Calcolo Numerico

(A.A. 2013-2014)

Esercitazioni

Lezione n. 16

Approssimazione

23-05-2014

Approssimazione di dati e funzioni

Problema

Data la **tabella** $\{x_i, y_i\}$, $i = 0, \ldots, n$, si vuole trovare una **funzione analitica** φ_M che **approssimi** i dati.

La **tabella** $\{x_i, y_i\}$ può essere il risultato di **misure sperimentali** oppure può rappresentare i valori di una funzione la cui **espressione analitica** è nota ma **complicata** da calcolare direttamente.

Per poter costruire una **funzione approssimante** bisogna stabilire

- in quale **classe** di funzioni si vuole operare
- il **metodo di approssimazione**

Interpolazione polinomiale

Tabella: $\{x_i, y_i\} \qquad i = 0, \ldots, n$

Intervallo di interpolazione: $[a, b] = [x_0, x_n]$

Funzione approssimante: $p_n(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n$

Metodo di approssimazione: $p_n(x_i) = y_i \quad (i = 0, 1, \ldots, n)$ $\rightarrow$ **Interpolazione**

Risolvere il **problema dell'interpolazione** vuol dire individuare il polinomio p_n, cioè i **coefficienti reali** a_k, che soddisfano le **condizioni di interpolazione**. Questo equivale a risolvere il **sistema lineare**

$$\begin{array}{l} p(x_0) = \\ p(x_1) = \\ \\ \\ p(x_n) = \end{array} \left\{ \begin{array}{ccccccccc} a_0 & + & a_1 x_0 & + & a_2 x_0^2 & + & \cdots & + & a_n x_0^n & = & y_0 \\ a_0 & + & a_1 x_1 & + & a_2 x_1^2 & + & \cdots & + & a_n x_1^n & = & y_1 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ a_0 & + & a_1 x_n & + & a_2 x_n^2 & + & \cdots & + & a_n x_n^n & = & y_n \end{array} \right. \rightarrow \quad VA = Y$$

Unicità del polinomio interpolatore

$$\boxed{VA = Y} \text{ con } V = \underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}}_{\text{Matrice di Vandermonde}} \quad A = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} \quad Y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

La **matrice di Vandermonde** di $n+1$ nodi **distinti** $\{x_i\}$, $i = 0, \ldots, n+1$, è **regolare** poiché

$$\det V = \prod_{j>i} (x_i - x_j) \neq 0$$

$\Rightarrow$ esiste un'**unica** soluzione A del sistema.

$$\Downarrow$$

Esiste **uno e uno solo** polinomio p_n di grado n che verifica le **condizioni di interpolazione**

$$p_n(x_i) = y_i \qquad i = 0, \ldots, n$$

Approssimazione ai minimi quadrati

Problema.
Data la **tabella** $\{x_i, y_i\}$, $i = 0, 1, \ldots, n$, si vuole trovare una **funzione analitica** φ_M che **approssimi** i dati.

In questo caso la **tabella** è il risultato di **misure sperimentali** ciascuna delle quali è affetta da un **errore di misura** ε_i.

Metodo di approssimazione: si sceglie la funzione approssimante φ_M in modo da **minimizzare**

$$\sum_{i=0}^{n} [\varphi_M(x_i) - y_i]^2 \qquad \textbf{Scarto quadratico}$$

oppure, introducendo i **pesi** $w_i > 0$, $\forall i$,

$$\sum_{i=0}^{n} w_i \, [\varphi_M(x_i) - y_i]^2 \qquad \textbf{Scarto quadratico pesato}$$

Polinomio algebrico ai minimi quadrati

Tabella: $\{x_i, y_i\} \qquad i = 0, 1, ..., n$

Funzione approssimante:

$$P_M(x) = a_0 + a_1 x + \cdots + a_{M-1} x^{M-1} + a_M x^M \qquad \boxed{M << n}$$

Metodo di approssimazione: si minimizza lo **scarto quadratico**

$$\sigma^2(a_0, a_1, ..., a_M) = \sum_{i=0}^{n} [\underbrace{a_0 + a_1 x_i + \cdots + a_{M-1} x_i^{M-1} + a_M x_i^M}_{P_M(x_i)} - y_i]^2$$

Funzioni di base:

$$\psi_0(x) = 1, \ \psi_1(x) = x, \ \ldots, \ \psi_k(x) = x^k, \ \cdots, \ \psi_M(x) = x^M$$

Esercizio

Scrivere la funzione matlab **vandermonde.m** che riceva in input un vettore $\mathbf{X}$ e un intero N e costruisca la matrice di Vandermonde $\mathbf{V}$ delle componenti del vettore $\mathbf{X}$ (nodi). Il numero di colonne di $\mathbf{V}$ deve essere $N + 1$.

