

Lab. 1 - Introduzione a Matlab

Laboratorio di Calcolo Numerico

Annalisa Buffa Fausto Cavalli

Corso di Laurea in Chimica

A.A. 2008-2009

Perchè è importante l'analisi numerica?

Alcuni esempi di cosa possa succedere se si omette una corretta analisi numerica degli algoritmi scelti si possono trovare digitando “numerical analysis disasters” su un motore di ricerca o collegandosi direttamente all'indirizzo web www.ima.umn.edu/~arnold/disasters/

Vediamo con un esempio molto semplice come facili calcoli matematici eseguiti con un calcolatore possano non dare i risultati previsti.

Cosa è Matlab

Matlab

MATLAB=**MAT**rix **LAB**oratory è uno strumento per il calcolo scientifico utilizzabile

- 1 come una calcolatrice scientifica avanzata
- 2 come un linguaggio di programmazione

Matlab non è un linguaggio simbolico, come per esempio Maple, per cui non è possibile effettuare operazioni analitiche come la derivazione o l'integrazione esatte

L'oggetto elementare con cui lavora sono le **matrici(array)**: ogni quantità (variabile) viene trattata come una matrice di dimensioni $m \times n$. Un numero reale è una matrice 1×1 .

Tutte le **variabili** vengono trattate come **numeri reali** (doppia precisione a 8 byte) anche se Matlab può gestire anche i numeri complessi.

Cosa è Matlab

Matlab include numerose funzioni predefinite di uso generale, dette **built-in functions**.

Esistono inoltre raccolte di funzioni dedicate ad uno specifico argomento che vengono dette **toolboxes**. (es: statistica, pde)

Per ulteriori informazioni su Matlab: www.mathworks.com. Matlab è un software a pagamento.

Controparte gratuita: Octave, che ne riproduce buona parte delle funzioni fondamentali (con una grafica più povera). Octave può essere scaricato alla pagina web www.octave.org.

Attenzione!

Matlab “parla” inglese!

Ambiente di lavoro: finestre

Apriamo Matlab avviandolo tramite la sua icona. Dopo un po' compare l'area di lavoro composta da diverse finestre.

Sulla destra c'è la finestra principale, la **Command Window**, dove vengono digitati i comandi e visualizzati i risultati.

A sinistra della finestra principale ci sono due finestre più piccole: nella parte superiore si può scegliere fra la finestra di **Workspace** quella della **Current Directory**.

Workspace: compare il nome delle variabili, con il valore, la dimensione, il numero di Bytes occupati, ed il tipo.

Current Directory: compare la lista dei file contenuti nella directory corrente con il tipo (file o folder) e la data dell'ultimo aggiornamento.

Nella parte a sinistra in basso si trova la **Command History** che contiene tutti i comandi che vengono eseguiti nella finestra principale.

Ambiente di lavoro: Menu

- **File:** gestisce i file di Matlab. Si apre un Menu con la possibilità di aprire files (nuovi o già esistenti), di stampare e di definire le preferenze.
- **Edit:** soliti comandi per editare i file.
- **View:** configurazione delle finestre.
- **Graphics:** gestisce i grafici.
- **Debug:** per trovare gli errori nel programma.
- **Desktop:** configurazione del desktop di MATLAB.
- **Window:** per navigare nelle finestre.
- **Help:** aiuto in linea (molto utile).

Cartella di lavoro

A questo punto si può creare la propria cartella di lavoro: cliccando sull'icona con i si può cambiare la propria area di lavoro: conviene creare una propria cartella dove lavorare sempre, creando eventualmente di volta in volta delle sottocartelle per organizzare il lavoro

Primi comandi

Alcuni comandi Matlab importanti da conoscere:

```
>> demo      % In matlab le parti che iniziano con %  
             % sono dei commenti
```

mostra esempi significativi di possibili applicazioni del software.

```
>> help [nomecomando] % Molto utile!
```

permette di ottenere informazioni dettagliate su qualsiasi comando o sui pacchetti presenti in Matlab.

```
>> diary lavoro.dat
```

crea un file di testo lavoro.dat nel quale viene salvato tutto ciò che compare (da quel momento in poi) nella Command Window. Invece

```
>> diary off
```

interrompe la scrittura della cronaca e chiude il file, che altrimenti viene chiuso all'uscita da Matlab

Primi comandi

Suggerimento

È utile, specie le prime volte, salvare attraverso il comando `diary` tutto quello che si esegue

Per conoscere il contenuto della cartella corrente si usa

```
>> ls
```

che elenca i file e le sottocartelle

Attenzione!

