Calcolo matriciale: usare la sintassi vettoriale

Evitare il più possibile i cicli `for` quando si fanno prodotti matrice–vettore o operazioni fra matrici in generale!

Data una matrice `A` ed un vettore `x`, calcolare il prodotto `b=A*x` direttamente con la sintassi di MatLab è molto più conveniente che implementare un doppio ciclo `for`. Per esempio, la lista di istruzioni

```
b = zeros(n,1);
for i = 1:n
    for j = 1:n
        b_for(i) = b_for(i) + A(i,j)*x(j);
    end
end
```

costa molto più di `b = A*x`, sopratutto a dimensioni alte.

Calcolo matriciale: usare la sintassi vettoriale

```
clear all; close all; clc

K = 50; n = 500;
t_for = zeros(K,1); t_mat = zeros(K,1);
for k = 1:K
    A = rand(n);
    x = rand(n,1);
    tic
    b_mat = A*x;
    t_mat(k) = toc;
    tic
    b_for = zeros(n,1);
    for i = 1:n
        for j = 1:n
            b_for(i) = b_for(i) + A(i,j)*x(j);
        end
    end
    t_for(k) = toc;
end
fprintf('Tempo medio con la sintassi di MatLab : %1.6f secs\n',...
mean(t_mat));
fprintf('Tempo medio usando un ciclo for       : %1.6f secs\n',...
mean(t_for));
```

Comandi utili: stringhe e caratteri

Una stringa è un vettore di caratteri. Esempio:

```
s = 'Malcom'
```

```
>> s(4)
ans =
c
```

Per accedere ad uno degli elementi si accede come nel caso dei vettori numerici.

```
>> double(s)
ans =
    77    97   108    99   111
        109
```

Il comando `double` restituisce il codice ASCII corrispondente.

```
>> char(65)
ans =
A
```

Il comando `char` restituisce il carattere ASCII relativo all'argomento passato.

```
>> t = ' Raynolds';
>> [s,t]
ans =
Malcom Raynolds
```

Due (o più) stringhe possono essere concatenate.

Comandi utili: find e isempty

```
A =
     0    -9    -1     6    -9
     3    -3    -2     8     7
    -9    -1    -7    -5     2
    -8     3     7    -6    -2
     8     6     9     2    -5
>> [I, J] = find(A > 8)
I =
     5
J =
     3
```

Il comando `find` consente di trovare gli indici degli elementi che soddisfano una particolare condizione.

```
A =
     9    -7    -1     2    -5
     3    -5    -4    -7    -7
    -9     3     5    -9     9
     0    -1    -5     3    -3
     0     2    -2     6    -1
>> [I, J] = find(A > 8)
I =
     1
     3
J =
     1
     5
```

Nel caso in cui la condizione richiesta sia soddisfatta da più di un elemento, i vettori `I` e `J` contengono gli indici di riga e colonna, rispettivamente.

Comandi utili: find e isempty

Può capitare che alcune variabili siano vuote (per esempio, quando in una matrice non ci sono elementi che siano maggiori di un certo valore assegnato). Il comando `isempty(x)` restituisce `TRUE` nel caso la variabile `x` sia vuota, FALSE in caso contrario. Per esempio, le istruzioni seguenti

```
A = fix(-10+20*rand(5))

[I,J] = find(A > 8)
if isempty(I)
    fprintf('Non ci sono elementi maggiori di 8\n')
end
```

possono produrre il seguente output

```
A =
    -3    -2    -4    -8    -7
    -6     6     5    -8     1
    -4     1     5     0     0
     2     0    -2     5    -9
     0     8     1     8    -3
I =
   Empty matrix: 0-by-1
J =
   Empty matrix: 0-by-1
>> Non ci sono elementi maggiori di 8
```


Strutture

I dati di tipo `struct` sono le classiche strutture comuni ai linguaggi di programmazione. Esse sono array multidimensionali con all'interno vari campi, che possono essere di vario tipo.

Creazione e riempimento di una struttura

Per esempio, se si vuole creare la struttura `S` con i campi `Nome`, `Cognome` e `Anno`, si procede nel seguente modo

```
% Si crea una struttura con i campi 'Nome', 'Cognome' ed 'Anno'
% inizializzati a valori dati (due stringhe e un numero)
>> S = struct('Nome',    'Cleve', ...
              'Cognome', 'Moler', ...
              'Anno',    1939);
```

Per accedere ad un campo della struttura si usa la notazione `nome_struttura.nome_campo`:

```
>> S.Nome
ans =
Cleve
```

Strutture

Per modificare il valore di un campo si utilizza la sintassi `nome_struttura.nome_campo = nuovo_valore`.

Modifica valore di campo

Per modificare i campi della struttura precedentemente create si usano i seguenti comandi:

```
>> S.Nome = 'Jack';
>> S.Cognome = 'Little';
>> S.Anno = 1956;
```

Èpossibile aggiungere dinamicamente campi a una struttura.

Aggiunta di campi

```
>> S.Titolo = 'Presidente';
```

Si può creare una struttura in maniera dinamica ex novo.

Strutture

Si possono creare vettori di strutture, sfruttando l'allocazione dinamica della memoria di Matlab.