Si generi un vettore $\mathbf{X}$ di 20 elementi equispaziati nell'intervallo $[0,\ 1]$ e se ne calcoli la matrice di Vandermonde $\mathbf{V}$ associata di dimensione 20×5 usando la funzione **vandermonde.m**.

Confrontare i risultati usando la funzione predefinita di Matlab **vander.m**

```
function [V] = vandermonde(X,N)
% function [V] = vandermonde(X)
% calcola la matrice di Vandermonde dei nodi X(i).
%
% INPUT
% X = vettore di nodi
% N = numero di colonne +1 della matrice di Vandermonde
%
% OUTPUT
% V = matrice la cui k-esima colonna e' data da X.^k

dimX = size(X);
if size(dimX)>1
    error('La variabile di input deve essere un vettore!')
end

for k = 0:N
    V(:,k+1) = X.^k;
end
```

Dal Command Window

```
>> x = linspace(0,1,20);
>> V = vandermonde(x,4);
>> V

V =

    1.0000         0         0         0         0
    1.0000    0.0526    0.0028    0.0001    0.0000
    1.0000    0.1053    0.0111    0.0012    0.0001
    1.0000    0.1579    0.0249    0.0039    0.0006
    1.0000    0.2105    0.0443    0.0093    0.0020
    1.0000    0.2632    0.0693    0.0182    0.0048
    1.0000    0.3158    0.0997    0.0315    0.0099
    1.0000    0.3684    0.1357    0.0500    0.0184
    1.0000    0.4211    0.1773    0.0746    0.0314
    1.0000    0.4737    0.2244    0.1063    0.0503
    1.0000    0.5263    0.2770    0.1458    0.0767
    1.0000    0.5789    0.3352    0.1941    0.1123
    1.0000    0.6316    0.3989    0.2519    0.1591
    1.0000    0.6842    0.4681    0.3203    0.2192
    1.0000    0.7368    0.5429    0.4001    0.2948
    1.0000    0.7895    0.6233    0.4921    0.3885
    1.0000    0.8421    0.7091    0.5972    0.5029
    1.0000    0.8947    0.8006    0.7163    0.6409
    1.0000    0.9474    0.8975    0.8503    0.8055
    1.0000    1.0000    1.0000    1.0000    1.0000
```

Funzione polyval

Matlab rappresenta i polinomi mediante vettori che contengono i coefficienti del polinomio stesso

y = polyval (a,x)

valuta il polinomio i cui coefficienti sono contenuti nel vettore $\mathbf{a}$ in corrispondenza degli elementi del vettore $\mathbf{x}$ Il grado massimo n del polinomio è pari alla lunghezza del vettore $\mathbf{a}$ diminuita di 1. Il primo elemento di $\mathbf{a}$ è il **coefficiente moltiplicativo del monomio gi grado massimo**:

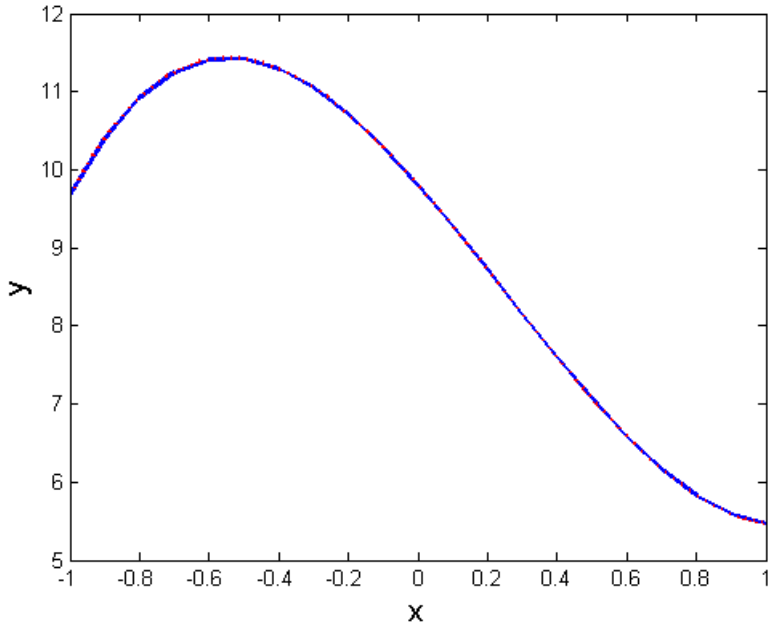
$$y_i = a_1\, x_i^n + a_2\, x_i^{n-1} + \cdot + a_n\, x_i + a_{n+1}$$

Esempio

```
>> a=[1 2 8];
>> x = 5;
>> polyval(a,x)
ans =
    43
>> x^2+2*x+8
ans =
    43
>> x = [5 4]; % x puo' essere un vettore
>> polyval(a, x)
ans =
    43    32
>>  x.^2+2*x+8 % in questo caso scrivo x.^2
ans =
    43    32
```

La funzione polyval può essere utile anche per **disegnare un polinomio**

```
>> a = [3 -2.23 -5.1 9.8];
>> x = -1:.1:1;
>> y = polyval(a,x);
>> plot(x,y)
>> hold on, fplot(@(x)[3*x^3-2.23*x^2-5.1*x+9.8],[-1 1],'r:')
>> xlabel('x')
>> ylabel('y')
```



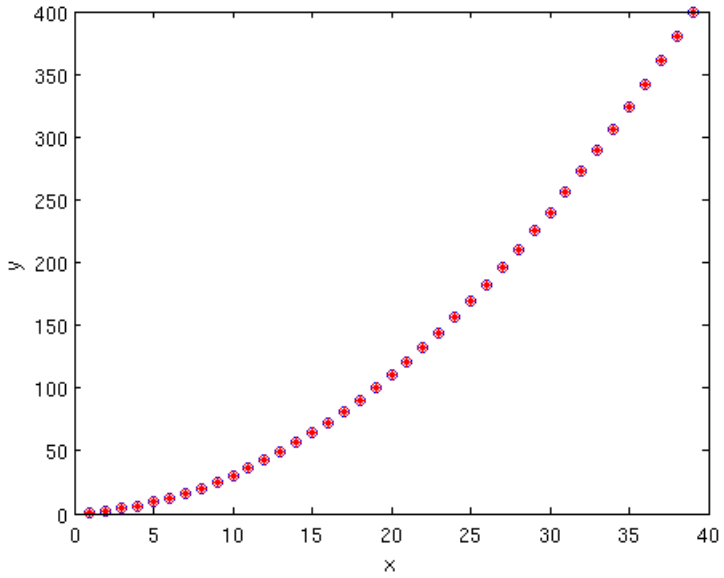
,

Funzione polyfit

a = polyfit (x,y,n)

calcola i coefficienti **a** del polinomio di grado **n** che approssima i valori in **y** corrisponenti ai nodi in **x** usando la tecnica dei minimi quadrati. I vettori delle ascisse e delle ordinate devono avere la stessa lunghezza. Il grado del polinomio $\mathbf{n <= length(x) - 1}$

```
>> x = [1:0.5:20]; y = x.^2;
>> a = polyfit(x,y,2);
>> yn = polyval(a,x);
>> figure, plot(y,'o'), hold on, plot(yn,'r*')
>> xlabel('x'), ylabel('y')
```

