UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
INDUSTRIALE

DIPARTIMENTO
MATEMATICA
DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

MATLAB STEP BY STEP

Materiale realizzato da Michela Redivo Zaglia

con il contributo di E. Bachini, L. Bruni, W. Erb, A. Larese, F. Piazzon

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

DIPARTIMENTO MATEMATICA

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Laboratorio di Calcolo Numerico LAB 2

Creare vettori, funzioni e grafici

Docenti: E. Bachini, L. Bruni

Email: elena.bachini@unipd.it Email: bruni@math.unipd.it

13 marzo 2024

Outline

Vettori riga e colonna

Possiamo definire un vettore scrivendone le componenti tra parentesi quadre. In funzione del separatore tra le componenti otterremo

- un vettore riga se separiamo le componenti con uno **spazio** (o una virgola)

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

- un vettore colonna se separiamo le componenti con **un punto e virgola**

```
>> u = [4; 5; 6]
u =
     4
     5
     6
```

Trasposto di un vettore

Possiamo anche trasformare un vettore riga nel corrispettivo vettore colonna (o viceversa). Il simbolo di trasposizione è l'**apostrofo**.

```
>> v = [ 1 2 3 ]
v =
     1     2     3
>> v = v'  % trasformo v in un vettore colonna
           % calcolandone il suo TRASPOSTO
v =
     1
     2
     3
```

Possiamo anche inizializzare un vettore (privo di componenti), basta eseguire il comando del tipo

```
>> w = []
w =
     []
```

Crea un vettore numerico di dimensioni 0 righe e 0 colonne.

Dimensione di un vettore

Esistono due comandi per determinare il numero di componenti di un vettore

$$v = (v_1, \ldots, v_n)$$

- il comando `length` che riporta quante componenti ha un vettore (sia riga che colonna),
- il comando `size` che determina la dimensione di un vettore e, a differenza di `length`, chiarisce se è di tipo riga o colonna, in quanto restituisce, nell'ordine, il numero di righe ed il numero di colonne.

```
>> vettore_colonna=[2;5;1] % vettore colonna con 3 componenti
                           % e dimensione 3 righe x 1 colonna
vettore_colonna =
     2
     5
     1
>> length(vettore_colonna)
ans =
     3
>> size(vettore_colonna)
ans =
     3     1
>>
```

Vettore nullo e unitario

Può essere utile nelle inizializzazioni, definire un vettore (o una matrice) con componenti tutte uguali a zero o ad uno

- Vettore (o matrice) di zeri: `zeros(nrighe,ncolonne)`,

```
>> zeros(5,1) % definisce un vettore colonna nullo avente
              % 5 righe e 1 colonna
ans =
     0
     0
     0
     0
     0
>> zeros(1,5) % definisce un vettore riga nullo avente
              % 1 riga e 5 colonne
ans =
     0     0     0     0     0
```

- Vettore (o matrice) di 1: `ones(nrighe,ncolonne)`

```
>> ones(1,6)
ans =
     1     1     1     1     1     1
```

Vettori equispaziati

I vettori riga $v = (v_1, v_2, \ldots, v_m)$ con componenti equispaziate ovvero tali che

$$v_{k+1} - v_k = h, \text{ per } k = 1, \ldots, m-1,$$

sono particolarmente facili da definire.

Questi vettori sono molto usati, si pensi ad esempio a quando si vuole valutare una funzione $f(x)$ al variare di x in un intervallo $[a, b]$, estremi inclusi.
Noti gli estremi a, b dell'intervallo in cui voglio valutare la funzione, ci sono due possibili casi: conosco o voglio fissare il passo (detto anche incremento) h (ovvero il valore $h = v_{k+1} - v_k$), oppure conosco o voglio fissare il numero di valori m in cui valutare la funzione.

Vettori equispaziati (definito il passo)

Se conosco il passo h potrò usare il comando

$$\texttt{v = a : h : b}$$

che genera un vettore (riga!) la cui prima componente è a e la cui ultima componente, chiamiamola l'm-esima, è

$$v_m = a + (m-1) \cdot h$$

con $v_m \leq b$

```
>> v= 2:4:10  % vettore riga con primo valore 2, ultimo valore 10
              % generato con passo 4
v =
     2     6    10
>> v= 10:-4:2 % vettore riga con primo valore 10, ultimo valore 2
              % generato con passo -4
v =
    10     6     2
```

Se il passo è unitario ($h = 1$) posso scrivere semplicemente `v = a:b`, senza indicare il passo.