Un errore comune è cercare di richiamare un file di matlab che non sta nella cartella corrente di lavoro: controllare sempre dove ci si trova e se i file che ci servono sono nella cartella di lavoro attuale

Attenzione!

Matlab è **CASE SENSITIVE**, cioè distingue fra minuscole e maiuscole

Matlab come calcolatrice scientifica

Per eseguire delle operazioni è sufficiente digitarle nella command window: per calcolare $2 - 3 \cdot 7$ si scrive

```
>> 2-3*7
```

e si digita invio. Il calcolo viene eseguito e nella command window compare il risultato

```
ans =  
-19
```

Viene sempre creata una variabile dal nome **ans** che contiene il risultato dell'ultima operazione eseguita non assegnata

Operazioni

Le operazioni matematiche utilizzabili in matlab sono:

^ elevamento a potenza

/ divisione

* prodotto

+ addizione

- sottrazione

elencate secondo le rispettive precedenze

Matlab come calcolatrice scientifica

Per **alterare la precedenza** delle operazioni è necessario usare delle **parentesi tonde** (quadre e graffe hanno altri significati in matlab)

Esercizio

Calcolare la seguente espressione:

$$3 + \frac{2 - 6^{\frac{5}{3}}}{2 * 5}$$

Si possono usare anche **funzioni scientifiche**: la sintassi è *nomefunzione(valori)*; per calcolare $\sqrt{10}$ si può usare

```
>> sqrt(10) % ma anche 10^(1/2)!
```

Si possono calcolare **funzioni goniometriche** dirette e inverse

```
>> sin(pi/3) % pi=3.14... è una costante di matlab
```

```
>> acos(0.3) % acos=arco coseno
```

e funzioni **logaritmiche** ed **esponenziali**

```
>> log(5) % log è il logaritmo in base e
```

```
>> exp(2)
```

Matlab come calcolatrice scientifica

Per ottenere l'elenco delle funzioni presenti si può digitare

```
>> help elfun % elfun=elementary functions
```

Un elenco delle più usate è il seguente

Funzioni elementari

Funzione	Significato
sin, cos, tan	seno, coseno, tangente
asin, acos, atan	arcoseno, arcocoseno, arcotangente
exp	esponenziale
sinh, cosh	seno iperbolico, coseno iperbolico
tanh	tangente iperbolica
log, log2, log10	logaritmo in base e, in base 2 e in base 10
sqrt	radice quadrata
abs	valore assoluto
sign	funzione segno

Matlab come calcolatrice scientifica

È possibile modificare il numero di cifre decimali ed il formato con cui vengono visualizzati i risultati attraverso il comando **format**. Alcune delle opzioni possibili sono

Format

Sintassi

`format`

`format short`

`format long`

`format short e`

`format long e`

`format short g`

`format long g`

Rappresentazione

default; virgola fissa scalata con 5 cifre

virgola fissa scalata con 5 cifre

virgola fissa scalata con 15 cifre

forma esponenziale con 5 cifre di mantissa

forma esponenziale con 15 cifre di mantissa

rappresentazione migliore con 5 cifre

rappresentazione migliore con 15 cifre

Matlab come calcolatrice scientifica

Esercizio

Provare le differenti modalità di rappresentazione applicate, per esempio al calcolo di e^4 , dando di volta in volta il comando `format` e ricalcolando l'espressione data. Tornare alla fine alla rappresentazione standard con `format`

Suggerimento

Matlab offre due modi per evitare di dover ridigitare un comando già dato

- fare doppio click sul comando stesso presente nella history
- scorrere la storia dei comandi dati con le frecce direzionali \uparrow e \downarrow

Assegnazione di variabili scalari

Variabili

Una variabile è una parte di memoria identificata attraverso un nome che contiene dei dati che possono essere modificati nel corso del programma