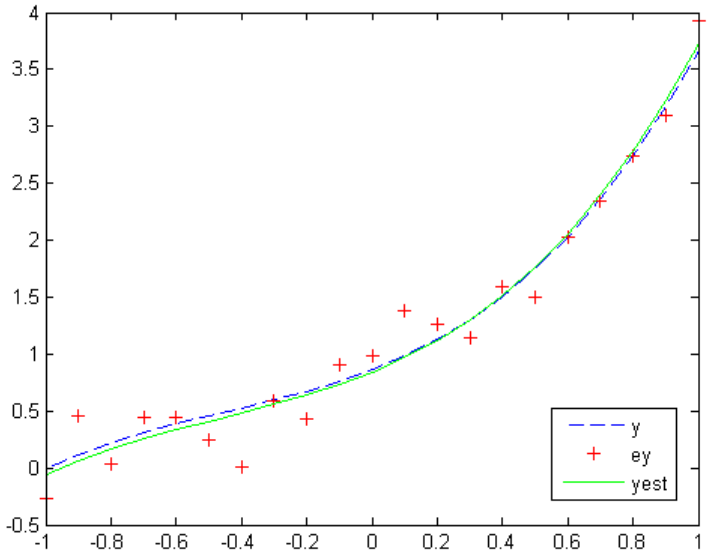

400
350
300
250
200
150
100
50
0
y
0
5
10
15
20
25
30
35
40
x
,

Esercizio

Scrivere uno script matlab che generi un vettore y contenente i valori di un polinomio di terzo grado in corrispondenza di nodi equidistanti di passo 0.1 nell'intervallo $[-1, 1]$. Si generi il vettore $noise$, tale che $\|noise\|_2 = 1$, della stessa dimensione di y usando la funzione **randn**. Sia $ey = y + noise$ (simulano i dati ottenuti in un esperimento numerico). Si stimi il polinomio di terzo grado che approssima i dati ey nel senso dei minimi quadrati. Sia $yest$ il vettore stimato e lo si confronti con il vettore originale y

```
>> a = [0.74 0.97 1.1 0.86];
>> x = -1:.1:1;
>> y = polyval(a,x);
>> noise = randn(1,length(y));
>> noise = noise/norm(noise);
>> ey = y + noise;
>> aest = polyfit(x,ey,3);
>> disp([a;aest])
    0.7400    0.9700    1.1000    0.8600
    0.7166    0.9985    1.1751    0.8379
>> yest = polyval(aest,x);
```

```
>> figure
>> plot(x,y,'b--')
>> hold on
>> plot(x,ey,'r+')
>> plot(x,yest,'g')
>> legend('y','ey','yest')
>> norm((y-yest))^2
ans =
    0.0324
```



,

Esercizio

La tabella seguente riporta le misure della densità relativa ρ dell'aria a diverse altezze h.

h (km)	0	1.525	3.050	4.575	6.100	7.625	9.150
ρ	1	0.8617	0.7385	0.6292	0.5328	0.4481	0.3741

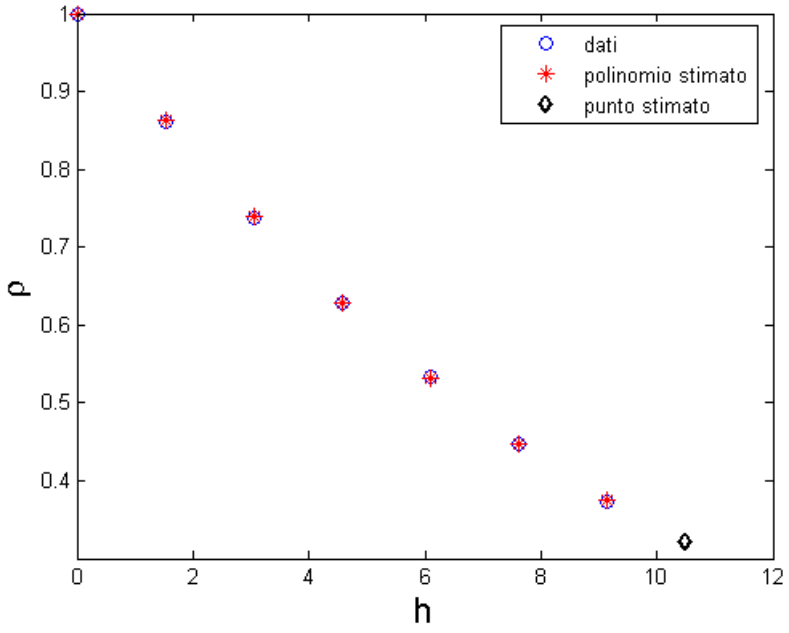
Si approssimi ρ con un polinomio di secondo grado e si stimi il valore di ρ in corrispondenza di $h = 10.5$ km.