Vettori equispaziati (definito il passo)

Non è detto che il valore di b venga assegnato all'ultima componente.

```
>> v=2:3:10 % vettore riga con primo valore 2, valore massimo 10,
            % ultima componente 8 < 10, generato con passo 3
v =
     2     5     8
>>
```

```
>> v = 0:pi/2:2*pi % vettore riga con primo valore 0,
                   % generato con passo pi/2
v =
         0    1.5708    3.1416    4.7124    6.2832
>>
```

```
>> v = 0.3:0.35:2.0 % vettore riga generato con passo 0.35

v =
    0.3000    0.6500    1.0000    1.3500    1.7000
```

Vettori equispaziati (definito il numero di componenti)

Se conosco il numero di componenti che voglio abbia il vettore posso usare

```
u = linspace(a,b,m)
```

genera un vettore riga $u = (u_1, \ldots, u_m)$ con m componenti equispaziate, che comincia da a e finisce con b, ovvero

$$u_k = a + (k-1) \cdot \frac{b-a}{m-1}, \quad k = 1, \ldots, m.$$

```
>> v=linspace(2,10,5) % vettore con primo valore 2,
                      % ultimo valore 10, e con 5 componenti
v =
     2     4     6     8    10
>>
```

- Se non specifico il numero di componenti il Matlab crea di default un vettore con 100 componenti.

Accesso alle componenti

Sia $v = (v_1, \ldots, v_m)$. Posso accedere ad una o piu componenti indicando tra parentesi la o le componenti $v(i)$ che desidero considerare (`i` deve essere un numero intero positivo minore di m).

```
>> v = [6 7 8 9];
>> v(2)    % accedo alla seconda componente del vettore
ans =
     7
```

o $v([i \quad j \quad k])$ per molteplici componenti di un vettore riga

```
>> v([3 4]) % Sono necessarie parentesi quadre
            % (restituisce un vettore riga)
ans =
     8     9
```

o $v([i\,;\, j;\, k])$ per molteplici componenti di un vettore colonna

```
>> v = [4; 5; 6;];
>> v([1; 2]) % Se il vettore é colonna, dobbiamo
             % separare le componenti con ";"
             % (restituisce un vettore colonna)
ans =
     4
     5
```

Accesso all'ultima componente di un vettore

Si osservi che per selezionare l'ultima riga o colonna di un vettore `v` è sufficiente scrivere `v(end)`, ovvero utilizzare la keyword `end`, senza necessariamente conoscere quale sia l'indice dell'ultima componente.

```
>> v = [3 4 5];
>> v(end)

ans =
     5
```

Concatenazione

E' possibile definire un vettore tramite concatenazione.

```
>> u1 = [1  2];
>> u2 = [4  5  6];
>> [u1  u2  u1]
ans =
     1     2     4     5     6     1     2
```

Risulta molto utile quando desidero aggiungere una componente alla fine di un vettore.

```
>> u = [1  2];
>> u = [u  6]

ans =
     1     2     6
```

attenzione alle dimensioni!

```
>> u1 = [1  2];      % vettore riga 1 x 2
>> u2 = [4; 5; 6]; % vettore colonna 3 x 1
>> [u1  u2]
Error using horzcat
Dimensions of arrays being concatenated are not consistent.
```

Operazioni aritmetiche componente per componente

Le seguenti operazioni (chiamate anche operazioni puntuali) tra due vettori dello stesso tipo (riga o colonna) con lo stesso numero di componenti, oppure tra due matrici aventi le stesse dimensioni, producono, rispettivamente, un vettore dello stesso tipo, con lo stesso numero di componenti, oppure una matrice con le stesse dimensioni.

$+$	addizione
$-$	sottrazione
$.*$	prodotto
$./$	divisione
$.\wedge$	potenza

- Il punto prima di $*$, $/$ e $\wedge$ indica che l'operazione è eseguita componente a componente.
  N.B. non serve mettere il punto quando si vogliono fare operazioni con variabili o costanti scalari!

Operazioni con i vettori

Siano $u = (u_1, \ldots, u_n)$ e $v = (v_1, \ldots, v_n)$ vettori della **stessa dimensione** (entrambi riga o entrambi colonna e con lo stesso numero di componenti) ed s uno scalare.

- `c=s*u`, prodotto dello scalare s con il vettore u,

$$c_1 = s \cdot u_1, \; c_2 = s \cdot u_2, \ldots, \; c_n = s \cdot u_n$$

- `c=u'` trasposta del vettore u,
- `c=u+v` somma del vettore u col vettore v

$$c_1 = u_1 + v_1, \; c_2 = u_2 + v_2, \ldots, \; c_n = u_n + v_n$$

- `c=u-v` differenza tra il vettore u e il vettore v

$$c_1 = u_1 - v_1, \; c_2 = u_2 - v_2, \ldots, \; c_n = u_n - v_n$$

Operazioni componente a componente

Attenzione!

Eseguibili solo se `size(u)=size(v)`

- `c=u.*v`, prodotto **componente a componente** del vettore u col vettore v

$$c_1 = u_1 \cdot v_1, \quad c_2 = u_2 \cdot v_2, \ldots, \quad c_n = u_n \cdot v_n$$

- `c=u./v` divisione **componente a componente** del vettore u col vettore v,

$$c_1 = \frac{u_1}{v_1}, \quad c_2 = \frac{u_2}{v_2}, \ldots, \quad c_n = \frac{u_n}{v_n}$$

- `c=u.^k` potenza k-sima **componente a componente** del vettore u con

$$c_1 = u_1^k, \quad c_2 = u_2^k, \ldots, \quad c_n = u_n^k$$

Tipi di function Matlab

Ci sono due tipologie di function Matlab:

- anonymous functions [trattate in particolare in questa lezione].
  Normalmente si usano per calcoli matematici relativamente semplici.
  - Si possono definire in uno script o nella command window,
  - Normalmente rappresentano delle espressioni matematiche.

