Polinomi e interpolazione

Un polinomio può essere assegnato in Matlab attraverso l'array dei suoi coefiicienti, ordinati secondo le potenze decrescenti.

Grafica

Esempio. Disegniamo il grafico in [-5,5] del polinomio

p(x)=x^5-3x^4+2x^2-x+7

Possiamo scrivere

```
>> p=[1 -3 0 2 -1 7];
>> x=linspace(-5,5,100);
>> y=polyval(p,x);  % è la funzione Matlab che valuta polinomi
>> plot(x,y)
>> title('x^5-3x^4+2x^2-x+7')
```

Esercizio. Scrivere un programma in grado di disegnare il grafico di un qualunque polinomio, che richieda in input il grado del polinomio e quindi successivamente i suoi coefficienti e infine l'intervallo su cui disegnare il grafico.

Operazioni tra polinomi

Allo stesso modo possiamo moltiplicare o dividere tra loro polinomi attraverso gli array dei loro coefficienti.

Esempi.

```
>> p=[1 -3 0 2 -1 7];
>> q=[2 -1 5 1];
>> z=conv(p,q)     % prodotto tra polinomi
```

fornisce il vettore dei coefficienti del polinomio prodotto di grado 8:

```
z =  2    -7     8   -10    -7    25   -10    34     7
```

```
>> [w,r]=deconv(p,q)     % quoziente tra polinomi
```

fornisce i vettori

```
w =   0.5000   -1.2500   -1.8750    %(coeff. del polinomio quoziente)

r =      0         0         0    5.8750    9.6250    8.8750   % (coeff. del polinomio resto)
```

Fate la prova:

```
>> conv(w,q)+r
```

deve restituire p .

Radici di polinomi

Il problema di trovare le radici di un polinomio è abbastanza complicato, specie se alcune sono multiple o molto vicine tra loro, o se vogliamo calcolare anche quelle complesse.
Matlab fornisce una function molto potente in grado di risolvere tutto questo: *roots .*
L'idea è quella che le radici di un polinomio coincidono con gli autovalori di una particolare matrice che si costruisce a partire dai coefficienti, autovalori a loro volta calcolabili con la funzione *eig*.

Provare, noto l'array *p*:

>> r= roots(p) % *r* è il vettore colonna 1xn delle radici del polinomio di grado n

>> A=diag(ones(n-1,1),-1); A(1,:)=-p(2:n+1)./p(1); s=eig(A)

e confrontare gli array *r* e *s*.

Esiste anche la funzione inversa di *roots*, *poly*, cioè quella in grado di identificare i coefficienti del polinomio che ha un set di radici assegnate. Con le notazioni precedenti

>> pp=poly(r)

e confrontare *pp* con *p*.

Approssimazione polinomiale nel senso dei minimi quadrati

A volte avendo a disposizione dei dati sperimentali o numerici, è utile individuare il polinomio di grado assegnato che meglio li approssima nel senso dei minimi quadrati (minimizzando cioè lo scarto quadratico medio): è il problema del *data fitting*.
Matlab consente di farlo con la funzione *polyfit* :

>> p=polyfit(x,y,n)

restituisce in p i coefficienti del polinomio di grado n che meglio approssima il set di valori (x,y).Considerate il seguente esempio:

```
% DATA FITTING: criterio ai minimi quadrati
x=0:.025:4;
y=exp(x);
p2=polyfit(x,y,2);%resistuisce i coeff di un polinomio di grado 2 che
                  % approssima (x,y) nel senso dei minimi quadrati
xx=0:.02:4;
yy=polyval(p2,xx);

  plot(x,y,'o',xx,yy);
  axis([0 4 0 60])
hold on
  p3=polyfit(x,y,3);
  yy=polyval(p3,xx);
  plot(xx,yy,'r')
  title('data fitting')
  legend('exp','p2','p3')
hold off
```