Vettori di strutture

```
S = struct('Nome',{},... % crea una struttura vuota
           'Cognome',{},...
           'Anno',{});

S(1).Nome    = 'Frank';
S(1).Cognome = 'Herbert';
S(1).Anno    = 1920;

S(2).Nome    = 'Isaac';
S(2).Cognome = 'Asimov';
S(2).Anno    = 1920;

S(2) % visualizza i campi dell'elemento 2
```

Cell array

I `cell array` sono particolari tipi di dato che possono contenere dati di qualsiasi tipo. La creazione di questo tipo di dato segue una notazione *vettoriale*.

Creazione di cell array ed accesso ai suoi elementi

Nel seguente codice viene creato un cell array di 4 elementi, organizzati come una tabella a due righe e due colonne.

```
>> A = cell(2,2);    % Viene inizializato il cell array
>> A{1,1} = 'ciao'; % elemento di posto 1,1
>> A{1,2} = 4;       % elemento di posto 1,2
>> A{2,1} = [1,2];   % elemento di posto 2,1
>> A{2,2} = [4;3];   % elemento di posto 2,2
>> A                 % Struttura simile a quella delle matrici
A =
    'ciao'           [         4]
    [1x2 double]     [2x1 double]
>> A{1,2} % l'accesso avviene tramite le parentesi graffe
ans =
     4
>> A{2,2}
ans =
     4
     3
```


Gestione I/O

Per poter gestire in maniera ottimale gli input e gli output delle funzioni si possono usare i seguenti comandi: `varargin, nargin, varargout, nargout`.

- `varargin` gestisce le variabili in input, di cui a priori non si sa il numero;
- `nargin` gestisce il numero di dati in input;
- `varargout` gestisce le variabili in output, di cui a priori non si sa il numero;
- `nargout` gestisce il numero di dati in output.

Uso di nargin e nargout

Se nella funzione `[x1,x2] = roots_2deg(a,b,c)` (vista nelle prime slide) si utilizzassero i comandi

```
fprintf('Numero di parametri in ingresso : %d\n',nargin);
fprintf('Numero di parametri in uscita   : %d\n',nargout);
```

si otterrebbe

```
Numero di parametri in ingresso : 3
Numero di parametri in uscita   : 2
```

`varargin` e `varargout` sono variabili di tipo `cell`

Gestione I/O: esempio

```
function [a, varargout] = prova_args(x, y, varargin)
%prova_args - Numero variabile di argomenti di input e output
% Mostra l?utilizzo di nargin, nargout, varargin e varargout per
% creare M-function con un numero variabile di argomenti di input
% e/o di output

if nargin < 2
  error('Attenzione! Assegnare almeno 2 variabili in input.');
end

% Se in input viene dato anche il terzo parametro (opzionale)
% allora viene utilizzato, altrimenti viene settato il
% valore di default che in questo caso vale 1
if nargin > 2
    z = varargin{1};
else
    z = 1;
end

c = x + y;
b = x - y;
a = c^z;
```

...continua nella prossima slide...

Gestione I/O: esempio

```
switch nargout
    case 0
        fprintf('Funzione chiamata senza variabili in output\n');
    case 1
        % niente da fare (primo output assegnato di default)
    case 2
        varargout{1} = c;
    case 3
        varargout{1} = c;
        varargout{2} = b;
    otherwise
        % nel caso in cui il numero di parametri
        % richiesti non rientri nella casistica considerata
        error('Attenzione! Numero max di parametri di output: 3');
end % end switch
end % end function
```

Il primo `if-then-else` controlla il numero di parametri in entrata, che in questo esempio deve essere almeno 2. Il secondo controlla se èpresente il parametro di input opzionale.
Il costrutto `switch` gestisce l'output in base a quanti dati in uscita sono stati richiesti nella chiamata alla funzione.

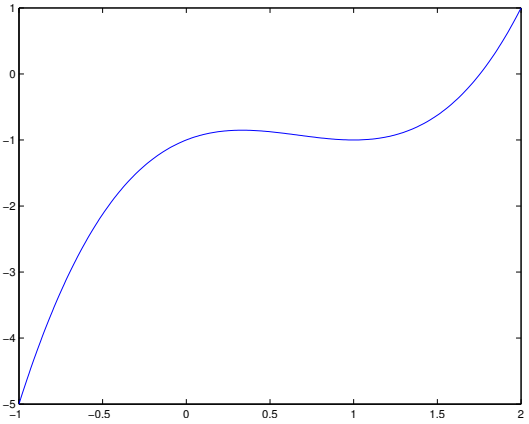
Grafica 2D

Si supponga di dover disegnare il grafico del polinomio

$$p(x) = x^3 - 2x^2 + x - 1$$

nell'intervallo $[-1, 2]$. In Matlab è presente l'istruzione `plot` che consente di creare grafici bidimensionali. Ricordando la sintassi vettoriale di Matlab, si ha:

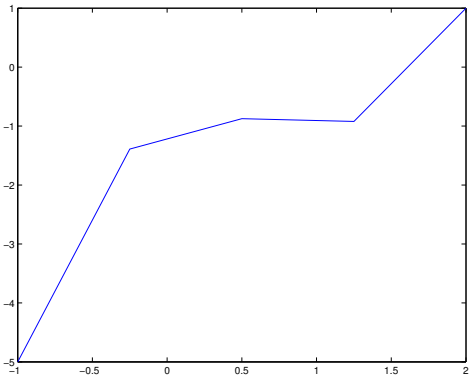
```
>> x = linspace(-1,2,50);
>> y = x.^3 - 2*x.^2 + x - 1;
>> ph = plot(x,y);
```