- m-files di tipo function [trattate in particolare nella prossima lezione].
  Nascono per algoritmi più complessi.
  - Non si possono definire (di norma) negli script e neppure nella command window (vengono scritte e memorizzare in un file a parte).

Cenni sulle function m-file

La **function m-file** è lo strumento principale con il quale vengono tradotti algoritmi o parte di essi, in modo da poter essere facilmente riutilizzati in altri esperimenti, essendo svincolati dai nomi di variabili in esso definite e contenute.

- Prevede parametri di ingresso, di uscita, e variabili locali.
- La function implementa il concetto di **black box** in cui il **passaggio di informazioni tra function e codice chiamante avviene solo attraverso i parametri di ingresso e di uscita**

Cenni sull'utilizzo di function m-file

Per sua natura una function m-file non può essere "eseguita" (come si può invece fare con uno script), va invece chiamata passandole l'input con la sintassi

$$[out_1, out_2, \ldots, out_m] = nomeFunzione(in_1, in_2, \ldots, in_n)$$

Esempio usando una function m-file predefinita nel Matlab (comando `sort`)

```
>> u = [2 7 -2 3 1 5];
>> [u_ord,ind]=sort(u)

u_ord =

    -2     1     2     3     5     7


ind =

     3     5     1     4     6     2

>>
```

Alcune funzioni elementari predefinite (built-in)

In Matlab ci sono una serie di funzioni m-file predefinite che sono molto utili. Ad esempio:

`abs`	valore assoluto
`sin`	seno
`cos`	coseno
`tan`	tangente
`cot`	cotangente
`asin`	arco seno
`acos`	arco coseno
`atan`	arco tangente
`sinh`	seno iperbolico
`cosh`	coseno iperbolico
`tanh`	tangente iperbolica
`asinh`	arco seno iperbolico
`acosh`	arco coseno iperbolico
`atanh`	arco tangente iperbolica
`sqrt`	radice quadrata
`exp`	esponenziale
`log2`	logaritmo base 2
`log10`	logaritmo base 10
`log`	logaritmo naturale
`fix`	arrotondamento verso 0
`round`	arrotondamento verso l'intero più vicino
`floor`	arrotondamento verso $-\infty$
`ceil`	arrotondamento verso $+\infty$
`sign`	segno
`rem`	resto della divisione

NB: esistono functions con più output.

Come definire le anonymous functions

All'interno di uno script o nella command window per definire una anonymous function ad una variabile si usa la sintassi

<functionName> = @(<varName>) espressione

```
>> f=@(x) x.^2
f =
  function_handle with value:
    @(x)x.^2
>> f(2)
ans =
     4
```

Ma si può anche definire una funzione a più variabili, che accetta più input e restituisce più output. Esempio:

```
>> g=@(x,y,z) [x+y+z, x^2*z-y];
>> g(1,2,3)

ans =

     6     1
```

Uso di str2func

A volte può essere utile definire l'espressione di una funzione come stringa per poi convertirla in una anonymous function tramite la built-in function str2func

<funcName> = str2func('@(<varName>) <espressione>')

```
>> str='@(x) x^2-2+sin(x)';
>> f=str2func(str)
f =
  function_handle with value:
    @(x)x^2-2+sin(x)
>> f(1)

ans =

   -0.1585
```

Nel workspace le anonymous function sono indicate appartenere alla classe function_handle.

Creare funzioni vettorializzate

Spesso (ed in particolare per **costruire grafici**) risulta utile creare delle functions che possano essere vettorializzate, i.e., *qualora valutate su un vettore di input restituiscono in output il vettore delle valutazioni effettuate componente per componente*.

Esempio:

```
>> f=@(x) sin(pi.*x);
>> f(linspace(0,1,5))
ans =
         0    0.7071    1.0000    0.7071    0.0000
```

Esempio con errore (errato utilizzo dell'operatore nella definizione):

```
>> f=@(x)x^2;
>> f([1 2 3])
Error using  ^  (line 51)
Incorrect dimensions for raising a matrix to a power.
Check that the matrix is square and the power is a scalar.
To perform elementwise matrix powers, use '.^'.

Error in @(x)x^2
```

Come creare un grafico

Per realizzare un grafico su un piano cartesiano, il modo più semplice consiste nell'utilizzo del comando

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

in cui x è il vettore delle ascisse (ovvero contiene le ascisse dei punti che si desidera connettere) e y quello delle ordinate.
I due vettori devono avere lo stesso numero di componenti!

Attenzione!

Matlab non disegna il "grafico di una funzione", ma visualizza solo la spezzata che unisce i punti di cui si sono fornite ascisse ed ordinate.

Nota.

