Metodi numerici con elementi di Programmazione

A.A. 2013-2014

Esercizi svolti in Laboratorio

Lezione del 03-12-2013

Docente: Vittoria Bruni

Email: vittoria.bruni@sbai.uniroma1.it

Ufficio: Via A. Scarpa,
Pal. B, I piano, Stanza n. 16
Tel. 06 49766648

Ricevimento: Giovedì 14.00-15.00

Il materiale didattico è disponibile sul sito http://ingaero.uniroma1.it/ nella pagina dedicata al corso Metodi Numerici con elementi di Programmazione

Per consultazione: Getting Started with MatLab – The mathworks www.mathworks.com

Esercizio 1

- Scrivere la funzione Matlab num_cond.m che legga in input una matrice quadrata A e la variabile numerica tipo_norma che può essere uguale a 1, 2 o inf. La funzione restituisca in output il numero di condizionamento rispetto alla norma indicata nella variabile tipo_norma. Se quest' ultima non viene data in input, la funzione deve calcolare il numero di condizionamento rispetto alla norma infinito.

Si usi la funzione num_cond.m per calcolare il numero di condizionamento delle seguenti matrici

$$A = \begin{pmatrix} 1 & 1.01 \\ 0.99 & 1 \end{pmatrix} \qquad A = \begin{pmatrix} 1 & 101 \\ 99 & 1 \end{pmatrix} \qquad A = \begin{pmatrix} 2 & 3 \\ -1 & 8 \\ 0 & 4 \end{pmatrix}$$

Soluzione

- Determinare la lista delle variabili di input e di output

Variabili di input:

- matrice *A* di cui calcolare in numero di condizionamento
- tipo di norma (tipo_norma) rispetto al quale calcolare il condizionamento. tipo_norma può non essere data in input

Variabili di output:

- numero di condizionamento K

Controllo sugli input:

- controllo sul numero degli input
- la matrice deve essere quadrata
- il tipo di norma deve essere un numero pari a 1, 2, o $inf

- Aprire la finestra di Editor
- Salvare il file come num_cond.m nella directory di lavoro
- Scrivere la seguente istruzione che contiene la lista degli output e degli input della funzione

```
function [K] = num_cond(A,varargin)
```

La funzione prevede parametri di input opzionali!

- Scrivere lo help della funzione

```
% function [K] = num_cond(A,varargin)
% calcola il numero di condizionamento della matrice quadrata
% A rispetto alla norma indicata in tipo_norma.
% Se tipo_norma non è data in input, il condizionamento è calcolato
% rispetto alla norma infinito. tipo_norma non può assumere valori
% diversi da '1', '2' o 'inf'
```

- Controllo del numero delle variabili di input e controllo degli input

```
if nargin == 0 % non sono stati assegnati input

   error('E'' necessario definire una matrice!!!')

elseif nargin == 1  % è stato passato un solo input

        tipo_norma = inf;

elseif nargin == 2

        tipo_norma = varargin{1};

        if  (tipo_norma ~= 1) & (tipo_norma ~= 2) & (tipo_norma ~= inf)
            error('la norma deve essere 1 2 o inf')
        end
else
        error('troppe variabili di input!!!')
end
```

Nota: poiché è stato assegnato un solo input, allora **necessariamente** è la matrice (o comunque deve essere assegnato alla variabile che corrisponde alla matrice). In questo caso è necessario assegnare il valore di default alla variabile tipo_norma, necessaria per il calcolo del numero di condizionamento

Nota: poiché sono stati assegnati due input, allora **necessariamente** il primo è la matrice mentre il secondo deve essere assegnato alla variabile tipo_norma,

- Controllo del numero delle variabili di input

```
if nargin == 0 % non sono stati assegnati input

   error('E'' necessario definire una matrice!!!')

elseif nargin == 1  % è stato passato un solo input

        tipo_norma = inf;

elseif nargin == 2

        tipo_norma = varargin{1};

        if  (tipo_norma ~= 1) & (tipo_norma ~= 2) & (tipo_norma ~= inf)
            error('la norma deve essere 1 2 o inf')
        end
else
        error('troppe variabili di input!!!')
end
```

