e colonna j della matrice A si usa la notazione $A(i, j)$.
- Ecco un esempio di istruzioni digitate al prompt per estrarre l'elemento $(2, 3)$ della matrice A:

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

- Gli indici di una matrice sono numeri interi positivi che partono da 1.

L'operatore due punti

- L'operatore due punti è un operatore di fondamentale importanza per costruire vettori equispaziati e per operare con indici. La sintassi di base dell'operatore è la seguente:

  `vettore = inizio:passo:fine`

  dove `vettore` è un vettore riga, `inizio` e `fine` indicano il valore iniziale e finale del vettore, e `passo` è un parametro opzionale che indica l'incremento relativo o la spaziatura tra gli elementi (se omesso `passo=1`).

- Come esempio di costruzione di vettori si consideri il seguente codice:

```
>> x = 1:10      % Elementi da 1 a 10 a passo 1
x =
     1     2     3     4     5     6     7     8     9    10

>> y = 10:-1:1  % Elementi da 10 a 1 a passo -1
y =
    10     9     8     7     6     5     4     3     2     1
```

L'operatore due punti

- Un uso particolarmente importante della notazione due punti si ha nella gestione di indici di vettori e matrici. Tale notazione consente di identificare facilmente un'intera riga o un'intera colonna di una matrice.

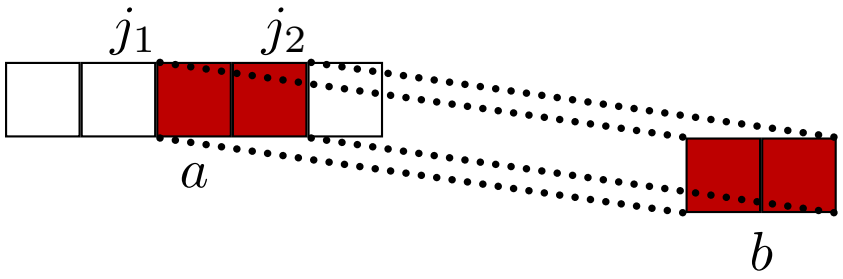
- Si consideri ad esempio il seguente codice:

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(:,1)  % Estrazione di tutte le righe della prima colonna
ans =
     1
     4
     7

>> A(2,:)  % Estrazione di tutte le colonne della seconda riga
ans =
     4     5     6
```

L'operatore due punti

- Tramite la notazione due punti è possibile specificare un intervallo di indici ed estrarre così parti di vettori (o matrici).
- Per quanto riguarda l'estrazione di parti di vettori, si consideri ad esempio il seguente codice:

```
>> a = 0:0.1:0.5
a =
         0    0.1000    0.2000    0.3000    0.4000    0.5000

>> b = a(2:4)  % Estrazione degli elementi di indici 2, 3, e 4
b =
    0.1000    0.2000    0.3000
```

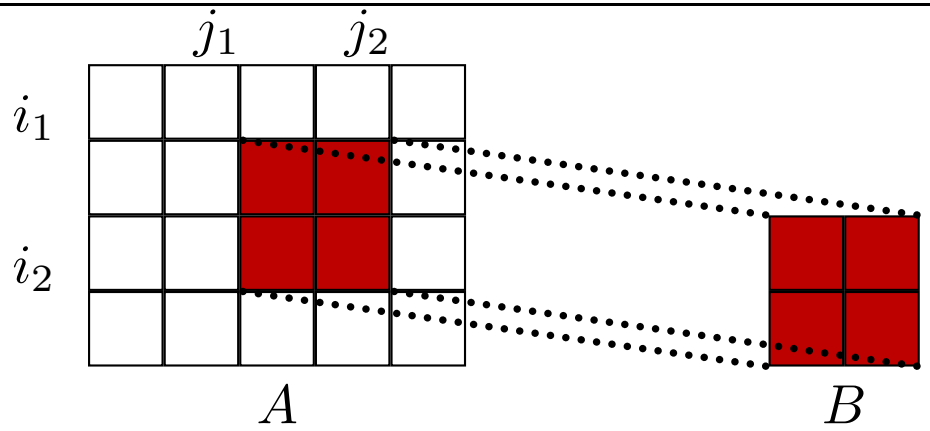


$$b = a(j_1 : j_2)$$

L'operatore due punti

- Per quanto riguarda l'estrazione di parti di matrici, si consideri ad esempio il seguente codice:

```
>> 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

>> B = A(2:3,2:4)  % Estrazione delle righe 2 e 3 e delle colonne 2, 3 e 4
B =
     6     7     8
    10    11    12
```



$$B = A(i_1 : i_2, j_1 : j_2)$$

L'operatore due punti

- L'operatore due punti è utile anche per tracciare grafici di funzioni. Esso infatti permette di creare tabelle di valori di funzioni (dominio e codominio) a partire dalle quali è possibile tracciare il grafico.

- Il dominio è costituito da una sequenza finita di punti in cui calcolare il valore della funzione (insieme discreto a causa della precisione finita con cui vengono rappresentati i numeri in un calcolatore), mentre il codominio è costituito dai valori che la funzione assume in corrispondenza dei punti del dominio.

- Si consideri ad esempio il seguente codice:

```
>> x = 0:pi/4:pi;
>> y = sin(x)  % Calcola il seno in tutti i punti x
y =
         0    0.7071    1.0000    0.7071    0.0000
```

- L'istruzione `y = sin(x)` è una formulazione vettoriale equivalente a un ciclo: il seno è calcolato automaticamente in tutti i punti del vettore `x`.

Gestione di matrici

- Per eliminare elementi, righe, o colonne di un vettore o di una matrice si utilizza il costrutto [ ].
- Si consideri ad esempio il seguente codice:

```
>> a = [1 2 3 4 5 6];
>> a(4) = []    % Eliminazione del quarto elemento
a =
     1     2     3     5     6

>> b = [1 2 3; 4 5 6];
>> b(1,:) = []  % Eliminazione della prima riga
b =
     4     5     6

>> c = [1 2 3; 4 5 6];
>> c(:,1) = []  % Eliminazione della prima colonna
c =
     2     3
     5     6
```

Creazione di matrici

- In Matlab sono presenti diverse funzioni speciali che consentono di costruire particolari matrici e vettori:
  - `linspace`: crea un vettore riga di elementi equispaziati;
  - `logspace`: crea un vettore riga di elementi equispaziati in scala logaritmica;
  - `zeros`: crea una matrice contenente solo elementi uguali a zero;
  - `ones`: crea una matrice contenente solo elementi uguali a uno;
  - `rand`: crea un matrice contenente numeri casuali;
  - `eye`: crea una matrice identità;
  - `diag`: crea una matrice diagonale;
  - `magic`: crea una matrice a valori interi con somme uguali su righe e colonne.
- Consultare la documentazione in linea per una descrizione dettagliata di ciascuna funzione.

Operazioni sulle matrici

- Sulle matrici è possibile effettuare le usuali operazioni algebriche di somma e prodotto. Si consideri ad esempio il seguente codice:

```
>> A = [1 2; 3 4];
>> B = [5 6; 7 8];
>> C = A+B  % Somma elemento per elemento
C =
     6     8
    10    12

>> C = A + [1 1] % Impossibile sommare a causa delle dimensioni delle matrici in gioco,
                 % Matlab segnala un errore
??? Error using ==> plus
Matrix dimensions must agree.

