

# Calcolo Scientifico - a.a. 2003/04

## Soluzione homework

prof.ssa Rossana Vermiglio, dott. Dimitri Breda  
Dipartimento di Matematica e Informatica  
Università degli Studi di Udine  
vermiglio,dbreda@dimi.uniud.it  
<http://www.dimi.uniud.it/~rossana,dbreda>

### Esercizio 1 : Equazioni non lineari

*Scrivi un programma MATLAB per approssimare lo zero di una funzione assegnata  $f(x)$  mediante il metodo di Halley*

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \left( 1 - \frac{f(x_k)f''(x_k)}{2f'(x_k)^2} \right)^{-1}$$

*In particolare la FUNCTION deve ritornare un valore approssimato dello zero, assegnando in input un valore  $x_0$  iniziale, la funzione  $f$  con le derivate  $f'$ ,  $f''$  e una prefissata tolleranza  $tol$ . Testa il metodo sui seguenti problemi:*

1.  $f(x) = x^n - 2$ ,  $n = 3, 4, 5$ ;
2.  $f(x) = a - e^x$ ,  $a > 0$ ;

*e confronta i risultati ottenuti con quelli forniti dalla funzione FZERO di MATLAB. Che ordine di convergenza ha il metodo di Halley? Sai stimarlo in base ai risultati ottenuti?*

**Soluzione.** Una possibile implementazione del metodo è la seguente:

```
function [x,e,k,p,L]=halley(f,df,ddf,x0,TOL);

% Metodo di Halley per l'approssimazione degli zeri di una funzione
% 'f'=funzione
% 'df'=derivata prima
% 'ddf'=derivata seconda
% 'x0'=valore iniziale
% 'TOL'=tolleranza per il criterio di arresto
% 'x'=vettore delle approssimazioni (iterate)
% 'e'=vettore errore relativo approssimato
% 'k'=numero iterazioni
```

```

% 'p'=stima ordine di convergenza
% 'L'=stima fattore di convergenza

k=1;
dx(k)=-f(x0)/df(x0)*(1-f(x0)*ddf(x0)/(2*df(x0)^2))^(1);
x(k)=x0+dx(k);
e(k)=abs(dx(k))/min(abs(x(k)),abs(x0));
while (e(k)>TOL)&(k<=100)
    dx(k+1)=-f(x(k))/df(x(k))*(1-f(x(k))*ddf(x(k))/(2*df(x(k))^2))^(1);
    x(k+1)=x(k)+dx(k+1);
    e(k+1)=abs(dx(k+1))/min(abs(x(k+1)),abs(x(k)));
    k=k+1;
end;
dx=abs(dx);
p=log(dx(2:k))./log(dx(1:k-1));
L=dx(2:k)./(dx(1:k-1).^3);

%display
disp(['k', 'x(k)', 'e(k)', 'p(k)', 'L(k)'])
disp([sprintf('%d %15.16f',0,x0)])
disp([sprintf('%d %15.16f %0.16f',1,x(1),e(1))])
for i=2:k
    disp([sprintf('%d %15.16f %0.16f %0.3g %3.5f',i,x(i),e(i),p(i-1),L(i-1))])
end

%plot
subplot(2,2,1)
semilogy((1:k),e,'-b.', 'markersize',10)
grid on
title('errore relativo')
xlabel('k');ylabel('e_{k}')
subplot(2,2,2)
loglog(e(1:k-1),e(2:k),'-g.', 'markersize',10)
grid on
title('stima pendenza')
xlabel('e_{k}');ylabel('e_{k+1}')
subplot(2,2,3)
plot(2:k,p,'-r.', 'markersize',10);
title('stima ordine')
xlabel('k');ylabel('p')
subplot(2,2,4)
plot(2:k,L,'-k.', 'markersize',10);
title('stima fattore')
xlabel('k');ylabel('L')

```

In particolare, si è considerato il seguente criterio d'arresto:

$$\frac{|x_{k+1} - x_k|}{\min\{|x_k|, |x_{k+1}|\}} \leq TOL$$

in cui l'errore assoluto (al numeratore) è stato approssimato con la distanza tra due iterate successive.

*halley* è stato testato sui problemi assegnati, i quali ammettono sempre uno zero positivo  $\alpha \in (1, 2)$  (in particolare si è scelto  $a = 3$  per il secondo problema). Si è dunque scelto come valore iniziale  $x_0 = 1$  in tutti i casi. Inoltre si è fissata una tolleranza  $TOL = 10^{-15}$  per il criterio d'arresto, con numero di iterazioni massimo pari a 100.