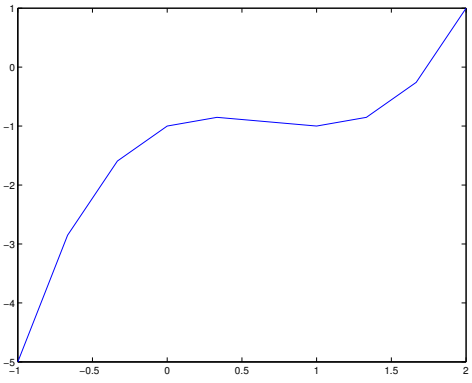
Matlab collega con una linea i punti `[x(i),y(i)]` e `[x(i+1),y(i+1)]`: più punti di discretizzazione vengono utilizzati, migliore qualità visiva avrà il grafico.

`ph` è un *graphics handle*: permette di accedere in lettura e scrittura a tutte le proprietà grafiche degli oggetti grafici creati dall'istruzione (la curva).
Ogni singolo oggetto grafico ha un proprio graphics handle. Ogni oggetto grafico fa parte di una gerarchia, al cui apice c'è l'oggetto grafico `Root`.
Ogni istruzione di creazione di oggetti grafici restituisce almeno un graphics handle, ma può restituirne più di uno.

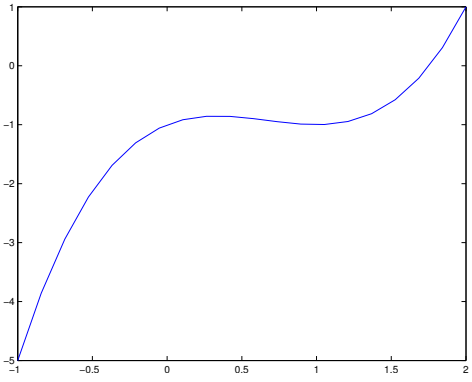
Grafica 2D



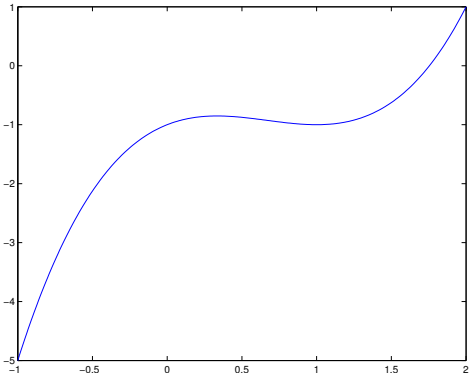
$n = 5$



$n = 10$



$n = 20$

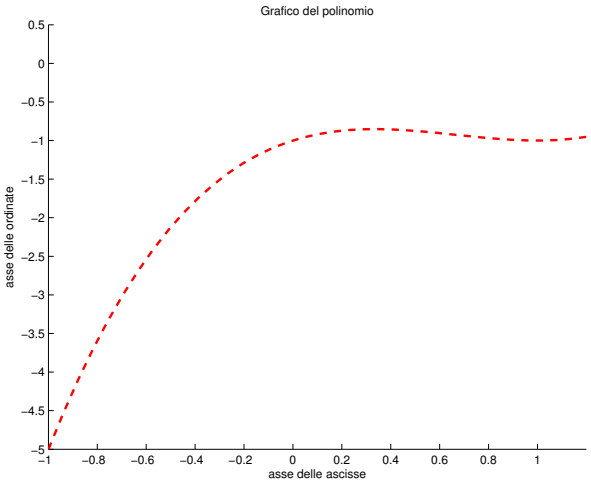


$n = 50$

Grafica 2D

È possibile personalizzare in vari modi il grafico:

```
x = linspace(-1,2,50);
y = x.^3 - 2*x.^2 + x - 1;
plot(x,y,'r--','Linewidth',2);
box off
axis([-1 1.2 -5 0.5]);
xlabel('asse delle ascisse');
ylabel('asse delle ordinate');
title('Grafico del polinomio');
```



Nel dettaglio:

- `axis([x1 x2 y1 y2])` consente di limitare la visualizzazione tra `x1` e `x2` per l'asse delle ascisse e fra `y1` e `y2` per l'asse delle ordinate;
- l'opzione `r--` consente di usare il colore rosso e di usare una linea tratteggiata;
- i comandi `xlabel` e `ylabel` consentono di mettere etichette all'asse delle x e delle y, rispettivamente;
- `title` consente di inserire una stringa come titolo del grafico.

Per personalizzazioni fini si usano i graphics handle.