- *Per assicurarci di cancellare precedenti grafici, si usa il comando* `clf` *prima di plot*

Opzioni del comando `plot`

Possiamo ovviamente decidere di realizzare il grafico in vari modi (tipo e colore delle linee, rappresentazione dei punti connessi con dei simboli, ...) aggiungendo un terzo parametro (stringa). Guardiamo l'help.

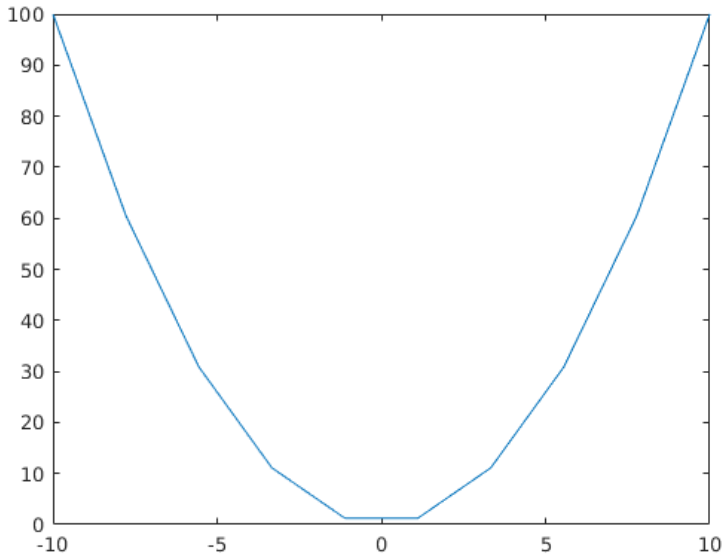
```
>> help plot
  plot   Linear plot.
    plot(X,Y) plots vector Y versus vector X. [...]
Various line types, plot symbols and colors may be obtained with
    plot(X,Y,S) where S is a character string made from one element
    from any or all the following 3 columns:

    b     blue          .     point              -     solid
    g     green         o     circle             :     dotted
    r     red           x     x-mark             -.    dashdot
    c     cyan          +     plus               --     dashed
    m     magenta       *     star             (none)  no line
    y     yellow        s     square
    k     black         d     diamond
    w     white         v     triangle (down)
                        ^     triangle (up)
                        <     triangle (left)
                        >     triangle (right)
                        p     pentagram
                        h     hexagram
```

Esempio di `plot`

Per disegnare una parabola definiamo un vettore di dieci componenti tra -10 e -10 e valutiamo su di essi la funzione $f(x) = x^2$ per ottenere le ordinate.

```
>> x_plot = linspace(-10,10,10);
>> f =@(x) x.^2;
>> y = f(x_plot);
>> plot(x_plot,y)
```



Nota:

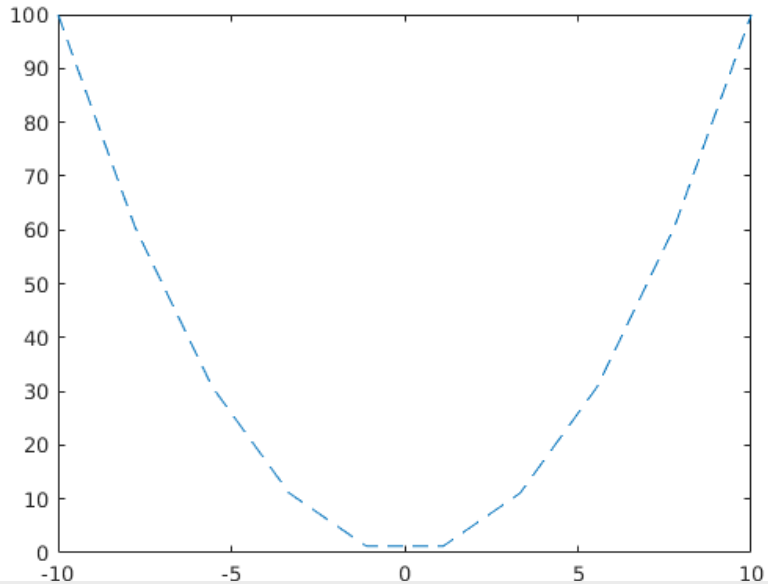
Come si può notare si vede una spezzata che unisce i punti forniti (10 punti sono troppo pochi!)

Tipi di linea

Possiamo decidere di cambiare il tipo di linea che unisce i punti scegliendo tra queste possibilità:

-	solid
:	dotted
- .	dashdot
- -	dashed
(none)	no line

```
>> xv = linspace(-10,10,10);
>> yv = xv.^2; % parabola
>> plot(xv,yv, '--')
```

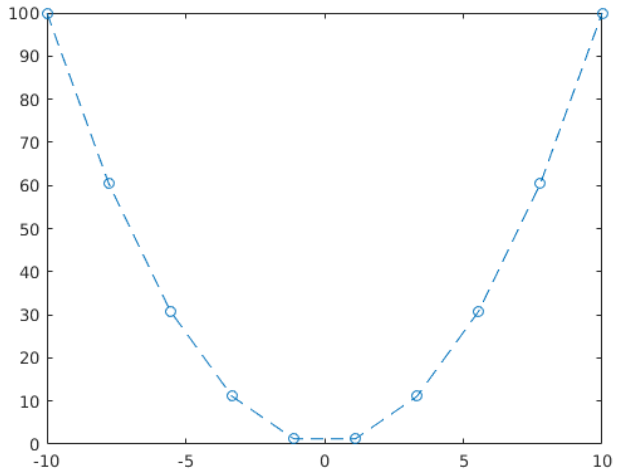


Tipi di marcatori

Possiamo decidere di cambiare il tipo di marcatore dei punti (di default non li indica)