Soluzione

```
h = [0 1.525  3.050  4.575 6.100 7.625 9.150];
rho = [1 0.8617  0.7385  0.6292 0.5328 0.4481 0.3741];

a = polyfit(h,rho,2);
disp(a)

rho_pol = polyval(a,h);

rho_punto = polyval(a,10.5);

figure, plot(h,rho,'o')
hold on, plot(h,rho_pol,'r*')
hold on, plot(10.5,rho_punto,'kd')
xlabel('h')
ylabel('\rho')
legend('dati','polinomio stimato','punto stimato')
```

dal command window

```
>> esercizio_rhoaria
0.0028    -0.0934     0.9989
```



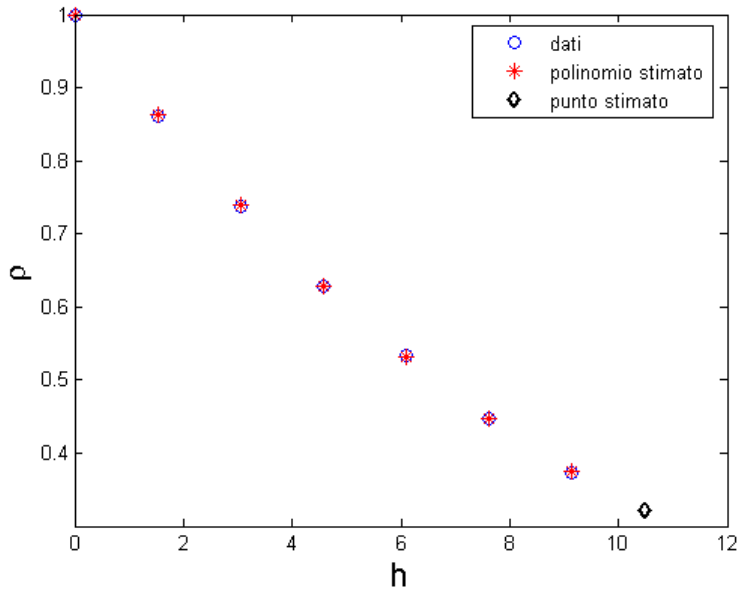
,

In modo analogo

```
V = vandermonde(h,2);
H = V'*V;
B = V'*rho';

a = H\B;
disp(a)

a = fliplr(a')
rho_pol = polyval(a,h);

rho_punto = polyval(a,10.5);

figure, plot(h,rho,'o')
hold on, plot(h,rho_pol,'r*')
hold on, plot(10.5,rho_punto,'kd')
xlabel('h')
ylabel('\rho')
legend('dati','polinomio stimato','punto stimato')
```

dal command window

```
>> esercizio_rhoaria
    0.9989
   -0.0934
    0.0028
a =
    0.0028   -0.0934    0.9989
```