Nota

`ans` che abbiamo incontrato eseguendo operazioni, è una variabile

Nota

I nomi delle variabili non devono contenere spazi e caratteri speciali come simboli di operazione (`-`, `=`, `+`, `*`, `^`, `/`, `\`), parentesi, apostrofi o accenti, segni di punteggiatura (`.`, `;`). Possono invece contenere numeri ma non come carattere iniziale, possono inoltre contenere il carattere underscore `_`

Assegnazione di variabili scalari

Per assegnare una variabile la sintassi è

$$\text{nomevariabile} = \text{contenuto}$$

Per esempio con

```
>> x=1.54
```

si crea la variabile x che contiene il valore 1.54. Si noti che la nuova variabile creata è comparsa nella finestra di workspace.

Per richiamare una variabile si usa il suo nome

```
>> x
```

Il contenuto di una variabile può essere cambiato semplicemente riassegnando la stessa variabile. Se si digita

```
>> x=log(5) % una variabile può contenere il risultato  
           % di una operazione
```

ora x conterrà un nuovo valore. Definiamo ora una nuova variabile con

```
>> y=6
```

Assegnazione di variabili scalari

Si può assegnare il valore di una variabile sfruttando il contenuto di altre, per esempio

```
>> x=2*y
```

cambia il valore della x in 12.

Nota

Le due variabili appena create sono **indipendenti** tra di loro: se cambio il valore della y quello della x rimane **invariato**, anche se era stata definita a partire dalla y

Per cancellare una variabile si può usare il comando **clear**: se per esempio voglio eliminare la variabile x basta digitare

```
>> clear x
```

Per cancellare tutte le variabili create basta dare il comando

```
>> clear all % funziona anche se si omette all
```

Assegnazione di variabili scalari

Matlab offre alcune variabili predefinite che contengono alcuni valori di costanti matematiche importanti o utili per la programmazione

Variabili predefinite

Nome	Significato
ans	risultato dell'ultima operazione non assegnata
pi	approssimazione di π
i, j	unità immaginaria, $i^2 = -1$
eps	precisione macchina
Inf	∞ risultato di $n/0$ con $n \neq 0$
NaN	Not a Number (0/0, Inf+Inf)

Assegnazione di variabili scalari

Nota

Il contenuto delle variabili predefinite può essere cambiato riassegnando la variabile. Dopo un comando `clear` il valore di una variabile predefinita viene riportato a quello originario. Per esempio

```
>> pi=10  
>> clear pi  
>> pi
```

Attenzione!

In Matlab non esiste tra le variabili predefinite la costante e , base dei logaritmi naturali. Per calcolare e usare il suo valore è necessario scrivere `exp(1)`. Genera quindi un errore una espressione del tipo `e^x`

Assegnazione di variabili vettoriali

Per assegnare un vettore riga basta elencare gli elementi racchiudendoli tra quadre e separandoli con spazi o virgole

```
>> v=[1 2 3]
```

oppure

```
>> v=[1, 2, 3]
```

Per assegnare un vettore colonna gli elementi devono essere separati da punto e virgola

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

Una caratteristica peculiare di matlab è la cosiddetta **notazione due punti**, che permette di definire vettori costituiti da elementi equi-spaziati con un unico comando

$$\text{vettore} = [\text{valore iniz}:\text{passo}:\text{valore finale}]$$

dove il passo può essere omissso se vale 1.

Assegnazione di variabili vettoriali

Per esempio per generare il vettore

$$v = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10]$$

si può dare il comando

```
>> v=[1:10]
```

mentre

```
>> v=[1:0.3:2]
```

```
v =
```

```
1.0000    1.3000    1.6000    1.9000
```

in cui si può notare come l'ultimo valore generato (1.9) non coincida con il valore finale dell'espressione (2). Il passo può anche essere negativo, per esempio

```
>> v=[4:-.5:2]
```

```
v =
```

```
4.0000    3.5000    3.0000    2.5000    2.0000
```

Assegnazione di variabili vettoriali

Un altro comando per generare vettori è **linspace** la cui sintassi è data da

$$\text{vettore} = \text{linspace}(\text{primo estremo}, \text{secondo estremo}, N)$$

Il comando `linspace` genera N valori equi-spaziati fra valore iniziale e valore finale (estremi compresi). Ad esempio

```
>> v=linspace(0,1,5)
v =
    0  0.2500  0.5000  0.7500  1.0000
```

Attenzione

Se nella notazione due punti il valore iniziale è maggiore di quello finale ed il passo è positivo, allora si genera un vettore vuoto.