.	point	*	star	$\wedge$	triangle (up)
o	circle	s	square	<	triangle (left)
x	x mark	d	diamond	>	triangle (right)
+	plus	v	triangle (down)	p	pentagram
h	hexagram				

```
>> xv = linspace(-10,10,10);
>> yv = xv.^2; % parabola
>> plot(xv,yv, '--o')
```

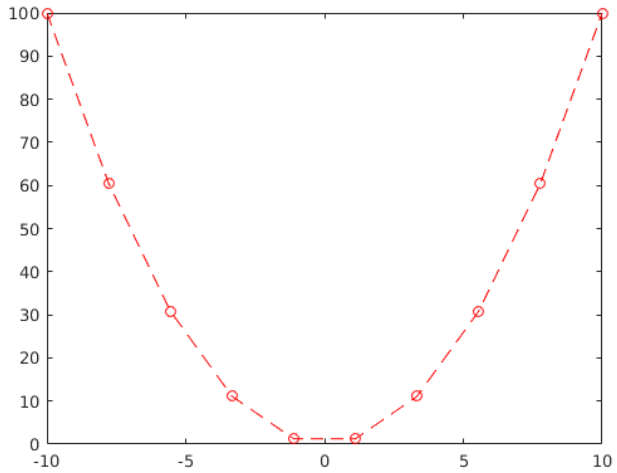


Colori delle linee e dei marcatori

Possiamo decidere di cambiare il colore della linea, dei marcatori dei punti, ecc...

b	blue	m	magenta
g	green	y	yellow
r	red	k	black
c	cyan	w	white

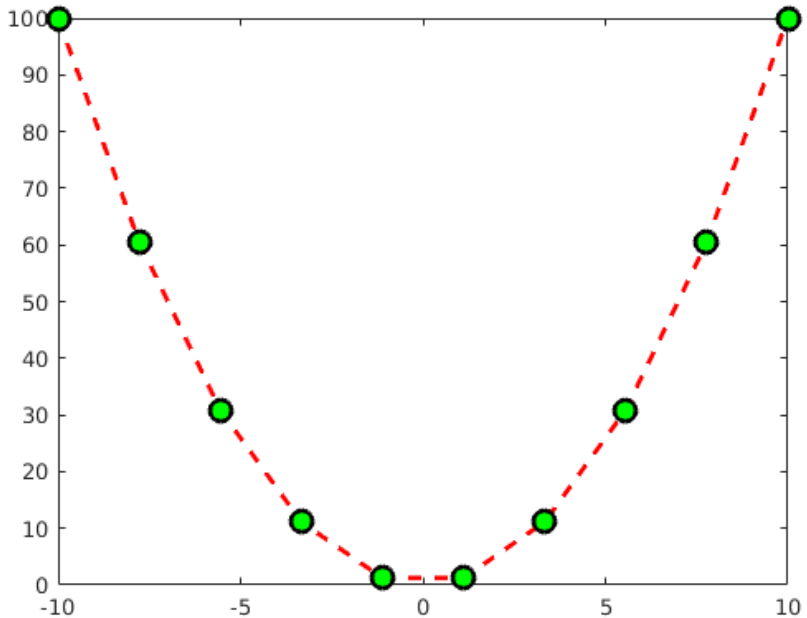
```
>> xv = linspace(-10,10,10);
>> yv = xv.^2; % parabola
>> plot(xv,yv, '--or')
```



Altre opzioni

Possiamo anche decidere lo spessore della linea, il contorno e il colore del marker separatamente dalla linea

```
>> xv = linspace(-10,10,10);
>> yv = xv.^2; % parabola
>> plot(xv,yv, '--or', 'LineWidth',2, 'MarkerEdgeColor','k', ...
        'MarkerFaceColor','g', 'MarkerSize',10)
```

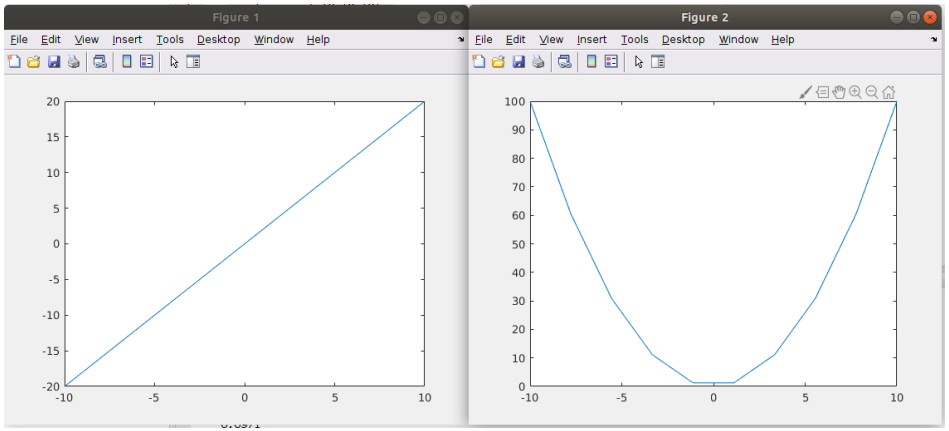


Grafici su finestre diverse

N.B. un comando `plot`, successivo ad un altro comando `plot`, ma senza usare il comando `figure`, usa la stessa finestra del grafico precedente (che viene perso).