Nota: Lo input da assegnare a tipo_norma è il primo elemento della cell varargin, cioè quella contenente tutte le variabili di input opzionali con cui è stata chiamata la funzione

Nota: poiché sono stati assegnati più di due input, si da il messaggio di errore

- Scrivere la lista di istruzioni per il controllo degli input

```
dim = size(A);
if dim(1)~=dim(2)
   error('La matrice deve essere quadrata')
end
if det(A)==0
   error('La matrice e'' singolare')
end
```

Nota: il controllo su tipo_noma è stato fatto nel caso in cui nargin=2 quindi non va ripetuto

- Scrivere la lista di istruzioni per il calcolo del numero di condizionamento rispetto al tipo di norma scelto

```
IA = inv(A);                  % matrice inversa di A
NA = norm(A,tipo_norma);      % norma di A
NIA = norm(IA,tipo_norma);    % norma della matrice inversa
K = NA*NIA;                   % numero di condizionamento
```

- Salvare il file

Dalla finestra del Command Window

```
>> A = [1 1.01;0.99 1];
>> KA_1 = num_cond(A,1);
>> disp(KA_1)
    4.0401e+004

>> KA = num_cond(A);
>> disp(KA)
    4.0401e+004
```

Non è stato specificato il tipo di norma

```
>> KA = num_cond(A,-1);
??? Error using ==> num_cond at 18
la norma deve essere 1 2 o inf
```

il valore di input assegnato per tipo di norma non è accettabile

```
>> A = [2  3; -1  8; 0  4];
>>KA_inf =num_cond(A,inf);
??? Error using ==> num_cond at 10
La matrice deve essere quadrata
```

Esercizio 2

- Modificare la funzione Matlab num_cond.m e chiamarla num_cond2.m. La funzione deve prevedere come ulteriori variabili di output, se richieste, la matrice inversa di *A* e la norma di *A* calcolata rispetto a quanto indicato in tipo_norma.

- Aprire la finestra di Editor
- Salvare il file come num_cond2.m nella directory di lavoro
- Scrivere la seguente istruzione che contiene la lista degli output e degli input della funzione

```
function [K,varargout] = num_cond2(A,varargin)
```

La funzione prevede parametri opzionali sia di input che di output

- Scrivere lo help della funzione

```
% function [K,varargout] = num_cond2(A,varargin)
% calcola il numero di condizionamento K della matrice quadrata
% A rispetto alla norma indicata in tipo_norma.
% Se tipo_norma non è data in input, il condizionamento è calcolato
% rispetto alla norma infinito. tipo_norma non può assumere valori
% diversi da '1', '2' o 'inf'
% Se si chiedono due output, allora il primo è K e il secondo è la
% matrice inversa di A
% Se si chiedono tre output, il primo è K, il secondo è la matrice
% inversa di A, il terzo è la norma di A
```

- Controllo del numero delle variabili di input

```
if nargin == 0 % non sono stati assegnati input
   error('E'' necessario definire una matrice!!!')
elseif nargin == 1  % è stato passato un solo input
        tipo_norma = inf;
elseif nargin == 2 % sono stati specificati due input
        tipo_norma = varargin{1};
        if  (tipo_norma ~= 1) & (tipo_norma ~= 2) & (tipo_norma ~= inf)
            error('la norma deve essere 1 2 o inf')
        end
else   % sono stati assegnati input più di due input
        error('troppe variabili di input!!!')
end
```

- Lista di istruzioni per il controllo degli input

```
dim = size(A);
if dim(1)~=dim(2)
   error('La matrice deve essere quadrata')
end
if det(A)==0
   error('La matrice e'' singolare')
end
```

- Lista di istruzioni per il calcolo del numero di condizionamento rispetto al tipo di norma scelto

```
IA = inv(A);                    % matrice inversa di A
NA = norm(A,tipo_norma);        % norma di A
NIA = norm(IA,tipo_norma);      % norma della matrice inversa
K = NA*NIA;                     % numero di condizionamento
```