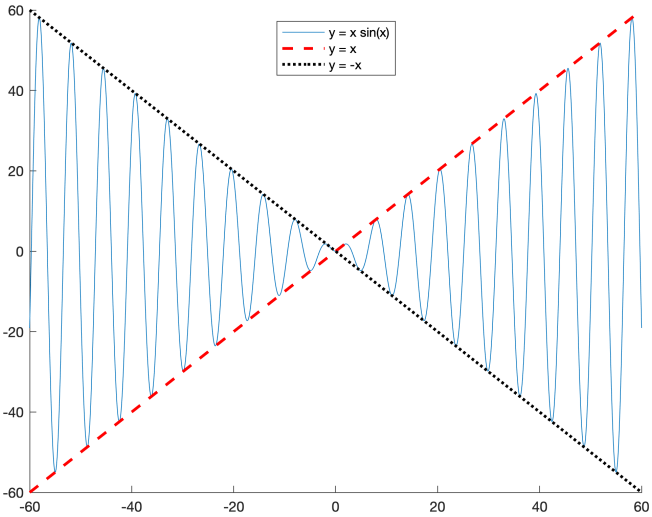
Grafica 2D

Si supponga di dover disegnare i grafici delle seguenti funzioni

$$f(x) = x\sin(x)$$
$$g(x) = x$$
$$h(x) = -x$$

nell'intervallo $[-20\pi, 20\pi]$.

```
x = -20*pi:0.1:20*pi;
f = x.*sin(x);
g = x;
h = -x;
plot(x,f)
hold on
plot(x,g,'r--','Linewidth',2)
plot(x,h,'k:','Linewidth',2)
axis([-60 60 -60 60])
box off
legend('y = x sin(x)',...
       'y = x','y = -x',...
       'Location','North');
```



Grafica 2D

I tre puntini `...` consentono di scrivere un'istruzione Matlab su più righe, ma il programma legge l'intera istruzione come se fosse su di una riga unica.

Vediamo nel dettaglio i vari comandi.

- `Linewidth` consente di specificare lo spessore della linea. Di default è 1.
- dopo aver dichiarato la variabile indipendente (`x`) e quella indipendente (`f`), si possono specificare i colori, lo stile e i marker dei punti.
  - le lettere `r,b,k,c,y,m` identificano i colori della linea: red, blue, black, cyan, yellow, magenta;
  - le scritture `-`, `.-`, `:`, `--` identificano lo stile della linea: continuo, punto-linea, punteggiata, tratteggiata.
  - le scritture `o`, `+`,`s`,... identificano lo stile dei markers: tondo, +, quadrato, ecc. ...

  è possibile inserire tutte queste opzioni in un'unica chiamata: per esempio `'hm.-'` disegna una linea di color magenta, con markers esagonali e una linea punteggiata e tratteggiata.
- `axis([x1 x2 y1 y2])` consente di limitare la visualizzazione tra `x1` e `x2` per l'asse delle ascisse e fra `y1` e `y2` per l'asse delle ordinate;
- `legend` consente di disegnare la legenda del grafico, mettendo fra apici le descrizione delle linee del grafico *nell'ordine in cui sono state disegnate*.
- `box off` consente di eliminare la "scatola" che appare attorno al grafico

Grafica 2D

Supponiamo di avere i seguenti dati:

x	y
147	1600
150	1300
220	1800
282	1900
312	2400
374	2600
412	2300
423	2600
457	2700
583	2800
602	2900
623	3100

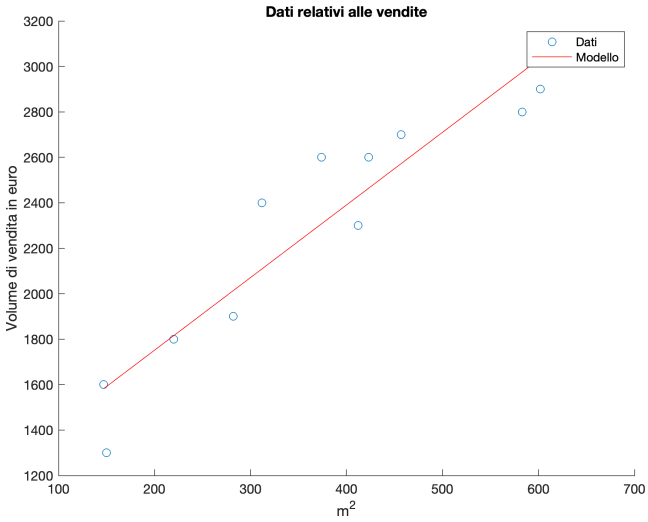
Si vogliono disegnare nello stesso grafico i punti (x_i, y_i) e la retta di regressione

$$y = 3.2x + 1112.5$$

introducendo nel grafico un titolo ("Dati relativi alle vendite"), un'etichetta per l'asse delle ascisse ("m^2"), un'etichetta per l'asse delle ordinate ("Volume di vendita in euro") e una legenda (composta da due voci: "Dati", "Modello"). Si vogliono inoltre usare per la retta e per i punti stili diversi da quelli standard.