Per disegnare più grafici e visualizzarli su finestre diverse, possiamo farlo inserendo ogni grafico in una figura a cui assegnamo un numero: si usa il comando `figure (n)`, con n intero maggiore o uguale ad 1.

```
>> xv  = linspace(-10,10,10);
>> yv  = 2*xv;  % retta passante per l'origine degli assi
>> yv1 = xv.^2; % parabola
>> figure(1)
>> plot(xv,yv)
>> figure(2)
>> plot(xv,zv)
```



Più grafici nella stessa figura

Un primo modo consiste nell'inserire in un unico comando `plot` le coppie di vettori ascisse-ordinate.

```
>> xv  = linspace(-10,10,10);
>> yv  = 2*xv;  % retta passante per l'origine degli assi
>> yv1 = xv.^2; % parabola
>> plot(xv,yv, xv,yv1)
```

Possiamo anche inserire le nostre preferenze per ogni curva:

```
>> plot(xv,yv,':r',xv,yv1,'-b*')
```

In alternativa, possiamo utilizzare il comando `hold on` ("mantieni" la finestra) che permette di dare più comandi `plot` separati, ma utilizzando la stessa finestra. Quando si termina di usare la stessa finestra bisogna dare il comando `hold off` ("rilascia" la finestra)

```
>> xv  = linspace(-10,10,10);
>> yv  = 2*xv;  % retta passante per l'origine degli assi
>> yv1 = xv.^2; % parabola
>> plot(xv,yv,'--bs','LineWidth',2)
>> hold on
>> plot(xv,yv1,':gv','LineWidth',1)
>> hold off
```

Titolo e legenda

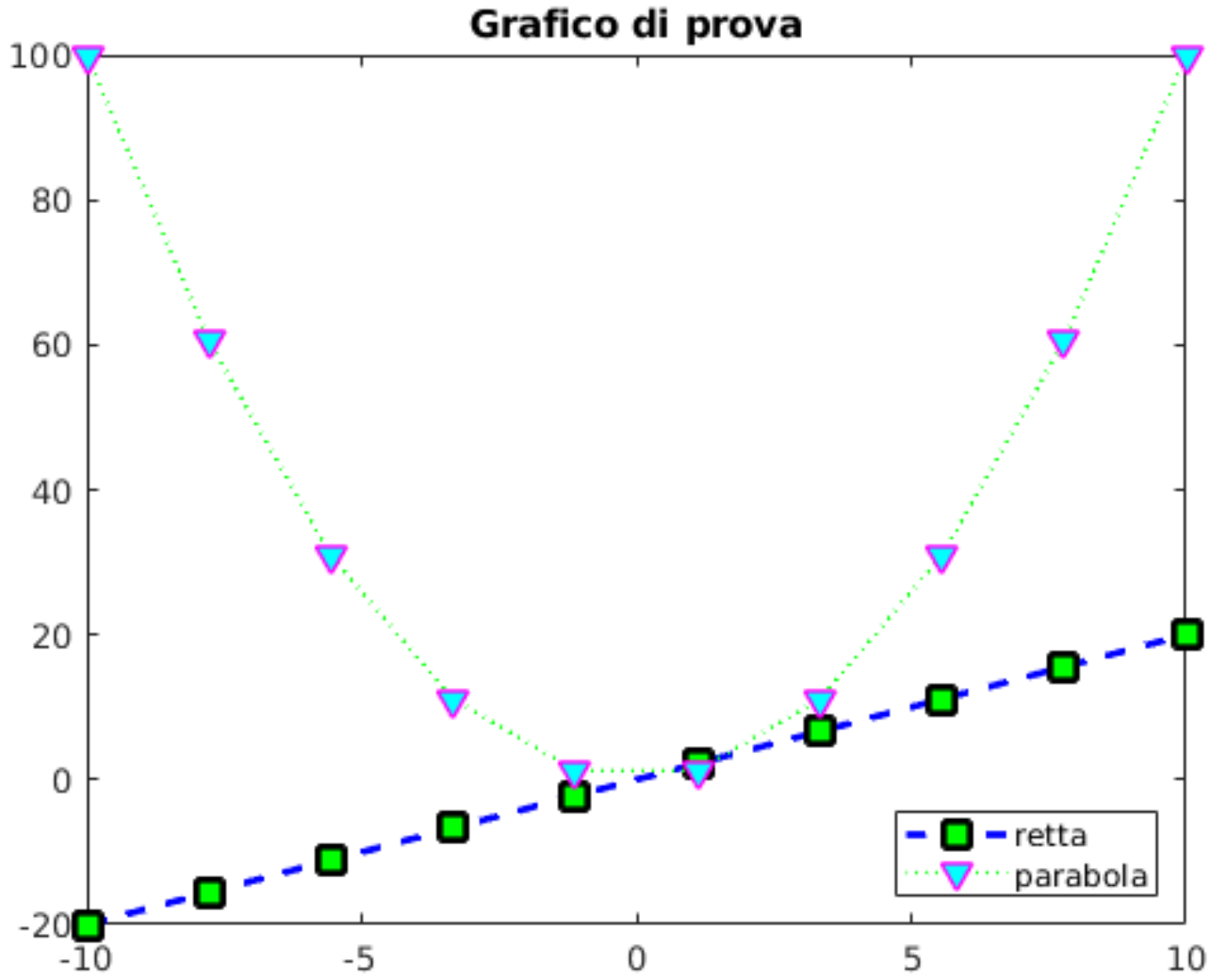
Se si vuole inserire un titolo basta usare il comando `title` **dopo** che il grafico è stato creato:

```
>> title('Grafico di prova')
```