- Controllo e assegnazioni output

```
if nargout == 2 % sono stati richiesti  due output
    varargout{1} = IA;
elseif nargout == 3  %sono stati richiesti  tre output
        varargout{1} = IA;
        varargout{2} = NA;
elseif nargout >3% sono stati richiesti  troppi output
     error('troppe variabili di output!!!')
end
```

Nota: poiché sono stati richiesti due output, allora **necessariamente** il primo è K mentre il secondo è l'inversa di A, che quindi diventa il primo elemento della variabile di tipo cell varargout, che contiene tutti gli output opzionali richiesti

- Lista di istruzioni per il calcolo del numero di condizionamento rispetto al tipo di norma scelto

```
IA = inv(A);                    % matrice inversa di A
NA = norm(A,tipo_norma);        % norma di A
NIA = norm(IA,tipo_norma);      % norma della matrice inversa
K = NA*NIA;                     % numero di condizionamento
```

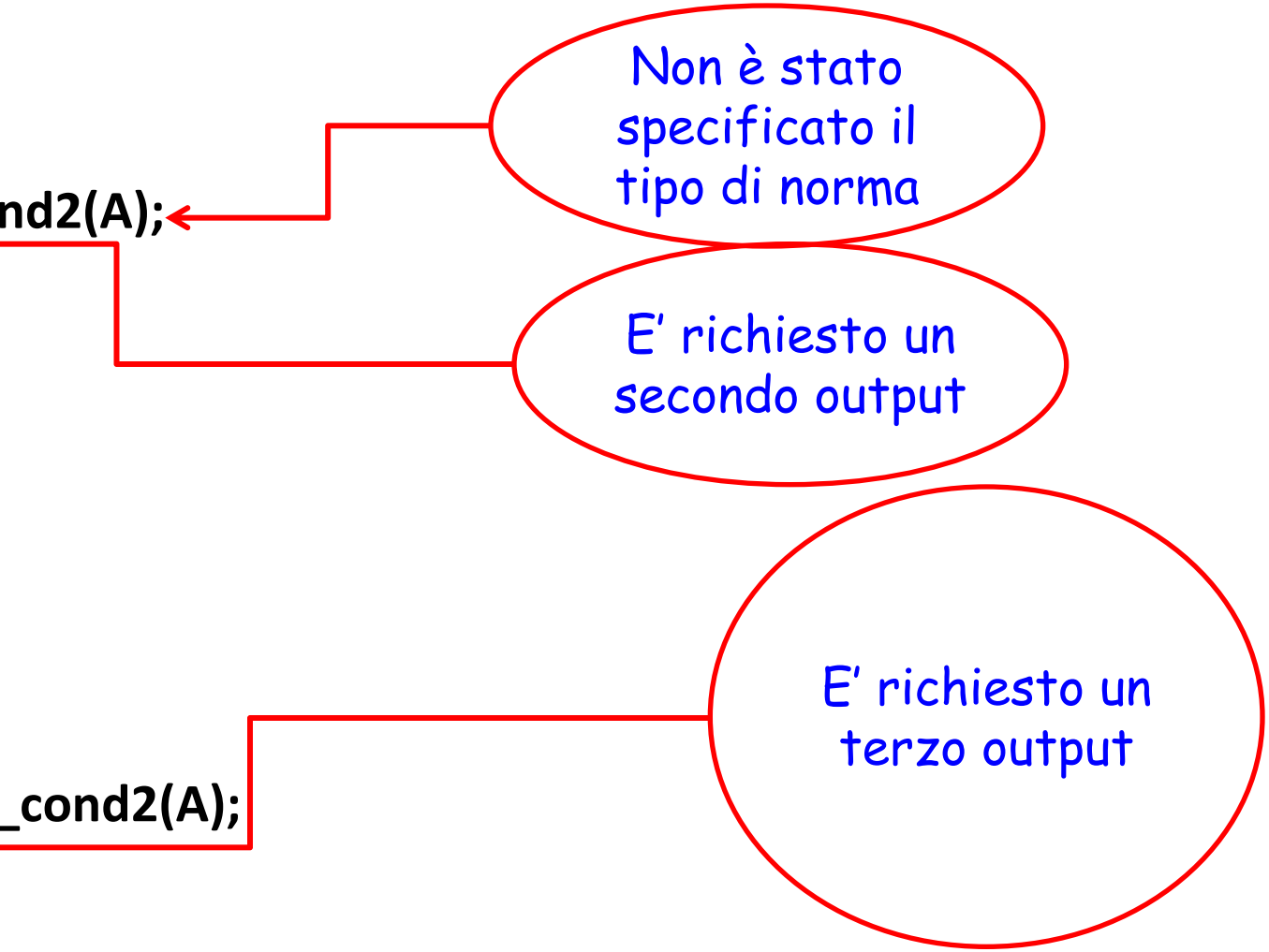
- Controllo e assegnazioni output

```
if nargout == 2 % sono stati richiesti  due output
    varargout{1} = IA;
elseif nargout == 3  %sono stati richiesti  tre output
        varargout{1} = IA;
        varargout{2} = NA;
elseif nargout >3% sono stati richiesti troppi output
     error('troppe variabili di output!!!')
end
```