Le funzioni e le loro derivate prime e seconde sono state create come oggetti *inline* con la seguente *function* ausiliaria:

```
function [f,df,ddf]=effe(i)

if i==1
    f=inline('x^3-2');df=inline('3*x^2');ddf=inline('6*x');
elseif i==2
    f=inline('x^4-2');df=inline('4*x^3');ddf=inline('12*x^2');
elseif i==3
    f=inline('x^5-2');df=inline('5*x^4');ddf=inline('20*x^3');
elseif i==4
    f=inline('3-exp(x)');df=inline('-exp(x)');ddf=inline('-exp(x)');
end
```

Per ciascun problema si riportano di seguito i risultati ottenuti con *halley* seguiti dai risultati forniti da *fzero*.

```
>> [f,df,ddf]=effe(1)
f =
    Inline function:
    f(x) = x^3-2
df =
    Inline function:
    df(x) = 3*x^2
ddf =
    Inline function:
    ddf(x) = 6*x

>> [x,e,k,p,L]=halley(f,df,ddf,1,1e-15);
Warning: Log of zero.
> In D:\matlab\didattica\corso_CS_2004\HW\halley.m at line 26
k      x(k)      e(k)      p(k)      L(k)
0 1.0000000000000000
1 1.2500000000000000 0.2500000000000000
```

```

2 1.2599206349206349 0.0079365079365079 3.33 0.63492
3 1.2599210498948732 0.0000003293653797 3.19 0.42501
4 1.2599210498948732 0.0000000000000000 Inf 0.00000

```

```
>> [z,fval]=fzero(f,1,optimset('Display','iter','TolX',1e-15))
```

Func-count	x	f(x)	Procedure
1	1	-1	initial
2	0.971716	-1.08248	search
3	1.02828	-0.912725	search
4	0.96	-1.11526	search
5	1.04	-0.875136	search
6	0.943431	-1.16029	search
7	1.05657	-0.820513	search
8	0.92	-1.22131	search
9	1.08	-0.740288	search
10	0.886863	-1.30246	search
11	1.11314	-0.620741	search
12	0.84	-1.4073	search
13	1.16	-0.439104	search
14	0.773726	-1.53681	search
15	1.22627	-0.155992	search
16	0.68	-1.68557	search
17	1.32	0.299968	search

Looking for a zero in the interval [0.68, 1.32]

18	1.22331	-0.169328	interpolation
19	1.2582	-0.00819634	interpolation
20	1.25992	1.86329e-005	interpolation
21	1.25992	-2.55115e-008	interpolation
22	1.25992	-7.92699e-014	interpolation
23	1.25992	0	interpolation

Zero found in the interval: [0.68, 1.32].

```

z =
    1.259921049894873e+000

```

```

fval =
    0

```

```
>> [f,df,ddf]=effe(2)
```

```

f =
    Inline function:
    f(x) = x^4-2

```

```

df =
    Inline function:
    df(x) = 4*x^3
ddf =
    Inline function:
    ddf(x) = 12*x^2

>> [x,e,k,p,L]=halley(f,df,ddf,1,1e-15);
k      x(k)      e(k)      p(k)  L(k)
0 1.0000000000000000
1 1.1818181818181819 0.1818181818181818
2 1.1892067551093526 0.0062518697079136 2.88 1.22927
3 1.1892071150027210 0.0000003026331350 3.02 0.89226
4 1.1892071150027210 0.0000000000000000 2.56 708.08632

>> [z,fval]=fzero(f,1,optimset('Display','iter','TolX',1e-15))

```

Func-count	x	f(x)	Procedure
1	1	-1	initial
2	0.971716	-1.10843	search
3	1.02828	-0.881972	search
4	0.96	-1.15065	search
5	1.04	-0.830141	search
6	0.943431	-1.20779	search
7	1.05657	-0.753792	search
8	0.92	-1.28361	search
9	1.08	-0.639511	search
10	0.886863	-1.38138	search
11	1.11314	-0.464695	search
12	0.84	-1.50213	search
13	1.16	-0.189361	search
14	0.773726	-1.64162	search
15	1.22627	0.261259	search

Looking for a zero in the interval [0.77373, 1.2263]