Se si vuole inserire una legenda basta usare il comando `legend`. Bisogna fornire come parametri, una stringa per ogni "spezzata" tracciata (corrispondenza biunivoca), specificando, eventualmente, la posizione in cui si vuole inserire tale legenda (parametro 'Location' seguito da una virgola e da `'north'`, `'south'`, `'east'`, `'west'` o loro combinazioni, ad esempio `'northeast'`)

```
>> legend('retta','parabola','Location','southeast')
```

Titolo e legenda



Assi

Si possono aggiungere le etichette agli assi usando i comandi `xlabel` e `ylabel`

```
>> xlabel('x [m]')
>> ylabel('f(x) [m]')
```

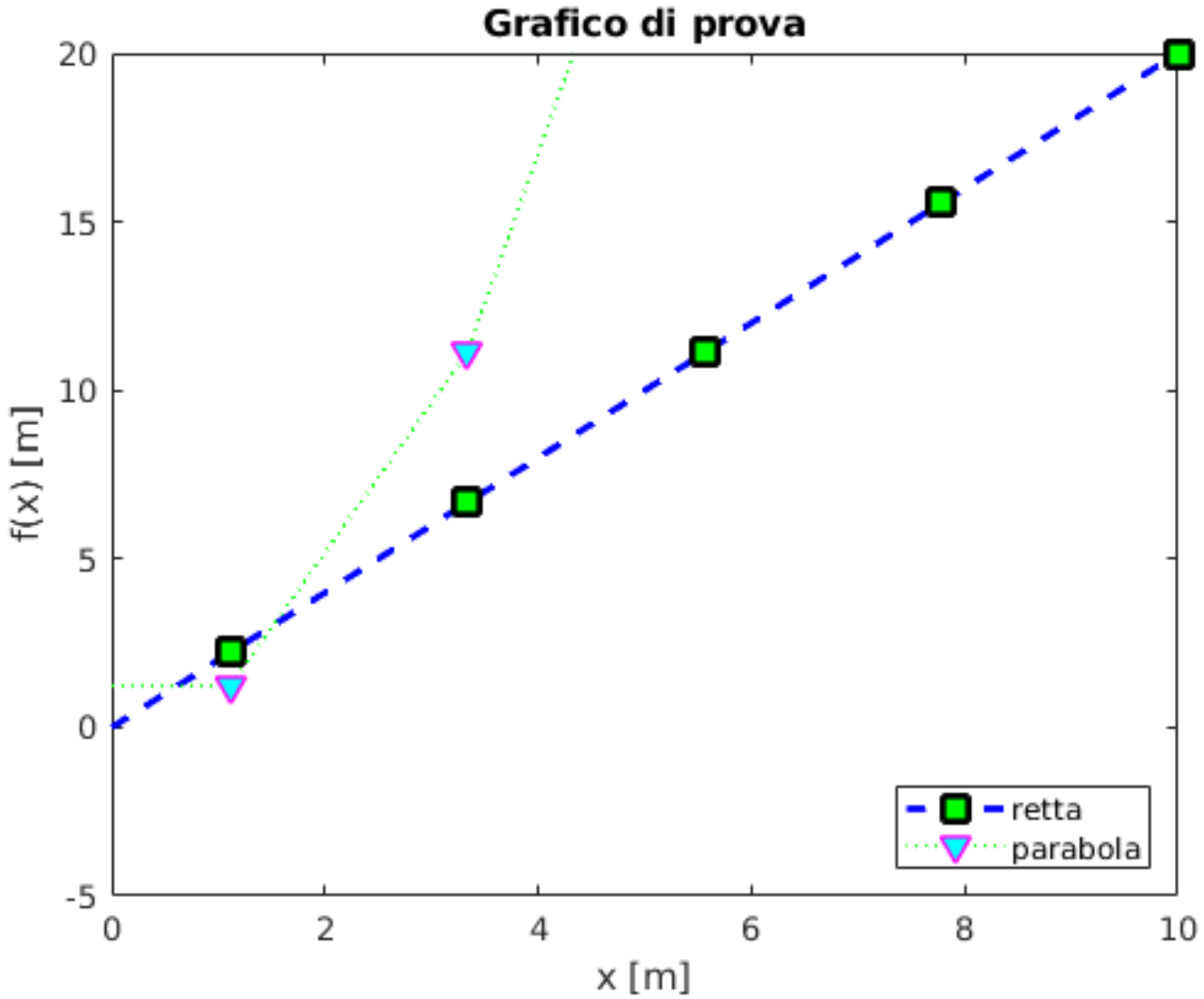
In base alle coordinate dei punti, Matlab decide automaticamente il rango da impostare per l'asse delle ascisse e quello delle ordinate.
E' però possibile impostare a proprio piacimento il valore massimo e minimo degli assi con il comando `axis([Xmin Xmax Ymin Ymax])`

```
>> axis([0.0 10.0 -5.0 20.0])
```