,

Esercizio

Scrivere una funzione matlab che riceva in input il vettore dei nodi x e il vettore delle misure y e restituisca come output il vettore a dei coefficienti del polinomio che meglio approssima i dati nel senso dei minimi quadrati e l'errore dell'approssimazione err. La funzione deve prevedere come eventuale terzo input un vettore contenente i punti in cui valutare il polinomio stimato. Sia $ynew$ il vettore contenente i valori calcolati. $ynew$ è un'ulteriore variabile di output della funzione.

```
function [coeff_pol,varargout] = find_bestpol(xdata,ydata,varargin)
% [coeff_pol,varargout] = find_bestpol(xdata,ydata,varargin)
% calcola i coefficienti del polinomio che meglio approssima i dati ydata
% nei punti xdata. Come terzo vettore di input  possibile passare i punti
% in cui stimare la funzione approssimante (terza variabile di output).
%
% INPUT
% xdata = vettore dei nodi
% ydata = vettore dei valori della funzione nei nodi
%
%
% OUTPUT
% coeff_pol = vettore dei coefficienti del polinomio di migliore
%             approssimazione
%
```

```
% controllo degli input e degli putput
if nargin < 2
    error('dati di input non sufficienti!')
elseif nargin ==2
    xnew = [];
elseif nargin == 3
    xnew = varargin{1};
elseif nargin>3
    error('troppe variabili di input!!!')
end

if nargout > 2
    error('troppe variabili di output!!!')
elseif (nargin == 3) & (nargout~=2)
    error('il numero delle variabili di output deve essere 2!!!')
end
```

```
xdata = xdata(:);
ydata = ydata(:);

if (length(xdata) ~= length(ydata))
    error('il numero di dati deve essere  uguale al numero di nodi!!!')
end

% calcolo dei polinomi di approssimazione ai minimi quadrati di grado n
% con 1<=n<=nmax
if length(xdata)>10
    nmax = 10;
else
    nmax = max(length(xdata)-2,1);
end
for n = 1:nmax
    coeff_poli{n} = polyfit(xdata,ydata,n);
    ystime = polyval(coeff_poli{n},xdata);
    errore(n) = sum((ystime-ydata).^2);
end
```

```
% visualizza i coefficienti di tutti i polinomi calcolati e l'errore di
% approssimazione corrispondente
celldisp(coeff_poli)
disp(errore)
% determina il grado del polinomio in corrispondenza del quale si ha
% l'errore minimo
errore(find(errore<=.5*10^-14))=0;
[min_errore,pos_min_errore] = min(errore);
coeff_pol = coeff_poli{pos_min_errore};

% valuta il polinomio di migliore approssimazione nei punti xnew
if ~isempty(xnew)
    ynew = polyval(coeff_pol,xnew);
    varargout{1} = ynew;
end
```

Osservazione: Nella funzione si pongono a zero gli errori di approssimazione che risultano inferiori o uguali a $0.5 \cdot 10^{-14}$.

Esempio 1

Dal Command Window Supponiamo di conoscere i valori della funzione $f(x) = x^4$ (incognita) sulla griglia degli interi x tali che $1 \leq x \leq 100$. Trovare il polinomio di migliore approssimazione per i dati a disposizione. A tal fine usiamo la funzione find_bestpol.m.

```
>> x = [1:100];
>> y = x.^4;
>> [coeff_pol] = find_bestpol(x,y);
coeff_poli{1} =
  1.0e+007 *
   0.08180798000000 -2.08096966000000

coeff_poli{2} =
  1.0e+006 *
   0.01744342857143 -0.94370648571429  9.14067025714289
```

```
coeff_poli{3} =
  1.0e+006 *
   0.00020200000000 -0.01315957142857  0.29881571428571 -1.57650034285711

coeff_poli{4} =
  Columns 1 through 4
   1.00000000000000 -0.00000000000034  0.00000000002487 -0.00000000072642
  Column 5
   0.00000000618095

coeff_poli{5} =
  Columns 1 through 4
  -0.00000000000000  1.00000000000001 -0.00000000000078  0.00000000003372
  Columns 5 through 6
  -0.00000000060876   0.00000000322184

coeff_poli{6} =
  Columns 1 through 4
   0.00000000000000 -0.00000000000000  1.00000000000005 -0.00000000000290
  Columns 5 through 7
   0.00000000008301 -0.00000000106551  0.00000000484500
```

```
coeff_poli{7} =
  Columns 1 through 4
  -0.00000000000000  0.00000000000000 -0.00000000000000  1.00000000000029
  Columns 5 through 8
  -0.00000000001384  0.00000000034828 -0.00000000413917  0.00000001714164

coeff_poli{8} =
  Columns 1 through 4
  -0.00000000000000  0.00000000000000 -0.00000000000000  0.00000000000004
  Columns 5 through 8
   0.99999999999783  0.00000000006844 -0.00000000118606  0.00000000992706
  Column 9
  -0.00000002894008

coeff_poli{9} =
  Columns 1 through 4
  -0.00000000000000  0.00000000000000 -0.00000000000000  0.00000000000000
  Columns 5 through 8
  -0.00000000000013  1.00000000000529 -0.00000000012804  0.00000000172519
  Columns 9 through 10
  -0.00000001116357  0.00000002460266
```

```
coeff_poli{10} =
  Columns 1 through 4
   0.00000000000000 -0.00000000000000  0.00000000000000 -0.00000000000000
  Columns 5 through 8
   0.00000000000000 -0.00000000000005  1.00000000000123 -0.00000000001540
  Columns 9 through 11
   0.00000000005338  0.00000000051068 -0.00000000259616

errore =
 1.0e+016 *
  Columns 1 through 4
   1.83734693275675  0.14778539999400  0.00226077716367  0.00000000000000
  Columns 5 through 8
   0.00000000000000  0.00000000000000  0.00000000000000  0.00000000000000
  Columns 9 through 10
   0.00000000000000  0.00000000000000
```