Se invece con il comando `linspace` il primo estremo è maggiore del secondo, il vettore viene comunque creato con dei punti equi-spaziati decrescenti

Assegnazione di variabili vettoriali

Nota

Ci sono alcune differenze e similitudini fra la notazione due punti ed il comando `linspace`

due punti	<code>linspace</code>
generano vettori riga	
si usa quando si conosce il passo	si usa quando si conosce il n° di suddivisioni
gli estremi non sempre sono inclusi	include sempre gli estremi
non si conosce (e non interessa) a priori la lunghezza del vettore	si vuole un vettore di lunghezza prestabilita

Assegnazione di variabili vettoriali

Per accedere alla componente di un vettore basta indicare fra tonde l'indice dell'elemento che si vuole richiamare

vettore(indice)

Per esempio

```
>> v(3)    % l'indice è sempre un numero naturale!  
ans =  
    0.5000
```

Attenzione!

In Matlab l'indicizzazione inizia da 1 e non da 0! Matlab produce un messaggio di errore quando si cerca di accedere ad una componente non definita (troppo grande o negativa)

In questo modo posso anche cambiare i valori di alcune componenti di un vettore. Per esempio

```
>> v(3)=4  
v =  
    0    0.2500    4.0000    0.7500    1.0000
```

Assegnazione di variabili vettoriali

Per accedere all'ultimo elemento di un vettore è possibile usare la parola chiave `end`

```
>> v(end) % equivale a v(5)
ans =
    1.0000
```

Per conoscere la lunghezza di un vettore `v` si usa il comando `length`

```
>> length(v)
ans =
     5
```

Si possono richiamare anche più posizioni di un vettore contemporaneamente, indicando invece che un solo indice un vettore di indici

```
>> v([1 5 3])
ans =
     0     1.0000     4.0000
```

Assegnazione di variabili vettoriali

In questo modo è possibile riassegnare più componenti di un vettore con un comando unico. Con

```
>> v([1 3 5])=-1
```

```
v =
```

```
-1.0000    0.2500   -1.0000    0.7500   -1.0000
```

pongo tutte le componenti selezionate uguali ad uno scalare mentre con

```
>> v([2 4])=[0 0]
```

```
v =
```

```
-1.0000    0.0000   -1.0000    0.0000   -1.0000
```

pongo ciascuna delle componenti selezionate uguale alle componenti del vettore a destra dell'uguale (questo vettore deve avere la stessa dimensione del vettore degli indici).

Operazioni fra vettori

Vediamo quali operazioni con i vettori matlab mette a disposizione: la più facile è la **trasposizione**, che permette di passare da un vettore riga ad uno colonna e si realizza aggiungendo `'` a ciò che si vuole trasporre.

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

```
>> u'
```

```
ans =
```

```
1
```

```
2
```

```
3
```

Poi ci sono **somma ed alla sottrazione**, che sono possibili tra vettori di uguale lunghezza e tipo (cioè entrambi riga o entrambi colonna)

```
>> u=[1 2 -1];
```

```
>> v=[-1 0 2];
```

```
>> u+v
```

```
ans =
```

```
0
```

```
2
```

```
1
```

Operazioni fra vettori

Somma e sottrazione si possono fare anche tra uno scalare ed un vettore

```
>> 2+[-1 1 3]
ans =          % lo scalare viene sommato (o sottratto)
    1 3 5      % a tutte le componenti del vettore
```

Per il **prodotto** si usa il simbolo $*$: si può eseguire un prodotto tra uno scalare ed un vettore

```
>> 2*[1 0 1];
ans =
    2    0    2
```

Suggerimento

Moltiplicando per 0 una vettore è possibile crearne velocemente un altro tutto nullo della stessa dimensione. Esempio

```
>> v=[1 2 3];
>> w=0*v
w =
    0    0    0
```