Interpolazione polinomiale

Molte volte abbiamo la necessità di ricostruire una funzione incognita F avendo a disposizione un set di dati (x,y=F(x)) (dove x e y sono due array della stessa dimensione N, con x_1<x_2<..<x_N), e vogliamo ricavare informazioni sui possibili valori assunti da questa funzione in punti diversi dai precedenti.
Per farlo abbiamo a disposizione le tecniche di interpolazione, che costruiscono a partire da x delle funzioni polinomiali a tratti che forniscono poi i valori cercati.
Matlab fornisce la function *interp1* per l'interpolazione in una dimensione. Analizziamo il seguente script *interpunod.m* :

```
%INTERPOLAZIONE usando POLINOMI
clear
x=[0 pi/4 pi/2 3*pi/4 pi];
y=sin(x);% (x,y) DATI per costruire POL. DI INTERPOLAZIONE

%xi=2.12312% punto su cui interpolo
x0=input('Punto su cui valutare l''interpolata (in [0,pi]): ');
yl=interp1(x,y,x0);%interpolazione lineare dei dati (x,y) nel punto xi
ys=interp1(x,y,x0,'spline');%interpolazione con spline cubiche
yh=interp1(x,y,x0,'cubic');%interpolazioni cubica composita di Hermite

yes=sin(x0);
err1=abs(yl-yes)
errs=abs(ys-yes)
err3=abs(yh-yes)

xi=linspace(0,pi,51);% xi vettore di punti su cui interpolo
% x deve contenere elementi crescenti
yl=interp1(x,y,xi);%interpolazione lineare dei dati (x,y) nei punti xi
ys=interp1(x,y,xi,'spline');%interpolazione con spline cubiche
yh=interp1(x,y,xi,'cubic');%interpolazioni cubica composita di Hermite

plot(x,y,'o',xi,yl,xi,ys,'r',xi,yh,'k',x0,yes,'*');
title('Confronto tra differenti interpolazioni');
axis([0 pi 0 1]);
legend('data','linear','spline','cubic','x0')
```

Scrittura e lettura da file

Spesso è comodo salvare un file di dati che poi può essere richiamato da un altro programma. Per scrivere possiamo ad esempio utilizzare i comandi di C++ importati da Matlab.

Esempio:

```
x=0:0.4:2;
y=[x;exp(x).*cos(x)];  % array 2x6
fid=fopen('val.dat','w');     %  apertura  del  file  val.dat  in
scrittura
fprintf(fid,'%6.2f %12.8f\n',y);  % legge y per colonne e scrive
                                  % per righe secondo il formato
fclose(fid);
```

Produrrà il file *val.dat* con i dati su due colonne (x,f(x)), che potremo rileggere in modo analogo facendo ricorso al comando *fscanf* . In alternativa, più semplicemente, potremo caricare il file con i seguenti comandi:

```
load val.dat   % crea l'array val (6x2)
x=val(:,1);  % x sarà la prima colonna di val
y=val(:,2);  % y sarà la seconda colonna di val
```

Esercizio.
Modificare il programma *interpunod.m*, in modo che legga un file di dati *valori.dat* contenente su due righe gli array x e y della stessa dimensione N a partire dai quali costruire l'interpolazione, e un file *points.dat* contenente l'array z=[z_1,..,z_M] di dimensione M dei punti sui quali calcolare i valori interpolati. Nel grafico finale mettere a confronto le tre funzioni interpolate (In questo caso non ci sono valori esatti con cui confrontare). Fare attenzione a costruire la finestra grafica in accordo con i dati letti. Il programma stamperà in uscita una tabella con M righe e 4 colonne, contenenti risp. i punti z e i valori corrispondenti dell'interpolazione linear, spline e cubic.

La stessa cosa si può fare in due dimensioni, utilizzando questa volta la function *interp2* di Matlab. Guardiamo il seguente programma:

```
%Interpolazione in dimensione 2
close all
x=linspace(-pi,pi,5);

[X,Y]=meshgrid(x);

Z=cos(X).*sin(Y);
mesh(X,Y,Z)
title('data');

xi=0.23123;
yi=0.123123;
zi=interp2(X,Y,Z,xi,yi);%zi=valore interpolato dai nodi (X(i,j),Y(i,j),Z(i,j))
                        %   nel punto (xi,yi)
                        % default=interpolazione lineare

Fes=cos(xi)*sin(yi);
err1=abs(Fes-zi)

zi=interp2(X,Y,Z,xi,yi,'cubic');
err3=abs(Fes-zi)

xi=linspace(-pi,pi,100);
[XI,YI]=meshgrid(xi);

ZI=interp2(X,Y,Z,XI,YI);%ZI=matrice di valori interpolati dai nodi (X,Y,Z)
                        %   nei punti (XI,YI)
                        % default=interpolazione lineare
figure
mesh(XI,YI,ZI)
title('linear')

ZI=interp2(X,Y,Z,XI,YI,'cubic');
figure
mesh(XI,YI,ZI)
title('cubic')
```