L'errore di approssimazione decresce al crescere del grado del polinomio approssimante ma diventa molto piccolo già dal quarto grado. Infatti, considerando i coefficienti del polinomio approssimante di grado 5,

```
>> coeff_pol
coeff_pol =
  Columns 1 through 4
  -0.00000000000000   1.00000000000001 -0.00000000000078   0.00000000003372

  Columns 5 through 6
  -0.00000000060876   0.00000000322184
```

si osserva che il coefficiente del monomio di grado massimo **è nullo** in precisione di macchina, mentre il coefficiente relativo al monomio di grado 4 è proprio 1

Se i dati in input sono affetti da errore

```
>> noise = 10^6*randn(1,length(y));
>> ydata = y + noise;
>> [coeff_pol_noise] = find_bestpol(x,ydata);
```

L'errore di approssimazione non cambia in modo significativo dal quarto grado in poi.

```
errore =
1.0e+016 *
  Columns 1 through 4
   1.83865002747658   0.14958824155293   0.01023540754299   0.00942206611061
  Columns 5 through 8
   0.00941161142473   0.00937567370055   0.00935613603876   0.00934454980644
  Columns 9 through 10
   0.00933311094961   0.00933042093266
>> coeff_pol_noise
coeff_pol_noise =
  1.0e+006 *
  Columns 1 through 4
   0.00000000000000 -0.00000000000006   0.00000000001192 -0.00000000114151
  Columns 5 through 8
   0.00000005584007 -0.00000083139602 -0.00004604518277   0.00264017610832
  Columns 9 through 11
  -0.05305544205054   0.44998990122359 -1.12224028234669
```

Esempio 2

Valutare la complessità asintotica di un algoritmo di ordinamento di cui si conoscono i tempi di esecuzione al variare della lunghezza $\mathbf{n}$ del dato di input, come riportato in tabella:

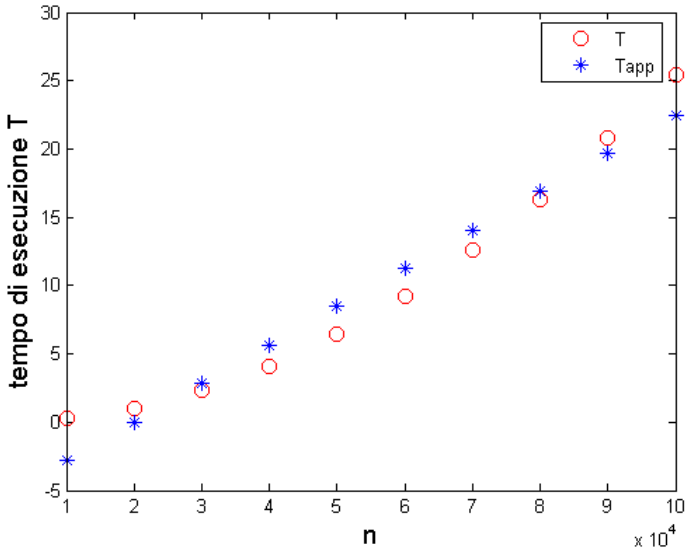
n	T (sec)
10000	0.312002
20000	0.998406
30000	2.293215
40000	4.102826
50000	6.411641
60000	9.204059
70000	12.604881
80000	16.317705
90000	20.826134
100000	25.474963

sapendo che il tempo di calcolo è una funzione polinomiale rispetto alla variabile $\mathbf{n}$ — precisamente $\mathbf{T = K\, n^P}$.