Nota:

Si noti nella figura successiva che tale ultima impostazione può impedire la rappresentazione di alcune spezzate e di alcuni punti.

Assi



Altri comandi per grafici

E' possibile inserire in una figura una stringa di caratteri posizionandola nella figura tramite il mouse. Si deve usare il comando `gtext` (si veda `doc gtext`).

```
>> gtext('il mio testo','r')
```

Inserisce nella figura la stringa 'il mio testo'
in rosso. Il posizionamento avviene tramite mouse nella figura corrente.

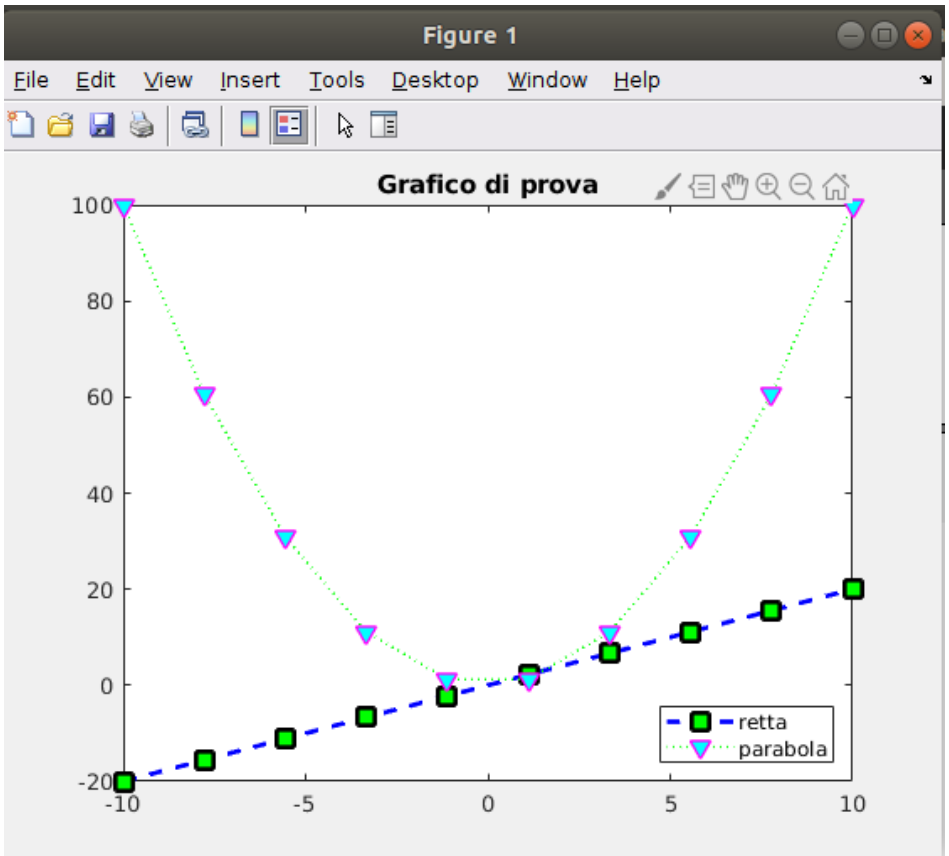
Talvolta vi è la necessità di realizzare dei grafici utilizzando per l'asse y (o l'asse x) la scala logaritmica (base 10). In tal caso, invece del comando `plot`, si deve usare il comando `semilogy` (o `semilogx`).

Nota:

Tutto quanto detto per il comando `plot` e gli altri comandi relativi alle rappresentazioni opzionali resta valido anche per i comandi `semilogy` e `semilogx`.

Menù opzioni figura

Ogni figura ha un proprio menù nella parte superiore che può essere utilizzato per salvare la figura in un file, utilizzando vari formati, e per impostare in modo diretto le caratteristiche desiderate.



I formati di salvataggio più noti ed usati sono:

- `.fig` (formato figura Matlab)
- `.eps`
- `.pdf`
- `.jpg`