Operazioni fra vettori

Il **prodotto** è anche definito tra vettori della stessa lunghezza, a patto che uno sia un vettore riga e l'altro un vettore colonna. In particolare se il primo vettore è riga e l'altro colonna ottengo il **prodotto scalare**

```
>> [1 4 2]*[2;-1;0]
ans =
    -2
```

mentre se moltiplico un vettore colonna per un vettore riga trovo una matrice quadrata, per esempio

```
>> [2;-1;0]*[1 4 2]
ans =
     2     8     4
    -1    -4    -2
     0     0     0
```

Operazioni componente per componente

Le operazioni viste in precedenza sono utili per eseguire calcoli di algebra lineare, tuttavia non sono di aiuto in un problema come il seguente. Supponiamo di voler calcolare la funzione

$$y = x^4 \sin(x)$$

in molti punti di un intervallo (per esempio $[0, 1]$) su cui è definita: questo può essere necessario per esempio per poter disegnare la funzione. Per fare questo si può creare un vettore x con il comando

```
>> x=linspace(0,1,100);
```

A questo punto non posso usare le operazioni descritte in precedenza per calcolare in un colpo solo la funzione in tutti i punti di x , dato che per esempio l'elevamento a potenza $\boxed{\wedge}$ non è definito per i vettori ed il prodotto $\boxed{*}$ non è definito tra vettori dello stesso tipo. Matlab offre però anche un'altra serie di operazioni tra vettori, che agiscono componente per componente.

Operazioni componente per componente

La prima è il **prodotto** `.*` che agisce tra vettori di uguale lunghezza e dello stesso tipo (quindi o entrambi riga o entrambi colonna), come

```
>> u=[1 2 -1];  
>> v=[0 4 2];  
>> u.*v    % in questo caso il prodotto * non è definito  
ans =  
    0     8    -2
```

Perfettamente analoga è la **divisione** `./`, anch'essa definita tra vettori di uguale dimensione e tipo, come

```
>> v=[1 2 3];  
>> w=[4 -1 -2];  
ans =  
    0.2500    -2.0000   -1.5000
```

Operazioni componente per componente

L'ultima operazione componente per componente è l'**elevamento a potenza** `.^` che può agire in due modi. La modalità più usata permette di elevare tutte le componenti di un vettore allo stesso esponente scalare

```
>> v=[1:4];  
>> v.^3  
ans =  
    1     8    27    64
```

l'altra lavora tra vettori di uguali dimensioni e tipo in modo che ogni elemento del primo venga elevato all'elemento del secondo nella stessa posizione

```
>> v=[-1 3 5];  
>> w=[2 3 4];  
>> v.^w  
ans =  
    1    27   625
```

Operazioni componente per componente

Nota

Non esistono operazioni $\cdot +$ e $\cdot -$ dato che $+$ e $-$ sono già operazioni componente per componente. Anche per il prodotto tra uno scalare ed un vettore non è necessario usare il punto, dato che già agisce su ogni componente.

Attenzione

Come più volte sottolineato, le operazioni che agiscono componente per componente ($+$, $-$, $\cdot *$, $\cdot /$, $\cdot ^$) sono definite per vettori che, oltre ad avere la stessa lunghezza, devono essere anche dello **stesso tipo**, quindi o entrambi riga o entrambi colonna. Nel caso in cui si cerchi, per esempio di sommare

```
>> v=[1 2];  
>> w=[3;4];  
>> v+w
```

si otterrà un messaggio di errore

Le funzioni, ripresa

Ora è possibile calcolare la funzione

$$y = x^4 \sin(x)$$

sul vettore x attraverso l'espressione

```
>> y=x.^4.*sin(x)
```

Come visto nell'esempio appena concluso, le funzioni scientifiche prese in esame in precedenza per gli scalari lavorano in realtà componente per componente anche sui vettori. Altri esempio sono dati da

```
>> x=0:0.1:1;
```

```
>> exp(x)
```

che calcola in un colpo solo su tutti gli elementi di x la funzione esponenziale oppure

```
>> y=sin(x).*exp(x)-1
```

che assegna ad y l'esito della valutazione. Si noti come y **non sia una funzione** ma un vettore.