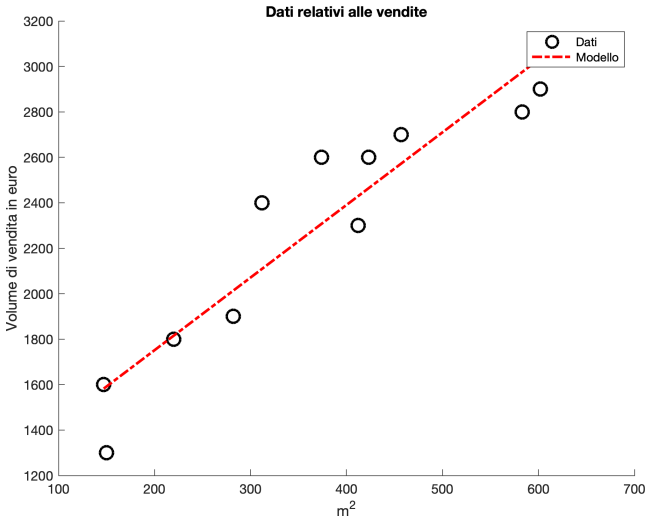
Grafica 2D

```
% Dati
x = [147,150,220,282,312,374,412,423,457,583,602,623]';
y = [1600,1300,1800,1900,2400,2600 2300,2600,2700,2800,2900,3100]';

% Coeff. retta di regressione
a = polyfit(x,y,1);
% Retta di regressione
r = a(1)*x + a(2);
% Plot dei punti
plot(x,y,'o');
% Sovrascrive grafici
hold on
% Plot della retta
plot(x,r,'r');
% Disattiva sovrascr. grafici
hold off
box off
xlabel('m^2');
ylabel('Volume di vendita in euro');
title('Dati relativi alle vendite');
legend({'Dati','Modello'});
```



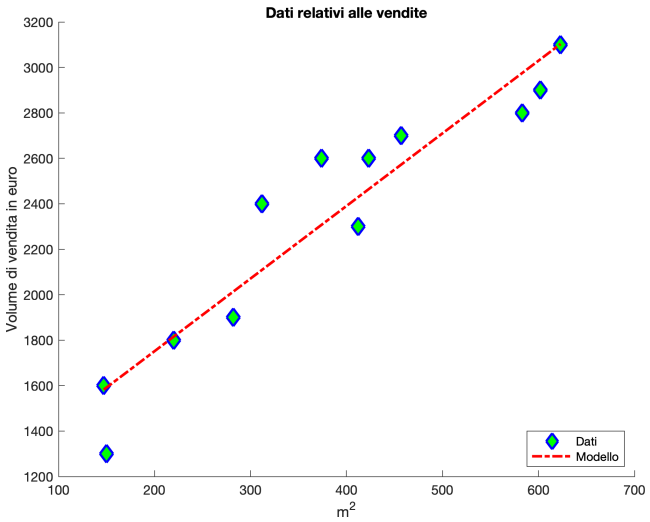
Grafica 2D

```
% Dati
x = [147,150,220,282,312,374,412,423,457,583,602,623]';
y = [1600,1300,1800,1900,2400,2600 2300,2600,2700,2800,2900,3100]';

% Coeff. retta di regressione
a = polyfit(x,y,1);
% Retta di regressione
r = a(1)*x + a(2);
% Plot dei punti
plot(x,y,'o',...
     'MarkerSize',10,...
     'MarkerEdgeColor','k',...
     'Linewidth',2)
hold on  % sovrascrive grafici
plot(x,r,'r-.','Linewidth',2);
hold off % disattiva sovrascr.
box off
xlabel('m^2');
ylabel('Volume di vendita in euro');
title('Dati relativi alle vendite');
legend({'Dati','Modello'});
```



Grafica 2D

```
% Dati
x = [147,150,220,282,312,374,412,423,457,583,602,623]';
y = [1600,1300,1800,1900,2400,2600 2300,2600,2700,2800,2900,3100]';

% Coeff. retta di regressione
a = polyfit(x,y,1);
% Retta di regressione
r = a(1)*x + a(2);
% Plot dei punti
plot(x,y,'d',...
     'MarkerSize',10,...
     'MarkerEdgeColor','b',...
     'MarkerFaceColor','g',...
     'Linewidth',2)
hold on  % sovrascrive grafici
plot(x,r,'r-.','Linewidth',2);
hold off % disattiva sovrascr.
box off
xlabel('m^2');
ylabel('Volume di vendita in euro');
title('Dati relativi alle vendite');
legend({'Dati','Modello'},'Location','SouthEast');
```



Grafica 2D

Una breve tabella riassuntiva di alcune opzioni per il comando `plot`.

Colore	Significato	Simbolo	Significato	Linea	Significato
w	bianco	.	punto	-	linea continua
y	giallo	o	circoletto	:	linea punteggiata
r	rosso	x	croce a "x"	-.	punto e linea
g	verde	+	croce a "+"	--	tratteggio
b	blu	*	stella		
k	nero	p	fiore		
m	magenta	s	quadrato		
c	ciano	d	rombo		
		h	esagono		
		v	triang. rovesciato		
		^	triangolo dritto		
		<	triangolo sinistro		
		>	triangolo destro		

Grafica 2D: subplot

Il comando `subplot(m,n,i)` consente di creare una griglia con `m` righe e `n` colonne. Alla `i`-esima posizione verrà posizionato un grafico con le caratteristiche elencate.