Nota: poiché sono stati richiesti tre output, allora **necessariamente** il primo è K, il secondo è l'inversa di A, che diventa il primo elemento della variabile di tipo cell varargout, mentre il terzo è la norma di A che diventa il secondo elemento di varargout

Dalla finestra del Command Window

```
>> A = [1 1.01;0.99 1];
>> KA_1 = num_cond2(A,1);
>> disp(KA_1)
    4.0401e+004

>> [KA,B] = num_cond2(A);
>> disp(KA)
    4.0401e+004
>> disp(B)
  1.0e+004 *
   1.0000   -1.0100
  -0.9900    1.0000

>> [KA,B,nA] = num_cond2(A);
>> disp(nA)
   2.0100
```



Alcune funzioni

y = polyval(a,x)

Valuta il polinomio, i cui coefficienti sono contenuti nel vettore a, in corrispondenza degli elementi del vettore x.
Il grado massimo n del polinomio è pari alla lunghezza del vettore a diminuita di 1.
Il primo elemento di a è il coefficiente del monomio di grado massimo.

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

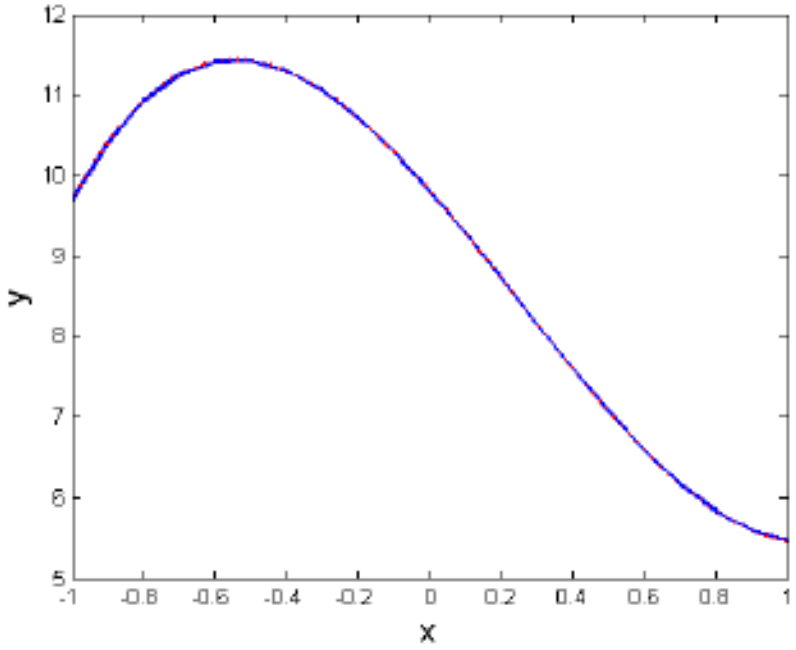
Esempio

```
>> a=[1 2 8]

a =

     1     2     8

>> polyval(a,5)

ans =

    43
```