16	1.16414	-0.163369	interpolation
17	1.18805	-0.00780238	interpolation
18	1.18921	1.77619e-005	interpolation
19	1.18921	-2.60439e-008	interpolation
20	1.18921	-8.68194e-014	interpolation
21	1.18921	-2.22045e-016	interpolation
22	1.18921	1.59872e-014	interpolation

Zero found in the interval: [0.77373, 1.2263].

z =

```

1.189207115002721e+000
fval =

-2.220446049250313e-016
-----

>> [f,df,ddf]=effe(3)
f =
    Inline function:
    f(x) = x^5-2
df =
    Inline function:
    df(x) = 5*x^4
ddf =
    Inline function:
    ddf(x) = 20*x^3

>> [x,e,k,p,L]=halley(f,df,ddf,1,1e-15);
k      x(k)      e(k)      p(k)  L(k)
0 1.0000000000000000
1 1.1428571428571428 0.1428571428571429
2 1.1486980506142963 0.0051107942875094 2.64 2.00343
3 1.1486983549970351 0.0000002649806348 2.92 1.52749
4 1.1486983549970351 0.0000000000000001 2.45 3617.81333

>> [z,fval]=fzero(f,1,optimset('Display','iter','TolX',1e-15))

Func-count      x      f(x)      Procedure
1              1      -1      initial
2      0.971716      -1.13364      search
3      1.02828      -0.850349      search
4      0.96      -1.18463      search
5      1.04      -0.783347      search
6      0.943431      -1.2526      search
7      1.05657      -0.683295      search
8      0.92      -1.34092      search
9      1.08      -0.530672      search
10     0.886863      -1.45137      search
11     1.11314      -0.290995      search
12     0.84      -1.58179      search
13     1.16      0.100342      search

Looking for a zero in the interval [0.84, 1.16]

14     1.14091      -0.0668756      interpolation
15     1.14855      -0.00132933      interpolation

```

```

16          1.1487  5.29744e-007      interpolation
17          1.1487 -1.40898e-010      interpolation
18          1.1487  8.88178e-016      interpolation
19          1.1487 -1.84297e-014      interpolation
Zero found in the interval: [0.84, 1.16].

```

```

z =
    1.148698354997035e+000
fval =
    8.881784197001252e-016
-----

```

```

>> [f,df,ddf]=effe(4)
f =

```

```

    Inline function:
    f(x) = 3-exp(x)

```

```

df =
    Inline function:
    df(x) = -exp(x)

```

```

ddf =
    Inline function:
    ddf(x) = -exp(x)

```

```

>> [x,e,k,p,L]=halley(f,df,ddf,1,1e-15);

```

```

Warning: Log of zero.

```

```

> In D:\matlab\didattica\corso_CS_2004\HW\halley.m at line 26

```

```

k          x(k)          e(k)          p(k)  L(k)
0 1.0000000000000000
1 1.0985324543253132 0.0985324543253131
2 1.0986122886680674 0.0000726736314797 4.07 0.08345
3 1.0986122886681098 0.00000000000000385 3.26 0.08320
4 1.0986122886681098 0.00000000000000000 Inf 0.00000

```

```

>> [z,fval]=fzero(f,1,optimset('Display','iter','TolX',1e-15))

```

Func-count	x	f(x)	Procedure
1	1	0.281718	initial
2	0.971716	0.357526	search
3	1.02828	0.203736	search
4	0.96	0.388304	search
5	1.04	0.170783	search
6	0.943431	0.431219	search
7	1.05657	0.123516	search
8	0.92	0.49071	search
9	1.08	0.0553204	search
10	0.886863	0.572498	search

```

11          1.11314      -0.0438924      search

Looking for a zero in the interval [0.88686, 1.1131]

12          1.09702      0.00475996      interpolation
13          1.0986      3.45213e-005      interpolation
14          1.09861 -2.64545e-010      interpolation
15          1.09861  1.77636e-015      interpolation
16          1.09861 -4.88498e-015      interpolation
Zero found in the interval: [0.88686, 1.1131].

z =
    1.098612288668109e+000
fval =
    1.776356839400251e-015
-----