Un polinomio e le sue derivate

Si supponga di dover disegnare nell'intervallo $[-5, 5]$ il seguente polinomio

$$p(x) = -x^5 - 0.2x^4 + 0.6x^3 - 2x^2 + x + 0.5$$

e le sue derivate. Le seguenti istruzioni in Matlab consentono di valutarlo nell'intervallo desiderato:

```
x = -5:0.01:5;
p = [-1 -0.2 0.6 -2 1 0.5];
y = polyval(p,x);
```

I seguenti comandi invece consentono di calcolarne le derivate in maniera compatta:

```
% derivata prima
dp = polyder(p);
dy = polyval(dp,x);
% derivata seconda
d2p = polyder(dp);
d2y = polyval(d2p,x);
```

```
% derivata terza
d3p = polyder(d2p);
d3y = polyval(d3p,x);
% derivata quarta
d4p = polyder(d3p);
d4y = polyval(d4p,x);
```

```
% derivata quinta
d5p = polyder(d4p);
d5y = polyval(d5p,x);
% derivata sesta
d6p = polyder(d5p);
d6y = polyval(d6p,x);
```

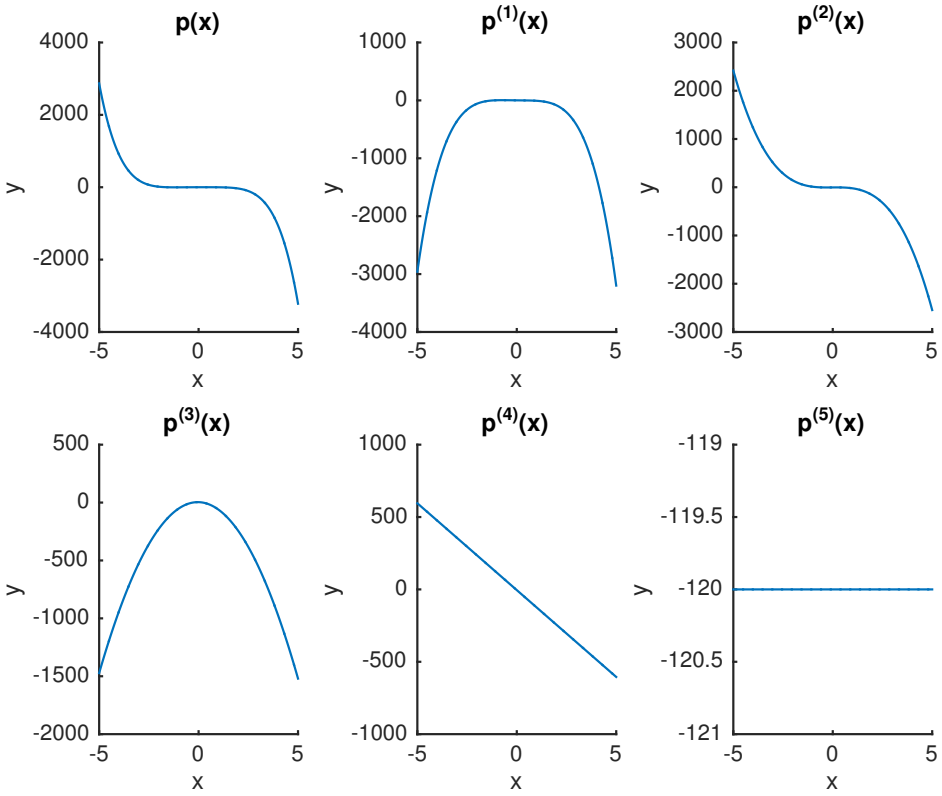

Grafica 2D: subplot

<continua> Un polinomio e le sue derivate

```
subplot(2,3,1)
plot(x,y);
xlabel('x')
ylabel('y')
box off
title('p(x)')

subplot(2,3,2);
plot(x,dy);
xlabel('x')
ylabel('y')
box off
title('p^{(1)}(x)')

subplot(2,3,3);
plot(x,d2y);
xlabel('x')
ylabel('y')
box off
title('p^{(2)}(x)')
```

```
subplot(2,3,4);
plot(x,d3y);
xlabel('x')
ylabel('y')
box off
title('p^{(3)}(x)')

subplot(2,3,5);
plot(x,d4y);
xlabel('x')
ylabel('y')
box off
title('p^{(4)}(x)')

subplot(2,3,6);
plot(x,d5y);
xlabel('x')
ylabel('y')
box off
title('p^{(5)}(x)')
```

Grafica 2D: subplot



Grafica 2D: figure

Il comando `figure` consente di creare una nuova finestra grafica.

```
x = -2:0.01:1;
y = x.^2.*sin(x);
plot(x,y)   % disegna il grafico di y in una finestra
figure;     % crea una finestra vuota
y2 = sin(x);
plot(x,y2) % disegna il grafico di y2 nella finestra appena creata
```

Se è necessario agire su una finestra già creata si usa il comando `figure(N)`.

