Calcolo Numerico

(A.A. 2014-2015)

Lab n. 12

Approssimazione

17-12-2014

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 1

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**

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**:

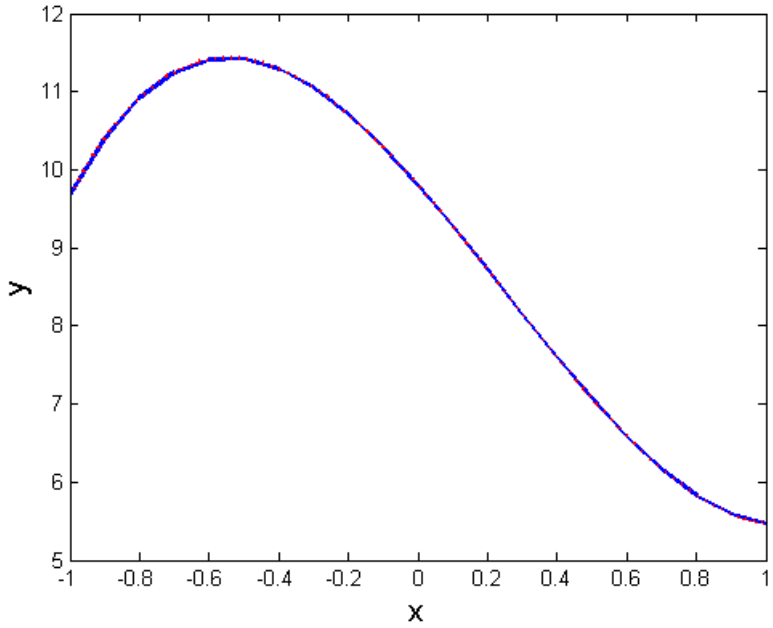
$$y_i = a_1\, x_i^n + a_2\, x_i^{n-1} + \cdot + a_n\, x_i + a_{n+1}$$

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')
```



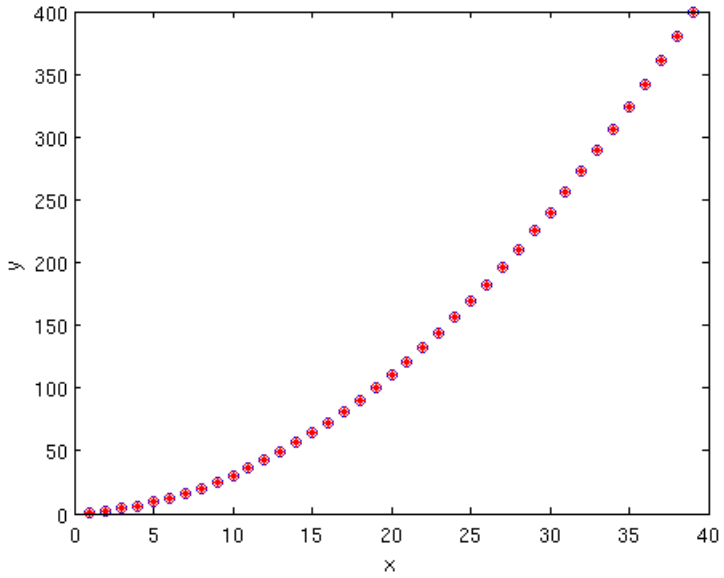
,

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')
```

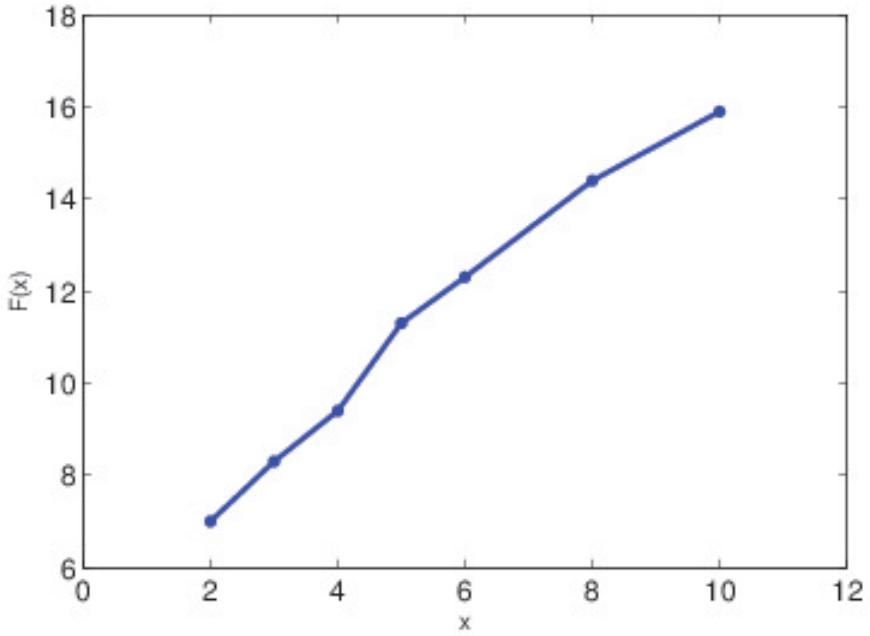

400
350
300
250
200
150
100
50
0
y
0
5
10
15
20
25
30
35
40
x
,

Esempio

Nella tabella sono riportate le **misure sperimentali** relative alla **forza** $F(x)$ necessaria per allungare una **molla** fino alla **lunghezza** x.