Valuto il polinomio di secondo grado x^2+2x+8 in corrispondenza del punto x=5

```
>> 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')
```

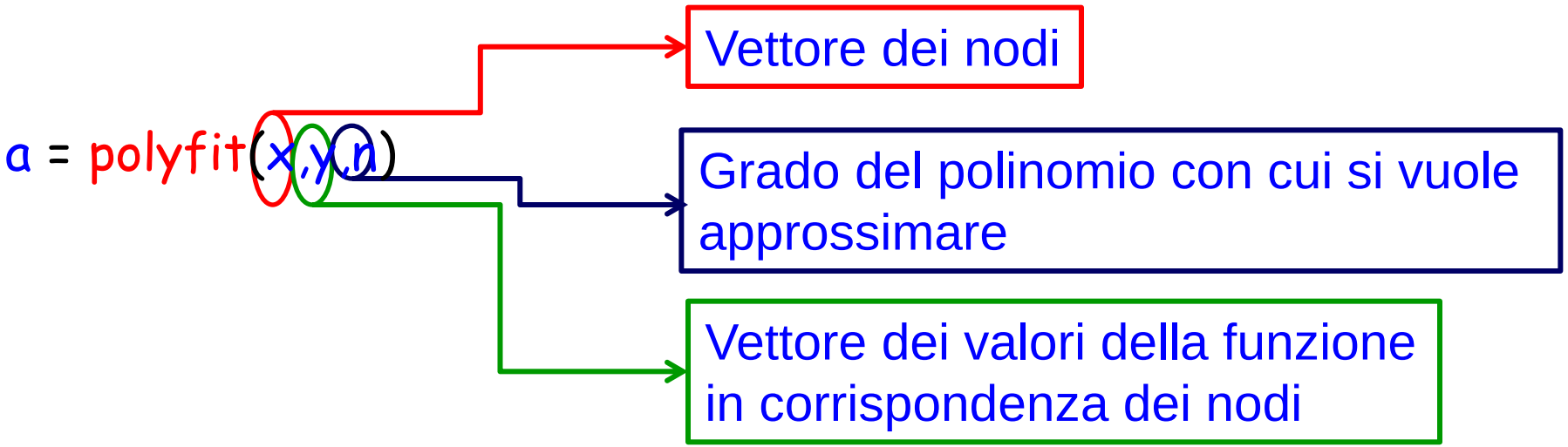
Valuto il polinomio di terzo grado $3x^3-2.23x^2-5.1x+9.8$ in corrispondenza degli elementi del vettore x



Alcune Funzioni

a = polyfit(x,y,n)

Calcola i coefficienti a del polinomio di grado n che approssima i valori in y corrispondenti ai nodi in x usando la tecnica dei minimi quadrati