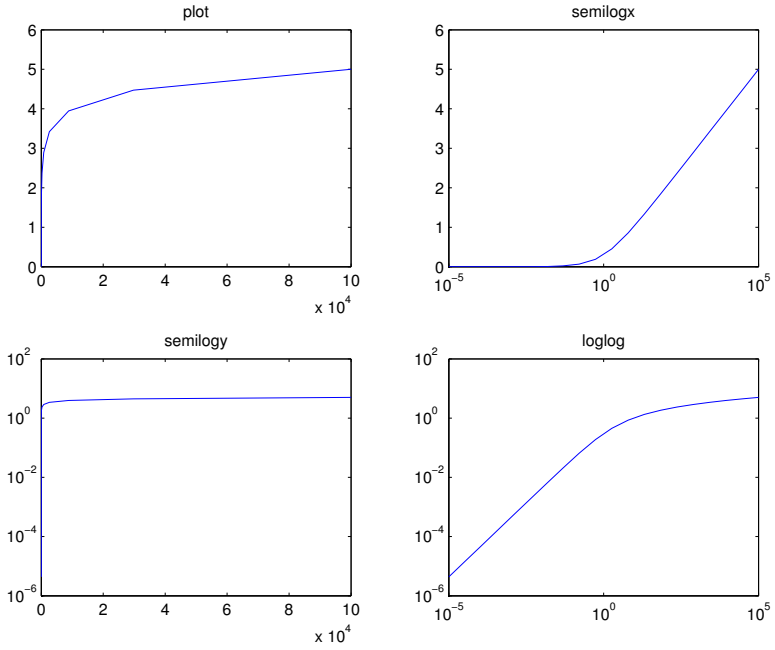
```
x = -2:0.01:1;
y = x.^2.*sin(x);
figure(1)
plot(x,y)
figure(2);
y2 = sin(x);
plot(x,y2)
figure(1)
title('Primo test')
figure(2)
xlabel('x')
```

Il comando `figure(N)` consente di creare una nuova figura "etichettata" `N` e, una volta creata, si può lavorare su tale finestra richiamandola tramite lo stesso comando.

Grafica 2D: semilog

I comandi `semilogx`, `semilogy`, `loglog` consentono di disegnare i grafici utilizzando una scala logaritmica sull'asse delle ascisse, delle ordinate o su entrambi, rispettivamente.

```
x = logspace(-5,5,20);
y = log10(x+1);
subplot(2,2,1)
plot(x,y)
title('plot')

subplot(2,2,2)
semilogx(x,y)
title('semilogx')

subplot(2,2,3)
semilogy(x,y)
title('semilogy')

subplot(2,2,4)
loglog(x,y)
title('loglog')
```



Grafica 2D: altri comandi utili

`text`

`text(x,y,str)`	inserisce la stringa specificata nel punto di coordinate `(x,y)`; una variante di questa istruzione è `gtext(str)`, che inserisce la stringa specificata nel punto in cui si clicca con il mouse. In `legend`, `xlabel`, `ylabel`, `text`, `title` la stringa può contenere simboli specificati mediante comandi TEX.
`fill(x,y,colore)`	crea un poligono di vertici aventi coordinate $x(i), y(i)$ e lo riempie con il `colore` specificato; l'istruzione chiude il poligono

`print -d`*driver* [`-r`*ndpi*] `NOMEFILE.`*ext*
Salva il contenuto della finestra grafica corrente nel file `NOMEFILE.`*ext*, nel formato grafico specificato da *driver*, alla risoluzione opzionalmente specificata di *ndpi* punti per pollice (dpi, *dot-per-inch*). Quest'ultima opzione non è disponibile per alcuni formati grafici. Anche `print('-d`*driver*`', '-r`*ndpi*`', 'NOMEFILE.`*ext*`')`.

`print -dpng -r300 plot1.png`	salva nel formato vettoriale Portable Network Graphics alla risoluzione di 300 dpi
`print -depsc2 plot1.eps`	salva nel formato Encapsulated PostScript Color Level 2
`print -dpdf plot1.pdf`	salva nel formato Portable Document Format

Grafica 2D: altri comandi utili

`axis`

`v = axis`	nel vettore v è riportata la scala usata
`axis auto`	ritorna alla modalità di scalatura automatica
`axis equal`	fissa la stessa unità di misura sui due assi
`axis square`	rende il sistema di riferimento quadrato, usando unità di misura diverse sui due assi
`axis on; axis off`	abilita e disabilita la visualizzazione degli assi, con le etichette per gli assi
`axis ij; axis xy`	pone l'origine degli assi in alto a destra, con valori crescenti dell'asse y verso il basso; `axis xy` è la modalità normale

Di ciascuno di questi comandi sono accettate anche la forma con parametro di tipo stringa, come ad es. `axis 'auto'`, e la forma funzionale, come ad es. `axis('auto')`.

`grid on|grid off|grid`

Abilita o disabilita la visualizzazione di una griglia nel piano cartesiano; `grid` da solo permette di vedere se la specifica è abilitata o meno.

Grafica 2D: circonferenza

L'equazione cartesiana della circonferenza di raggio r è

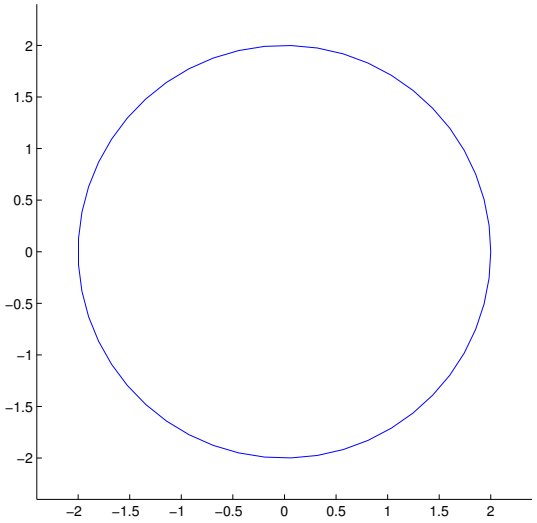
$$x^2 + y^2 = r^2$$

La sua forma parametrica è quindi

$$\begin{cases} x = r\cos(\theta) \\ y = r\sin(\theta) \end{cases}$$

con $\theta \in [0, 2\pi)$.