Definizione di nuove funzioni

In Matlab è possibile anche definire nuove funzioni, da utilizzare come quelle predefinite. Un modo è quello di utilizzare il comando **inline** la cui sintassi è la seguente

$$\text{nomefunzione} = \text{inline}(\text{'definizione'}, [\text{'elenco variabili'}])$$

Per definire la funzione $f(x) = xe^{-x^2}$ si digita

```
>> f=inline('x.*exp(-x.^ 2)', 'x')
f =
    % con ' ' in matlab si delimitano le
    Inline function:    % stringhe di testo
    f(x) = x.*exp(x.^2)
```

Controllando nella finestra di workspace si può notare come **f** sia un oggetto di tipo **inline**, diverso da quelli presi in esame sinora. Nel caso in cui la funzione sia di una sola variabile non è necessario specificare l'elenco variabili, per cui si avrebbe dato lo stesso risultato anche

```
>> f=inline('x.*exp(-x.^ 2)')
f =
    Inline function:
    f(x) = x.*exp(x.^2)
```

Definizione di nuove funzioni

Un altro modo è attraverso il simbolo di puntatore `@`, con la seguente sintassi

nomefunzione = @(elenco variabili) definizione

Per la funzione di prima si avrebbe

```
>> g=@(x) x.*exp(x.^2) % questa sintassi è disponibile solo  
g = % a partire dal matlab 7  
    @(x) x.*exp(x.^2)
```

Ci sono alcune differenze tecniche tra i due modi, che però in generale sono intercambiabili. Nel seguito si adotterà `inline`. Una volta definita una funzione, la si può utilizzare come tutte le predefinite. Per calcolare quanto vale $x \cdot e^{-x^2}$ su $[0, 1]$ ad intervalli di 0.1 basta scrivere

```
>> x=0:0.1:1;  
>> f(x)
```

Definizione di nuove funzioni

Attenzione!

Negli esempi precedenti le operazioni di prodotto e di elevamento a potenza sono state definite usando il punto. Questo, si ricorda è il modo con cui matlab permette di valutare le funzioni su dei vettori. Se si definisce

```
>> h=inline('x*exp(-x^ 2')
```

e si prova a calcolare

```
>> x=0:0.1:1;
```

```
>> f(x)
```

si ottiene un messaggio di errore. In generale nelle funzioni si vorrà sempre che siano calcolabili su vettori e quindi i simboli matematici $*$, $/$, $^$ saranno usati nella modalità $.*$, $./$, $.^$. Solo nel caso in cui si vogliono riprodurre le operazioni dell'algebra lineare (ma in genere nelle funzioni non succede) si dovranno usare $*$, $/$, $^$ senza il punto.

Grafica

Matlab offre moltissime possibilità di disegno sia sul piano (`help graph2d`), sia nello spazio (`help graph3d`), oltre a numerose funzioni per elaborare immagini e per creare grafici particolari (`help specgraph`). Un primo modo per disegnare una funzione è attraverso il comando **fplot**, che ha la sintassi

```
fplot('Stringa della funzione',[primo estremo, secondo estremo])
```

Per esempio per disegnare $e^x \sin(x)$ su $[-\pi, \pi]$ si usa

```
>> fplot('exp(x)*sin(x)',[-pi pi])
```

Dato che matlab non è un linguaggio simbolico, il comando `fplot` in realtà agisce in questo modo

- 1 Costuisce un vettore di punti compresi tra $-\pi$ e π
- 2 Valuta la funzione in questi punti
- 3 Richiama un'altra funzione che disegna i punti sul piano collegandoli con linee

Inoltre è molto “indulgente”, non richiedendo espressamente tutte le operazioni col punto necessarie.

Grafica

Il comando che `fplot` richiama per disegnare è **plot**, la cui sintassi è

```
plot(punti x,punti y,['stringa opzioni'])
```

Il grafico dell'esempio precedente può essere riprodotto seguendo i tre punti che anche il comando `fplot` utilizza per disegnare.

```
>> x=linspace(-pi,pi,10); % linspace si usa molto nei plot
>> f=inline('exp(x).*sin(x)');
>> plot(x,f(x))           % dopo plot il ; è ininfluente
```

Come si può vedere il grafico non è molto preciso, dato che è costruito usando solo 10 punti. Infittendo il dominio si migliora il risultato

```
>> x=linspace(-pi,pi,100);
>> plot(x,f(x))
```

Grafica

È possibile disegnare con colori diversi o con tratti di linea differenti aggiungendo una stringa di opzioni. Per esempio per disegnare in rosso si usa la lettera `r` mentre per disegnare linee puntinate si usa `:`, provare per esempio

```
>> plot(x,f(x),'r:')
```

È possibile poi disegnare dei simboli nei punti definiti, per esempio

```
>> plot(x,f(x),'g--*')
```

disegna in verde (`g=green`), con una linea tratteggiata (`--`), disegnando degli asterischi (`*`) in corrispondenza dei punti

Grafica

Per ottenere una panoramica di tutte le opzioni che si possono passare a matlab conviene ricorrere ad `help plot`.