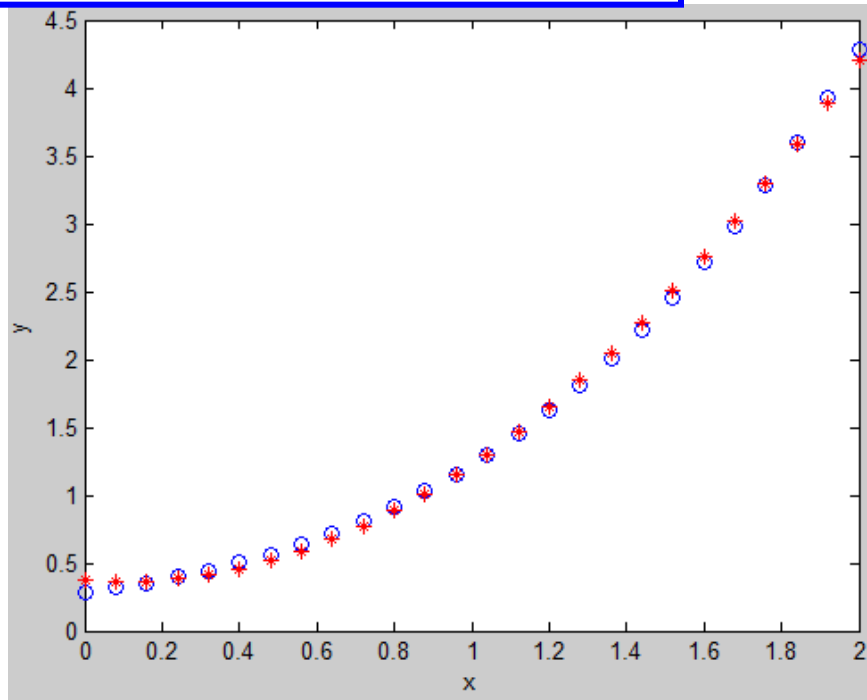
Esempio

```
>>coeff = [0.74 0.97 1.1 0.86]/3;
>>x = 0:.08:2;
>>y = polyval(coeff,x);
```

Approssimo i dati nel vettore y con un polinomio di secondo grado nel senso dei minimi quadrati

```
>>a = polyfit(x,y,2);
>>yn = polyval(a,x);
>>figure, plot(x,y,'bo')
>>hold on
>>plot(x,yn,'r*')
>>xlabel('x')
>>ylabel('y')
```

Calcolo i coefficienti del polinonio e valuto il polinomio stimato in corrispondenza dei nodi



Esercizio 3

Scrivere la funzione matlab find_bestpol.m 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.
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 è possibile passare i punti  in cui stimare
% la funzione approssimante che viene restituita come 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 output
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. Si pongono a zero gli errori di approssimazione che risultano inferiori o
% uguali a 0.5*10^14
errore(find(errore<=.5*10^-14))=0;
[min_errore,pos_min_errore] = min(errore);
coeff_pol = coeff_poli{pos_min_errore(1)};

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

Dal Command Window

Supponiamo di conoscere i valori della funzione $f(x) = x^4$ (incognita) sulla griglia degli interi x nell'intervallo $[1, 100]$.
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 non cambia in modo significativo dal 4° 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
```

Esercizio 4

«per casa»

Modificare la funzione find_bestpol.m aggiungendo un'ulteriore variabile di input che indica il massimo valore consentito al grado del polinomio approssimante. Usare la funzione per valutare la complessità asintotica di un algoritmo di ordinamento di cui si conoscono i tempi di esecuzione al variare della lunghezza 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 n, precisamente $T = K n^p$.

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);

coeff_poli{1} =
    0.00028101925939 -5.60147606666666
errore=
    34.21648743086648

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

Si valuta l'errore di approssimazione di T usando un polinomio di primo grado

Il grafico confronta i valori del tempo di calcolo T e quelli dati dalla approssimazione con un polinomio di primo grado.
Osservare che il tempo di calcolo T non dipende in modo perfettamente lineare da n

```
>> [C,Tapp] = find_bestpol_mod(n,T,n,2);
coeff_poli{2} =
    0.00000000254447 0.00000112747189 -0.00364031666667
errore=
    0.03201643016316

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

Si valuta l'errore di approssimazione di T usando un polinomio di secondo grado

Osservare che in questo caso l'errore di approssimazione è molto più piccolo e i valori di T sono ben approssimati da un ramo di parabola

```
>> [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')
```

Si valuta l'errore di approssimazione di T usando un polinomio di terzo grado

Osservare che in questo caso, l'errore diminuisce anche se non in modo considerevole. Infatti, il coefficiente del monomio di grado massimo del polinomio approssimante è nullo in precisione di macchina.
Quindi, la dipendenza del tempo di calcolo T dal numero di dati n è ben descritta da un polinomio di secondo grado.