```

La stima dell'ordine  $p$  e del fattore  $L$  di convergenza è basata sulla definizione classica (vedi appunti sulle equazioni non lineari) dove  $|x_{k+1} - \alpha|$  è sostituito con  $|x_{k+1} - x_k|$ . Il metodo di Halley converge con ordine 3 se lo zero  $\alpha$  è semplice. Lo si può dimostrare (con un po' di pazienza) calcolando le derivate del corrispondente metodo di iterazione di punto fisso

$$g(x) = x - \frac{f(x)}{f'(x)} \left( 1 - \frac{f(x)f''(x)}{2f'(x)^2} \right)^{-1}$$

e osservando che  $g'(\alpha) = g''(\alpha) = 0$  e  $g'''(\alpha) \neq 0$

Lo si può anche dedurre dalle stime fornite anche se l'approssimazione non è molto precisa: si osserva infatti che l'ordine è definito mediante un limite, quindi non implementabile in aritmetica finita. Una sua approssimazione finita è tanto più vicina al valore esatto quanto più grande è il numero di iterazioni  $k$ : come si vede dai risultati però, per raggiungere  $TOL$  è sufficiente in generale  $k \leq 4$ , ragion per cui le stime non sono molto precise. Inoltre si tenga conto che non conoscendo a priori lo zero  $\alpha$ , l'errore assoluto è comunque approssimato dalla distanza tra iterate successive. Gli stessi motivi sono validi per la stima del fattore di convergenza  $L = |g'''(\alpha)|/6$ .

Si osserva infine che i valori finali  $p = Inf$  ottenuti nel primo e nel quarto problema sono dovuti al fatto che la distanza tra  $x_k$  e  $x_{k-1}$  è inferiore alla precisione macchina per cui si ha il fenomeno della cancellazione e viene calcolata la quantità  $\log(0) = -Inf$  (come da *warning* di MATLAB). La cancellazione influisce anche sul calcolo del fattore  $L$  per quanto riguarda l'ultima iterazione: si ottengono infatti valori molto alti (o nulli nel caso  $p = Inf$  di cui sopra). ■

## Esercizio 2 : Interpolazione con funzioni splines

*Si vuole determinare la spline cubica interpolante il profilo superiore della Figura 1, che rappresenta Snoopy.*



Per avere i dati, dobbiamo riportare il profilo su di una griglia come indicato in Figura 2. Poichè ci sono dei punti in cui la curva non ha derivata continua, rappresentiamo il profilo con tre curve distinte e usiamo una spline cubica interpolante per ogni regione. I dati, più fitti dove la curva cambia più rapidamente, sono riportati in Tabella 1.

Curva1			Curva2			Curva3		
$k$	$x_k$	$y_k$	$k$	$x_k$	$y_k$	$k$	$x_k$	$y_k$
1	1	3.0	1	17	4.5	1	27.7	4.1
2	2	3.7	2	20	7.0	2	28	4.3
3	5	3.9	3	23	6.1	3	29	4.1
4	6	4.2	4	24	5.6	4	30	3.0
5	7	5.7	5	25	5.8			
6	8	6.6	6	27	5.2			
7	10	7.1	7	27.7	4.1			
8	13	6.7						
9	17	4.5						

Tabella 1: dati relativi al profilo di Figura 1

Scrivi un programma MATLAB per interpolare dei dati con la spline cubica **naturale**. In particolare, assegnati i vettori dei dati  $X$ ,  $Y$  ed il vettore  $XX$  delle nuove ascisse, la funzione MATLAB deve fornire il vettore  $YY$  dei valori della splines cubica naturale interpolante corrispondenti a  $XX$ . Usa tale programma per interpolare i dati di Tabella 1 e ricostruire graficamente il profilo di Snoopy.

**Soluzione.** Una possibile *function* che implementa il calcolo nelle nuove ascisse  $XX$  dei valori  $YY$  della spline cubica naturale interpolante i dati  $x$  e  $y$  è la seguente:

```
function YY=natspl(x,y,XX)

% calcola nelle ascisse XX i valori YY della
% spline cubica naturale interpolante i dati (x,y)

m=length(x);
h=x(2:m)-x(1:m-1);
H=2*diag(h(1:m-2)+h(2:m-1))+diag(h(2:m-2),1)+diag(h(2:m-2),-1);
Y=6*((y(3:m)-y(2:m-1))./h(2:m-1)-(y(2:m-1)-y(1:m-2))./h(1:m-2));
M=[0;H\Y;0];
alfa=(y(2:m)-y(1:m-1))./h(1:m-1)-(M(2:m)-M(1:m-1)).*h(1:m-1)/6;
beta=y(1:m-1)-M(1:m-1).*h.^2/6;
for i=1:m-1
    ind=find(x(i)<=XX&XX<=x(i+1));
    if ind==[]
        break
    end
    YY(ind)=M(i+1)*(XX(ind)-x(i)).^3/(6*h(i))-...
        M(i)*(XX(ind)-x(i+1)).^3/(6*h(i))+...
        alfa(i)*(XX(ind)-x(i))+beta(i);
end
plot(XX,YY)
```