Nota

Per disegnare un grafico basta che le variabili che contengono i punti delle ascisse e delle ordinate siano due vettori di uguale dimensione. Se per qualche ragione ho già calcolato il valore di una funzione su un vettore, per esempio con il comando

```
>> y=f(x);
```

è possibile disegnare usando direttamente il vettore `y`

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

È bene ricordare che se `x` cambia, `y` rimane quello vecchio finché non viene ricalcolato.

È sbagliata invece la sintassi

```
>> plot(x,f) % errore, ci vuole f(x)
```

dato che `f` non è un vettore.

Grafica

Plot permette di tracciare più disegni sulla stessa figura, per esempio

```
>> x=linspace(0,pi,100);  
>> f=inline('x.*sin(x)');  
>> g=inline('x.*cos(x)');  
>> plot(x,f(x),x,g(x))
```

Matlab provvede ad usare colori diversi per le diverse linee

Nota

Se si disegnano più grafici con lo stesso comando plot, alla fine di ogni coppia $x, f(x)$ si può specificare una stringa di opzioni. Quello che è importante notare è che se il vettore del dominio è lo stesso per più grafici, va comunque ripetuto. Nel caso precedente sarebbe stato sbagliato dare

```
>> plot(x,f(x),g(x))
```

Grafica

Un'altra alternativa per sovrapporre grafici è data dal comando **hold on/off** che funziona nel seguente modo:

- 1 si disegna il primo grafico
- 2 si da' il comando `hold on`
- 3 si disegnano tutti i grafici successivi
- 4 si da' il comando `hold off`

Per esempio per ottenere lo stesso risultato di prima si può scrivere

```
>> plot(x,f(x))  
>> hold on  
>> plot(x,g(x),'g') % bisogna cambiare il colore "a mano"
```

per poi dare il comando `hold off` per “rilasciare” la figura.

Grafica

Se invece si vuole che i grafici compaiano su figure diverse si usa il comando **figure(n)** che ha due funzioni

- se non esiste, apre la finestra `n`
- rende attiva la finestra `n`

Provare, dopo aver chiuso tutte le finestre di disegno eventualmente aperte, con

```
>> plot(x,f(x))
>> figure(2)
>> plot(x,g(x),'g')
```

Grafica

Matlab fornisce anche numerosi comandi per la gestione del grafico e per inserire scritte o informazioni. I principali sono

Comandi grafici

Comando	Descrizione
<code>title('Titolo')</code>	Aggiunge un titolo alla finestra
<code>grid on/off</code>	Aggiunge/toglie una griglia
<code>axis([xmin xmax ymin ymax])</code>	Ridimensiona gli assi
<code>xlabel('testo')</code>	Aggiunge una stringa all'asse X
<code>ylabel('testo')</code>	Aggiunge una stringa all'asse Y

Grafica

Un comando molto usato è **legend** che permette di aggiungere una legenda alla figura, molto utile nel caso in cui siano presenti più grafici. La sua sintassi è

```
legend('Descrizione grafico 1',['Descrizione grafico 2',...])
```

Vediamo come funziona (conviene chiudere tutte le figure aperte)

```
>> x=linspace(0,1,100);  
>> plot(x,x.^2,x,x.^3,x,x.^4)  
>> legend('x^2','x^3','x^4')
```

Chiaramente spetta all'utente elencare con `legend` i grafici seguendo lo stesso ordine con cui sono stati dati nel comando `plot`.

Ulteriori opzioni per modificare i grafici sono presenti poi nei menu della figura e nelle icone della toolbar. In particolare è possibile esportare le figure salvandole nei principali formati disponibili (.jpg,.eps,..)