Soluzione

```
>> n = 10000:10000:100000;
>> T = [0.312002 0.998406 2.293215 4.102826 6.411641 9.204059...
        12.604881 16.317705 20.826134 25.474963];
>> [C,Tapp] = find_bestpol_mod(n,T,n,1);
% e' la funzione find_bestpol in cui il quarto
% input e' il grado massimo consentito al polinomio
% approssimante
% Valutiamo l'errore di approssimazione di T usando
% un polinomio di primo grado
coeff_poli{1} =
   0.0002810192593   -5.6014760666666
errore=
  34.21648743086648
```

Il grafico riporta i valori del tempo di calcolo T e quelli dati dalla approssimazione con un **polinomio di primo grado**

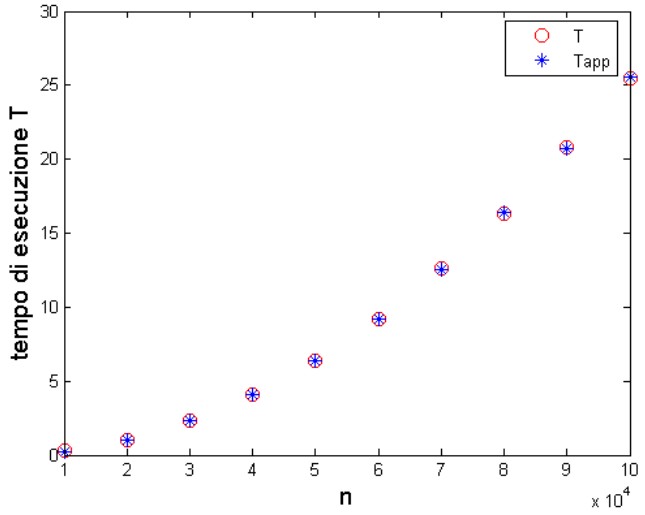



```
>> figure, plot(n,T,'ro'),
>> hold on, plot(n,Tapp,'*b')
```

Il tempo di calcolo T non dipende in modo perfettamente lineare da n.

Consideriamo ora un **polinomio di secondo grado**

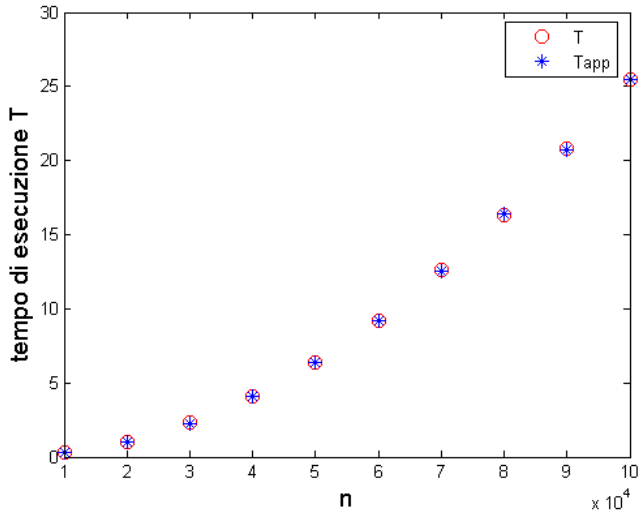
```
>> [C,Tapp] = find_bestpol_mod(n,T,n,2);
coeff_poli{2} =
   0.00000000254447   0.00000112747189  -0.00364031666667
errore=
  0.0320164301631 6
>> figure, plot(n,T,'ro'),
>> hold on, plot(n,Tapp,'*b')
```



In questo caso, l'errore di approssimazione è molto più piccolo e i valori di T sono ben approssimati da un ramo di parabola.

Considerando ora un **polinomio di terzo grado**

```
>> [C,Tapp] = find_bestpol_mod(n,T,n,3);
coeff_poli{3} =
  -0.00000000000000   0.00000000282247 -0.00001169437438   0.14092136666668
errore=
   0.02324800486742
>> figure, plot(n,T,'ro'),
>> hold on, plot(n,Tapp,'*b')
```



l'errore diminuisce anche se non in modo considerevole. Infatti, osservando il coefficienti del polinomio approssimante di terzo grado, si nota che il coefficiente del monomio di grado massimo è praticamente nullo (in precisione di macchina).

Quindi, si può concludere che la dipendenza del tempo di calcolo T dal numero di dati n è ben descritta da un polinomio di secondo grado.