In breve, *natspl* calcola la dimensione  $m$  del problema, il vettore dei passi  $h$  della suddivisione, costruisce la matrice tridiagonale  $H$  e il vettore dei termini noti  $Y$ , calcola il vettore dei momenti  $M$  risolvendo il sistema  $HM = Y$ , calcola i vettori delle costanti  $\alpha$  e  $\beta$  e valuta la spline in  $XX$ . Quest'ultima operazione viene fatta valutando di volta in volta, tramite l'istruzione *find*, in che sottointervallo  $[x_i, x_{i+1}]$  cadono le nuove ascisse  $XX$  e calcolando i valori  $YY$  utilizzando il polinomio di terzo grado relativo a quell'intervallo. L'istruzione *if* all'interno del ciclo *for* serve per evitare la valutazione del  $j$ -esimo polinomio se nessuna nuova ascissa cade in  $[x_j, x_{j+1}]$ .

Il seguente *script snoopy* raccoglie le istruzioni specifiche per la ricostruzione del profilo di snoopy secondo i dati forniti in Tabella 1.

```
% snoopy

x1=[1;2;5;6;7;8;10;13;17];
y1=[3;3.7;3.9;4.2;5.7;6.6;7.1;6.7;4.5];
XX1=linspace(x1(1),x1(length(x1)),1000);
x2=[17;20;23;24;25;27;27.7];
y2=[4.5;7;6.1;5.6;5.8;5.2;4.1];
XX2=linspace(x2(1),x2(length(x2)),1000);
```

```

x3=[27.7;28;29;30];
y3=[4.1;4.3;4.1;3];
XX3=linspace(x3(1),x3(length(x3)),1000);

```

```

hold on,grid on
YY1=natspl(x1,y1,XX1);
YY2=natspl(x2,y2,XX2);
YY3=natspl(x3,y3,XX3);

```

Il profilo ottenuto utilizzando 1000 punti per ciascuna delle tre regioni è rappresentato in Figura 1. ■

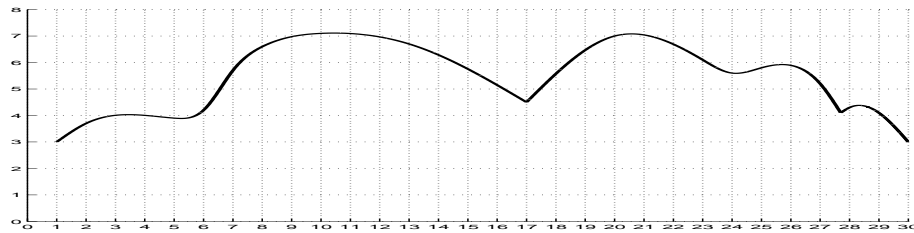


Figura 1: profilo di snoopy ricostruito mediante splines cubiche naturali.

### Esercizio 3 : Minimi quadrati

Assegnati i punti  $(x_i, y_i)$ ,  $i = 1, \dots, n$ , si vuole costruire la spline cubica  $s_3(x)$ ,  $x \in [x_0, x_n]$  approssimante nel senso dei minimi quadrati tali dati. Sia  $\Delta$  una suddivisione dell'intervallo  $[x_0, x_n]$  ottenuta con  $m \ll n$  punti  $t_j$ ,  $j = 1, \dots, m$ ,  $\Delta = t_1 < \dots < t_m$ , lo spazio delle splines cubiche relative a tale partizione ha dimensione  $m + 2$ . Volendo utilizzare le funzioni splines fornite da MATLAB con *SPLINE*, consideriamo quelle con condizioni al bordo del tipo not-a-knot. Con queste due ulteriori condizioni, lo spazio ha dimensione  $m$ . Una base per tale spazio può essere ottenuta considerando le splines not-a-knot  $\sigma_k(x)$ ,  $k = 1, \dots, m$  definite come i polinomi di Lagrange:  $\sigma_k(t_j) = 1$  se  $k = j$  e  $\sigma_k(t_j) = 0$  se  $k \neq j$ , per  $k, j = 1, \dots, m$ . La generica splines cubica not-a-knot  $s_3(x)$  si può scrivere come segue  $s_3(x) = \sum_{k=1}^m a_k \sigma_k(x)$  e vale  $s_3(t_j) = a_j$ ,  $j = 1, \dots, m$ . Scrivi un programma MATLAB che legge in ingresso i vettori  $X, Y$  dei dati, determina la spline cubica not-a-knot che approssima nel senso dei minimi quadrati tali dati, calcola i valori  $YY$  corrispondenti alle nuove ascisse  $XX$  e ne disegna il grafico. Confronta graficamente i risultati ottenuti con quelli forniti dal polinomio algebrico di terzo grado che approssima gli stessi dati nel senso dei minimi quadrati. Testa il programma generando i dati con **RANDN** e ordinandoli con **SORT** (vedi l'help in linea di MATLAB per queste funzioni) oppure utilizza i dati sulla popolazione statunitense tra il 1790 e il 1990 disponibili sul file *census.mat* di MATLAB (istruzione `>>load census`).