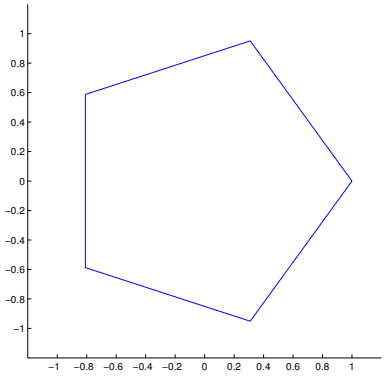
```
t = linspace(0,2*pi,100);
r = 2;
x = r*cos(t);
y = r*sin(t);
plot(x,y)
axis square
axis(1.2*[-r r -r r])
box off
```



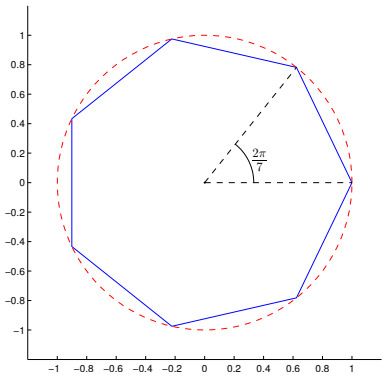
Grafica 2D: poligono regolare

Per disegnare un poligono di n lati, si prendono $n + 1$ punti equidistanziati sulla circonferenza unitaria.

```
t = linspace(0,2*pi,6);
r = 1;
x = r*cos(t);
y = r*sin(t);
plot(x,y)
axis square
axis(1.2*[-r r -r r])
box off
```



```
t = linspace(0,2*pi,8);
r = 1;
x = r*cos(t);
y = r*sin(t);
plot(x,y)
axis square
axis(1.2*[-r r -r r])
box off
```

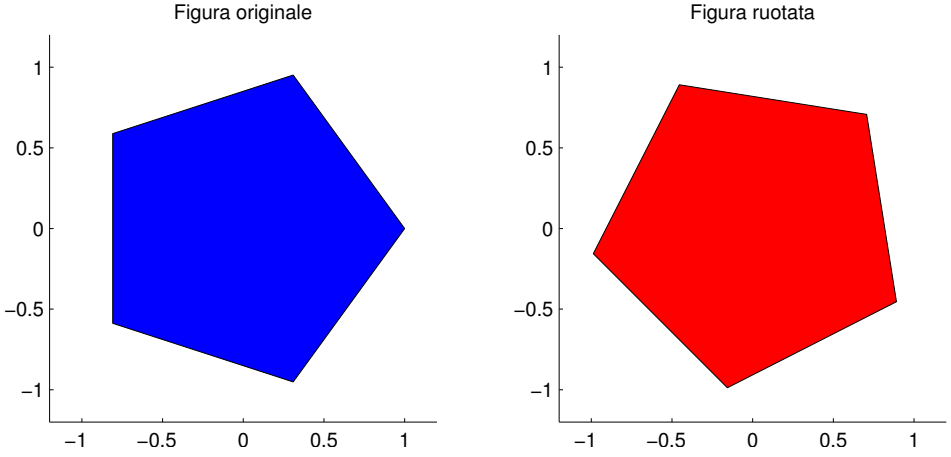


Grafica 2D: matrici di rotazione

Dal corso di Matematica Discreta. Una matrice del tipo

$$A = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix}$$

è una matrice di rotazione: considerando un vettore x, il prodotto $y = Ax$ corrisponde alla rotazione di x attorno all'origine di un angolo φ in senso antiorario.



Grafica 2D: matrici di rotazione

```
% numero di lati
n = 5;

% Angolo di rotazione
phi = pi/4;
% Matrice di rotazione
A = [cos(phi) -sin(phi)
     sin(phi)  cos(phi)];

t = linspace(0,2*pi,n+1);

% Matrice contenente le
% coordinate dei punti:
% nella prima riga si salvano
% le ascisse dei punti,
% sulla seconda le ordinate.
% L'i-esima colonna
% contiene le coordinate
% dell'i-esimo punto
P = zeros(2,n+1);
P(1,:) = cos(t);
P(2,:) = sin(t);
```

```
% Plot poligono originale
subplot(1,2,1)
fill(P(1,:),P(2,:),'b');
hold on;
plot(P(1,1),P(2,1),'pk');
axis square
axis([-1.2 1.2 -1.2 1.2])
box off
title('Figura originale')

% Rotazione
P_rot = A*P;

% Plot poligono originale
subplot(1,2,2)
fill(P_rot(1,:),P_rot(2,:),'r');
hold on;
plot(P_rot(1,1),P_rot(2,1),'pk');
axis square
axis([-1.2 1.2 -1.2 1.2])
box off
title('Figura ruotata')
```