>> D = A*B  % Prodotto matriciale
D =
    19    22
    43    50

>> E = A^2  % Elevamento a potenza (equivalente a A*A matriciale)
E =
     7    10
    15    22
```

Operazioni sulle matrici

- E' possibile effettuare le operazioni di prodotto e quoziente elemento per elemento utilizzando l'operatore punto ".".
- Si consideri ad esempio il seguente codice:

```
>> A = [1 2; 3 4];
>> B = [5 6; 7 8];
>> C = A.*B  % Prodotto elemento per elemento
C =
     5    12
    21    32

>> D = A./B  % Quoziente elemento per elemento
D =
    0.2000    0.3333
    0.4286    0.5000

>> E = A.^2  % Elevamento a potenza elemento per elemento
E =
     1     4
     9    16
```

Operazioni sulle matrici

- Nel caso di moltiplicazioni per scalari è indifferente utilizzare o meno l'operatore punto.
- Si consideri ad esempio il seguente codice:

```
>> A = [1 2; 3 4];
>> B = 2.*A
B =
     2     4
     6     8

>> C = 2*A
C =
     2     4
     6     8
```

Operazioni logiche

- Le operazioni logiche sono operazioni su matrici che forniscono come risultato valori booleani.
- Sono molto usate nei costrutti condizionali, ossia quando si vogliono effettuare operazioni diverse a seconda del valore logico di un'espressione.
- Il valore falso è indicato con 0, il valore vero è indicato con 1.
- In Matlab, il risultato di una operazione logica è costituito da un dato di tipo "logical", che può essere utilizzato come indice per indicizzare altre matrici.

Operazioni logiche

- Si consideri ad esempio il seguente codice:

```
>> A=[1 2 3; 2 2 4]
A =
     1     2     3
     2     2     4

>> B=(A==2)  % Verifica in che posizione la matrice A ha elementi pari a 2
B =
     0     1     0
     1     1     0

>> whos
  Name        Size                Bytes  Class      Attributes
  A           2x3                    48  double
  B           2x3                     6  logical

>> A(B) = 0  % La matrice 'logica' B può essere usata per indicizzare un'altra matrice
A =
     1     0     3
     0     0     4
```

- La matrice B è di tipo "logical" e contiene "1" nelle posizioni corrispondenti al valore "2" della matrice A. Può essere usata per indicizzare altre matrici.

Stringhe

- Anche le stringhe sono viste come matrici: esse sono interpretate come vettori di caratteri. Per creare una stringa si utilizzano gli apici ' '.

```
>> str = 'ciao'
str =
ciao

>> whos
  Name        Size              Bytes  Class     Attributes
  str         1x4                   8  char
```

- Per convertire stringhe in numeri e viceversa (quando possibile) si usano le funzioni `str2num` e `num2str`.

```
>> str = '123';
>> a = str2num(str)
a =
   123

>> whos
  Name        Size              Bytes  Class     Attributes
  a           1x1                   8  double
  str         1x3                   6  char
```

Altre strutture dati

- Altre strutture dati particolarmente importanti in Matlab sono le seguenti:
  - cell array: sono vettori di matrici, e vengono creati mediante il comando `cell`;
  - struct: sono strutture per raccogliere dati diversi (campi), e vengono create mediante il comando `struct`; si accede ai vari campi con l'operatore punto.
- Si rimanda alla documentazione in linea per maggiori dettagli sull'utilizzo di questi comandi.

Visualizzazione scientifica

- Tutte le operazioni che coinvolgono matrici vengono effettuate da Matlab in numeri floating point a doppia precisione (`double`).

- E' possibile cambiare il numero di cifre con cui vengono visualizzati i risultati mediante il comando `format`:

```
>> cos(pi/4)  % Visualizzazione 'breve' di default
ans =
    0.7071

>> format long;  % Notazione 'lunga' con più cifre decimali
>> cos(pi/4)
ans =
   0.707106781186548

>> format short e;  % Notazione scientifica 'breve'
>> cos(pi/4)
ans =
  7.0711e-001.

```

```
>> format long e;  % Notazione scientifica 'lunga'
>> cos(pi/4)
ans =
    7.071067811865476e-001

>> format short eng;  % Notazione ingegneristica 'breve'
>> cos(pi/4)
ans =
   707.1068e-003

>> format long eng;  % Notazione ingegneristica 'lunga'
>> cos(pi/4)
ans =
    707.106781186548e-003