x	2	3	4	5	6	8	10
$F(x)$	7.0	8.3	9.4	11.3	12.3	14.4	15.9

Determinare la **costante elastica** della molla.

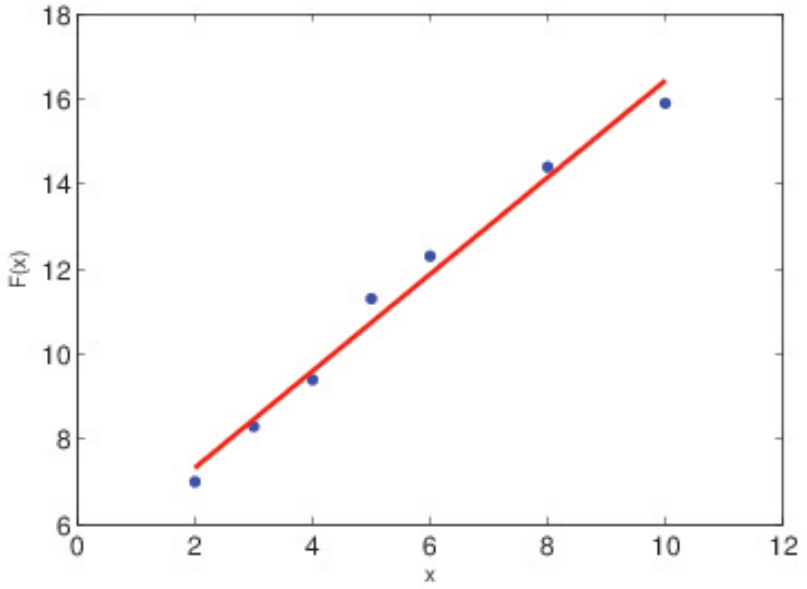


Soluzione

La forza $F(x)$ necessaria per allungare una molla fino alla lunghezza x è data da $\boxed{F(x) = k(x - l)}$ (**Legge di Hooke**) dove k è la **costante elastica** e l è la **lunghezza a riposo** della molla.

Per trovare la costante elastica della molla si devono approssimare i dati in tabella con il **polinomio ai minimi quadrati** di primo grado (**retta di regressione**): $\boxed{P_1(x) = a_0 + a_1\, x}$

Il coefficiente a_1 fornisce l'approssimazione della **costante elastica**.



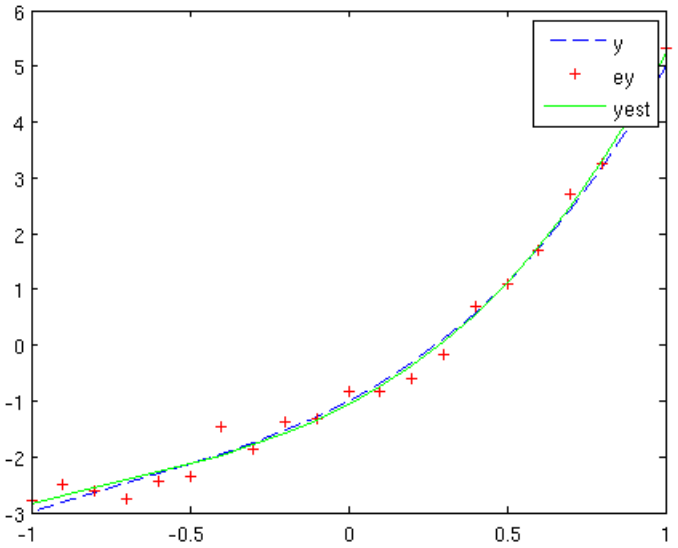

```
>> X=[2, 3, 4, 5, 6, 8, 10]
>> F=[7.0, 8.3, 9.4, 11.3, 12.3, 14.4, 15.9]
>> X2 = linspace(X(1),X(7))
>> p = polyfit(X,F,1);y2 = polyval(p,X2)
>> p

p =

1.1383 5.0491
```

Esercizio 2

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 = [1 2 3 -1];
>> 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 3

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.

Esercizio 4

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.

Esempio

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
```