**Soluzione.** L'approssimazione nel senso dei minimi quadrati mediante le splines cubiche 'not-a-knot' di MATLAB può essere implementata con la seguente function:

```

function yy=minqspline(x,y,m,xx)

% approssima i dati (x,y) nel senso dei minimi quadrati
% con le spline 'not-a-knot' su m intervalli e restituisce
% i valori yy della spline nelle nuove ascisse xx

t=min(x)+(0:m-1)*(max(x)-min(x))/(m-1);
I=eye(m);
for j=1:m
    sigma=spline(t',I(:,j));
    A(:,j)=ppval(sigma,x);
    B(:,j)=ppval(sigma,xx');
end
a=A\y;
yy=B*a;

```

*minqspline* prende in input gli  $n$  dati  $(x_i, y_i)$ , il numero  $m$  dei punti della suddivisione  $\Delta = t_1 < \dots < t_m$ , il vettore delle nuove ascisse  $xx$  e restituisce il vettore  $yy$  dei valori della spline in  $xx$ . La spline viene costruita utilizzando la base  $\{\sigma_k\}_{k=1,\dots,m}$  dove  $\sigma_k$  è la spline cubica 'not-a-knot' che interpola sui nodi  $\Delta$  il  $k$ -esimo vettore della base canonica, ovvero la  $k$ -esima colonna della matrice identità di dimensione  $m$ . Si può ottenere agevolmente mediante l'istruzione *spline* di MATLAB. Mediante *ppval* si costruisce la matrice  $A \in \mathbb{R}^{m \times n}$  del sistema sovradeterminato  $Aa = y$  che ha come elementi  $A_{ij} = \sigma_j(x_i)$ ,  $i = 1, \dots, n$ ,  $j = 1, \dots, m$ . Il sistema viene quindi risolto con l'operatore ' $\backslash$ ' determinando il vettore  $a = (a_1, \dots, a_m)^T$  dei coefficienti della combinazione lineare delle  $\sigma_k$  che forma la spline  $s_3$  come specificato nel testo dell'esercizio. La valutazione di  $s_3$  nelle nuove ascisse  $xx_i$ ,  $i = 1, \dots, p$ , si ottiene in modo compatto costruendo una nuova matrice  $B \in \mathbb{R}^{m \times p}$  di elementi  $B_{ij} = \sigma_j(xx_i)$ ,  $i = 1, \dots, p$ ,  $j = 1, \dots, m$ , e calcolando il prodotto  $yy = Ba$ .

L'approssimazione nel senso dei minimi quadrati mediante il polinomio algebrico di grado  $q$  può essere implementata con la seguente *function*:

```

function yy=minqpol(x,y,q,xx)

% approssima i dati (x,y) nel senso dei minimi quadrati
% con un polinomio di grado q e restituisce i valori yy
% del polinomio nelle nuove ascisse xx

p=polyfit(x,y,q);
yy=polyval(p,xx);

```

*minqpol* ha la medesima sintassi di *minqspline* e calcola il polinomio di grado  $q$  che approssima gli  $n$  dati  $(x_i, y_i)$  mediante *polyfit* per valutarlo successivamente nelle nuove ascisse  $xx$  mediante *polyval*.