>> format short;  % Per tornare alla visualizzazione 'breve' di default
>> cos(pi/4)
ans =
    0.7071
```

Visualizzazione scientifica

- L'integrazione delle funzionalità di calcolo numerico con le elevate capacità grafiche è un punto di forza dell'ambiente Matlab.

- Sono forniti un ampio insieme di funzioni di alto livello per visualizzare dati (2D, 3D, istogrammi, torte).

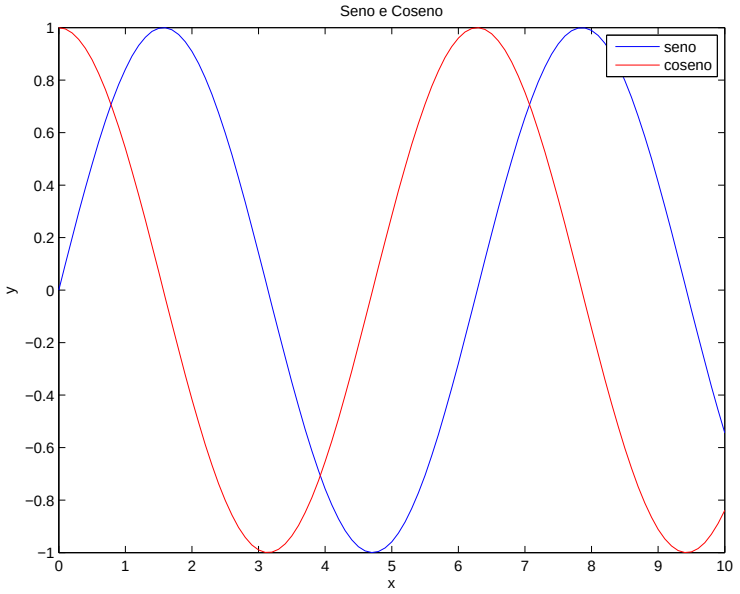
- Ecco un elenco di alcune delle funzioni più usate:
  - `plot`: usato per costruire grafici bidimensionali;
  - `plot3`: usato per costruire grafici tridimensionali;
  - `bar`: usato per costruire particolari grafici bidimensionali (grafici a barre);
  - `mesh`: usato per costruire grafici di superfici;
  - `surf`: simile al precedente, è usato per costruire grafici di superfici;
  - `hist`: usato per costruire istogrammi.

Grafici di funzioni di una variabile

- E' riportato un esempio di utilizzo della funzione `plot` per tracciare grafici di funzioni di una variabile.

- Grafici delle funzioni $y_1 = f_1(x) = \sin(x)$ e $y_2 = f_2(x) = \cos(x)$:

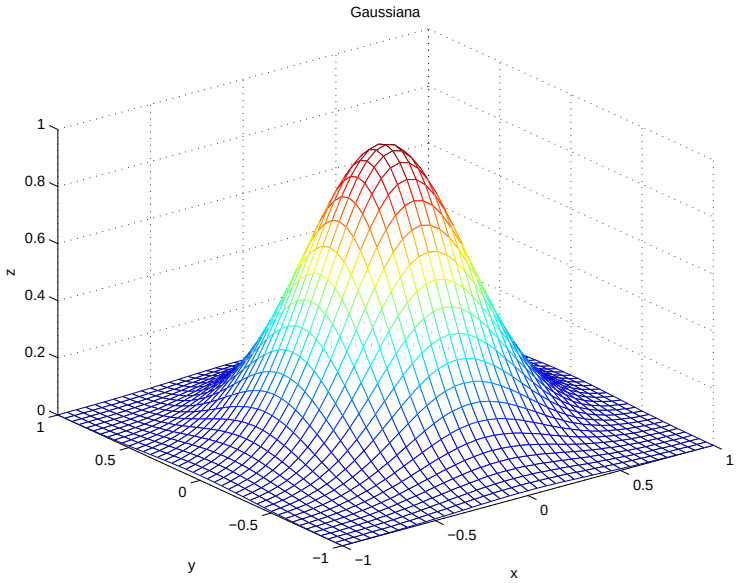
```
% Esempio di utilizzo di plot
x = 0:0.1:10;  % Punti dominio
y1 = sin(x);
y2 = cos(x);
figure;  % Crea una nuova figura
hold('on');  % Per tracciare più grafici
plot(x, y1, '-b');  % Tracciamento grafico 1
plot(x, y2, '-r');  % Tracciamento grafico 2
box('on');  % Inscatola il grafico
xlabel('x');  % Etichetta asse x
ylabel('y');  % Etichetta asse y
title('Seno e Coseno');  % Titolo figura
legend('seno', 'coseno');  % Legenda
% Salvataggio grafico in formato eps
saveas(gcf, 'utilizzoPlot.eps', 'psc2');
```



Grafici di funzioni di due variabili

- E' riportato un esempio di utilizzo della funzione `mesh` per tracciare grafici di funzioni di due variabili.
- Grafico della funzione $z = g(x, y) = \exp\left[-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2}\right]$:

```
% Esempio di utilizzo di mesh
sx = 0.35;
sy = 0.35;
[X,Y]=meshgrid(-1:0.05:+1); % Punti dominio
Z = exp(-(X.^2)/(2*sx^2)-(Y.^2)/(2*sy^2));
figure;  % Crea una nuova figura
mesh(X,Y,Z);  % Tracciamento grafico
xlabel('x');  % Etichetta asse x
ylabel('y');  % Etichetta asse y
zlabel('z');  % Etichetta asse z
title('Gaussiana');  % Titolo figura
% Salvataggio grafico in formato eps
saveas(gcf, 'utilizzoMesh.eps', 'psc2');
```



Controllo flusso dati

- In Matlab si usano 5 costrutti base per il controllo del flusso del programma:
  - `if`: costrutto condizionale;
  - `switch`: usato nel caso di scelte multiple;
  - `for`: usato per cicli di istruzioni;
  - `while`: usato per cicli di istruzioni;
  - `break/return`: usato per l'interruzione di cicli o il ritorno da funzione.

- La sintassi dei vari costrutti è molto simile a quella di altri linguaggi di programmazione, quali C++ o Fortran.

Costrutto if

- Il costrutto condizionale `if` permette di effettuare istruzioni diverse a seconda del valore logico (vero o falso) di una espressione.
- Esempio di utilizzo del costrutto `if`:

```
1 x = 1.2;
2 if(x<0)  % Si effettua questa istruzione se x<0
3    y = -1;
4 elseif(x>0) % Si effettua questa istruzione se x>0
5    y = 1;
6 else % Si effettua questa istruzione se nessuna delle condizioni precedenti è vera
7    y = 0;
8 end
```

- Eseguendo il codice precedente si ottiene come risultato $y = 1$.

Costrutto switch

- Il costrutto `switch` è utilizzato quando occorre scegliere tra diverse possibilità in base al valore di una espressione.
- Esempio di utilizzo del costrutto `switch`:

```
1  i = 1;
2  switch(i)  % Scelte multiple sul valore della variabile 'i'
3  case 0  % Viene scelta questa istruzione se i=0
4     y = 0;
5  case 1  % Viene scelta questa istruzione se i=1
6     y = 1;
7  case 2  % Viene scelta questa istruzione se i=2
8     y = 2;
9  otherwise  % Viene scelta questa istruzione se nessuna delle condizioni precedenti è vera
10    y = 3;
11 end
```

- Eseguendo il codice precedente si ottiene come risultato $y = 1$.
- E' possibile sostituire il costrutto `switch` mediante una sequenza di `if`.

Costrutto for

- Il costrutto `for` è utilizzato per cicli di istruzioni, ossia quando si deve ripetere un numero di volte noto a priori un certo insieme di istruzioni.
- Esempio di utilizzo del costrutto `for`:

```
1 for i=1:3  % 'i' è l'indice del for e assume valori 1, 2, e infine 3
2    for j=1:3  % 'j' è l'indice del for e assume valori 1, 2, e infine 3
3       A(i,j) = (i-1)*3 + j;
4    end
5 end
```

- Eseguendo il codice precedente si ottiene come risultato la matrice $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$.
- In Matlab è più efficiente usare codice "vettorizzato" anziché usare cicli `for`.

Costrutto while

- Il costrutto `while` è utilizzato per cicli di istruzioni, ossia quando si deve ripetere un numero di volte non noto a priori un certo insieme di istruzioni.
- Esempio di utilizzo del costrutto `while`:

```
1 b = 10;
2 a = -3;
3 while(b-a>0)  % Il ciclo viene ripetuto finché la condizione entro parentesi è vera
4    b = b-1;
5 end
```

- Eseguendo il codice precedente si ottiene come risultato $b = -3$.

Costrutto break/return

- Il costrutto `break` permette di interrompere cicli di istruzioni al verificarsi di una data condizione.
- Il costrutto `return` permette di terminare una funzione ritornando così il controllo alla funzione chiamante.
- Esempio di utilizzo del costrutto `break`:

```
1 b = 10;
2 a = -3;
3 while(b-a>0)
4     if(b<=0)
5         break;  % Se b<=0 il ciclo while viene interrotto
6     end
7     b = b-1;
8 end
```

- Eseguendo il codice precedente si ottiene come risultato $b = 0$.

M-file

- Gli m-file sono file che contengono codice in linguaggio Matlab. Devono avere estensione .m. Possono contenere qualunque istruzione che è possibile utilizzare al prompt dei comandi.

- Sono file di testo: possono essere creati con qualsiasi editor ma è preferibile usare quello di Matlab (evidenzia la sintassi e contiene un debugger).

- Esistono due tipi di m-file, script e function (funzioni):

Script	**Function**
- Non ha argomenti di Input/Output - Opera sui dati nello workspace - Serve per elencare sequenze di comandi	- Ha argomenti di Input/Output - Le variabili interne sono locali - Utile quando si deve richiamare più volte lo stesso codice o si vogliono estendere le funzionalità di Matlab

M-file: script

- Esempio di script per eseguire una sequenza di istruzioni (batch):

```
1  % Esempio di script. File esempioScript.m
2  a = 2;  %Queste variabili sono memorizzate nello workspace
3  b = 3;
4  c = 0;
5  if(scelta==1) % Il valore di 'scelta' è preso dallo workspace
6     c = a*b;
7     display('operazione moltiplicazione'); % L'istruzione display scrive a monitor
8                                            % la stringa passatagli
9  else
10    c = a/b;
11    display('operazione divisione');
12 end
```

- La variabile `scelta` si deve trovare nello workspace. In caso contrario viene visualizzato un messaggio di errore. Il valore usato nello script è quello presente nello workspace.

- Tutte le variabili create all'interno dello script sono memorizzate nello workspace.

M-file: script

- Per eseguire lo script `esempioScript` creato in precedenza occorre utilizzare il seguente codice:

```
>> esempioScript  % La variabile 'scelta' non è presente nello workspace
                  % Matlab restituisce un messaggio di errore
??? Undefined function or variable 'scelta'.
Error in ==> esempioScript at 5
if(scelta==1)

>> scelta = 1;  % Creo la variabile 'scelta' nello workspace
>> esempioScript;  % La variabile 'scelta' è stata appena creata
                   % Lo script usa quel valore senza dare errore
operazione moltiplicazione

>> c  % La variabile 'c' contiene il risultato
c =
     6

>> whos  % Le variabili create all'interno dello script sono salvate nello workspace
  Name         Size              Bytes  Class     Attributes
  a            1x1                   8  double
  b            1x1                   8  double
  c            1x1                   8  double
  scelta       1x1                   8  double
```

M-file: function

- Il costrutto `function` permette di suddividere il programma in sottofunzioni, secondo il paradigma della programmazione strutturata.
- La sintassi per la dichiarazione di una funzione è la seguente:

```
function valoreRitorno = nomeFunzione(elencoParametri)
```

- Nel caso di più valori di ritorno la sintassi è la seguente:

```
function [valRit1,valRit2,valRit3] = nomeFunzione(elencoParametri)
```

- Se non si hanno valori di ritorno la sintassi è la seguente:

```
function nomeFunzione(elencoParametri)
```

M-file: function

- Esempio di function (è buona norma dare al file lo stesso nome della funzione):

```
1  function c = esempioFunction(scelta)
2  % Esempio di funzione. File esempioFunction.m
3  % La funzione esempioFunction ha come parametro la variabile 'scelta' e
4  % restituisce nella variabile 'c' il risultato dell'operazione effettuata
5  a = 2;  % Queste variabili sono locali alla funzione. Non vengono memorizzate
6  b = 3;  % nello workspace
7  c = 0;
8  if(scelta==1)  % Il valore di 'scelta' e' quello passato durante la chiamata
9     c = a*b;
10    display('operazione moltiplicazione'); % L'istruzione display scrive a monitor
11                                           % la stringa passatagli
12 else
13    c = a/b;
14    display('operazione divisione');
15 end
```

- La variabile `scelta` deve essere passata come argomento al momento della chiamata. Tutte le variabili create all'interno della funzione sono locali e non sono accessibili dall'esterno. Una funzione non può accedere alle variabili dello workspace (si possono però definire variabili globali con il comando `global`).

M-file: function

- Per utilizzare la funzione `esempioFunction` creata in precedenza occorre usare il seguente codice:

```
>> scelta = 1;       % La variabile 'scelta' è privata alla funzione
>> esempioFunction   % Non viene utilizzata quella nello workspace
??? Input argument "scelta" is undefined.

Error in ==> esempioFunction at 6
if(scelta==1)

>> out = esempioFunction(1);  % 'scelta' è passata come argomento e vale in questo caso 1
                              % Il risultato è assegnato alla variabile out
operazione moltiplicazione

>> out  % Valore ritornato dalla funzione
out =
     6

>> whos  % Le variabili interne alla funzione non sono salvate nello workspace
  Name        Size              Bytes  Class     Attributes
  out         1x1                   8  double
  scelta      1x1                   8  double
```

Matlab Tips & Tricks

- Lo svantaggio principale di Matlab è la lentezza di esecuzione dei programmi.
- La differenza rispetto a linguaggi di programmazione compilati come C++ o Fortran è evidente soprattutto nel caso di programmi complessi e di grandi dimensioni.
- Per ottenere codice più veloce ci sono due tecniche:
  - vettorizzazione dei loop, ossia la sostituzione di cicli `for` e `while` con equivalenti operazioni matriciali;
  - pre-allocazione di vettori e matrici, ossia l'allocazione in memoria dei vettori e delle matrici prima del loro utilizzo (usando la funzione `zeros` o `ones`).

Matlab Tips & Tricks

- Esempio di velocizzazione del codice tramite vettorizzazione di cicli:

```
1  % Codice non ottimizzato
2  N = 30000;
3  tic;  % Da questo punto Matlab inizia a contare il tempo di esecuzione
4  i = 0;
5  c = zeros(1,N);
6  for t=0:pi/N:2*pi  % Ciclo for per calcolare il valore del seno in diversi punti
7     i = i+1;
8     c(i) = sin(t);
9  end
10 time1 = toc;  % Qui Matlab termina di contare il tempo di esecuzione
11
12 % Codice ottimizzato
13 tic;  % Da questo punto Matlab inizia a contare il tempo di esecuzione
14 t = 0:pi/N:2*pi;  % Il ciclo for è stato sostituito da queste due istruzioni
15 c = sin(t);  % Il seno è calcolato automaticamente in tutti i punti del vettore 't'
16 time2 = toc;  % Qui Matlab termina di contare il tempo di esecuzione
```

- Eseguendo il programma si ha `time1`=11.7 secondi e `time2`=0.09 secondi.

Matlab Tips & Tricks

- Esempio di velocizzazione del codice tramite pre-allocazione in memoria di matrici:

```
1  % Codice non ottimizzato
2  N = 1000;
3  tic;  % Da questo punto Matlab inizia a contare il tempo di esecuzione
4  for i=1:N  % Ciclo per la costruzione della matrice 'A'
5     for j=1:N
6        A(i,j) = i+j;
7     end
8  end
9  time1 = toc;  % Qui Matlab termina di contare il tempo di esecuzione
10
11 % Codice ottimizzato
12 tic;  % Da questo punto Matlab inizia a contare il tempo di esecuzione
13 B = zeros(N);  % Pre-allocazione delle variabile 'B' in cui salvare i risultati
14 for i=1:N  % Ciclo per costruire la matrice 'B' precedentemente allocata
15    for j=1:N
16       B(i,j) = i+j;
17    end
18 end
19 time2 = toc;  % Qui Matlab termina di contare il tempo di esecuzione
```

- Eseguendo il programma si ha `time1`=9.5 secondi e `time2`=1.12 secondi.

Operazioni di Input/Output

- Matlab possiede molti comandi per scrivere o leggere file su disco.
- I più comuni sono `load` e `save`, rispettivamente per leggere un file e caricarne in memoria il contenuto e per scrivere un file memorizzandovi dei dati.
- Si possono salvare i dati in file ".mat" in un formato particolare riconosciuto da Matlab oppure in formato testuale.

```
1 a = 2;
2 b = 3;
3 save('prova.mat');  % Salva tutto il workspace sul file 'prova.mat'
4 save('prova2.txt', 'a', '-ascii');  % Salva solo la variabile a in un file testuale
5 load('prova3.mat');  % Carica nello workspace tutte le variabili salvate nel file
6 data = load(prova4.txt, '-ascii');  % Carica nella variabile 'data' la matrice presente
7                                     % in 'prova4.txt'
```

Operazioni di Input/Output

- Matlab fornisce anche funzioni di Input/Output simili al C++:
  - `fopen`, `fclose`: per aprire o chiudere file;
  - `fscanf`, `fprintf`: per leggere o scrivere dati formattati;
  - `fread`, `fwrite`: per leggere o scrivere dati in formato binario.

- Si rimanda alla documentazione in linea per i dettagli sulle varie funzioni.

Application Program Interface

- Sebbene Matlab sia un ambiente completo per programmare, esiste la possibilità di interfacciamento con programmi esterni attraverso la Application Program Interface (API).

- E' possibile:
  - utilizzare Matlab da programmi C++ (Matlab engine);
  - utilizzare proprie applicazioni C++ come funzioni "built-in" di Matlab (MEX-file).

- Il primo caso è utile quando si vuole sfruttare in C++ la flessibilità di Matlab, mentre il secondo è utile quando si vuole avere un codice più veloce.

Debugging

- Il debugging è il processo che permette di trovare errori nel proprio codice. Ci sono due tipologie di errori:
  - errori di sintassi: li indica Matlab al prompt;
  - errori runtime: sono errori algoritmici, rilevabili solo durante l'esecuzione del programma a causa di risultati inattesi.

- Gli errori runtime sono i più insidiosi da trovare e da risolvere. Esistono diverse tecniche per individuare questi errori:
  - togliere i punti e virgola dalle istruzioni in modo da visualizzare i risultati intermedi delle varie operazioni;
  - usare l'istruzione `keyboard` per inserire un breakpoint all'interno di una funzione;
  - usare il Matlab debugger.

I Toolbox di Matlab

- I Toolbox di Matlab sono pacchetti software utili per risolvere problemi specifici. Questi pacchetti non fanno parte del kernel vero e proprio di Matlab.

- Si tratta di codice scritto appositamente per risolvere problemi in moltissimi campi dell'ingegneria, della matematica, della fisica, dell'economia, della finanza, e altro ancora.