Lo script *minqdati* serve di ausilio per la creazione dei dati e il successivo test delle *function* di cui sopra:

```

% minqdati

n=100;
x=sort(randn(n,1));
y=sort(randn(n,1));
xx=linspace(min(x),max(x),1000);
m=7;
q=3;
yys=minqspline(x,y,m,xx);
yyp=minqppl(x,y,q,xx);

hold on,grid on
plot(x,y,'r.','markersize',10)
plot(xx,yyp,'-b')
plot(xx,yys,'-k')
xlabel('x')
ylabel('y')
legend('dati','polinomio di terzo grado',...
      ['spline cubica naturale,m=',num2str(m)])

```

In particolare, *minqdati* crea i dati  $x$  e  $y$  in modo random ordinato mediante *randn* e *sort*, crea le nuove ascisse  $xx$  suddividendo l'intervallo dei dati  $x$  con 1000 punti equidistanti, imposta  $m$  e  $q$  per le successive chiamate di *minqspline* e *minqppl* e infine provvede alla rappresentazione grafica dei risultati. La Figura 2 si riferisce ad  $m = 7$  e  $q = 3$ , mentre in Figura 3 sono rappresentate le spline base  $\sigma_k$  per  $k = 1, \dots, 7$ . La Figura 4 si riferisce ad  $m = 4$  e  $q = 3$ , caso in cui la spline e il polinomio di terzo grado coincidono. ■

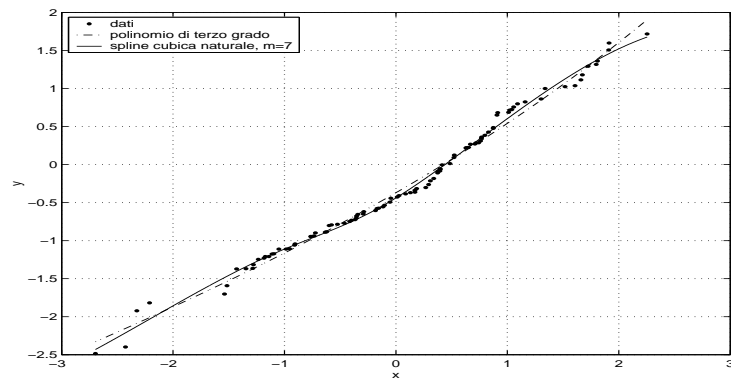


Figura 2: approssimazione ai minimi quadrati con polinomio di terzo grado e spline cubica naturale ( $m = 7$ ).

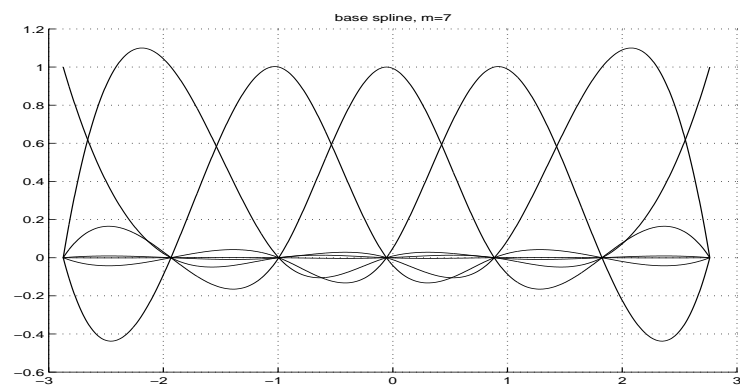


Figura 3: polinomi base della spline cubica naturale con  $m = 7$ .

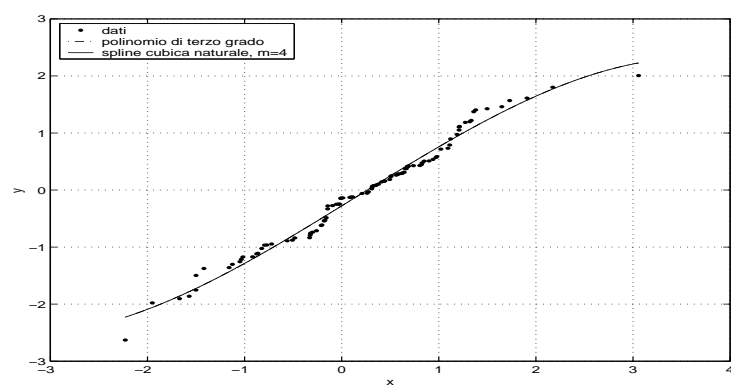


Figura 4: approssimazione ai minimi quadrati con polinomio di terzo grado e spline cubica naturale ( $m = 4$ ).