Elaborato Calcolo Numerico

Manetti Jacopo
Rosa Angelo

Appello di Luglio

Premessa

Nel seguente elaborato si è fatto utilizzo di function handler e anche di istruzioni simboliche. Nelle funzioni in cui è richiesta la tabulazione si è usato gli **fprintf** per ottenere una stampa a video immediata dei risultati nei vari cicli.

Esercizio 1

$\frac{3f(x)-4f(x-h)+f(x-2h)}{2h} = f'(x) + O(h^2)$

$3f(x) - 4f(x-h) + f(x-2h) = 2hf'(x) + O(h^3)$
$f(x) = f(x) + f'(x)(t-x) + \frac{f''(x)}{2!}(t-x)^2 + O(h^3)$
$f(x-h) = f(x) + f'(x)(\cancel{x} - h - \cancel{x}) + \frac{f''(x)}{2!}(\cancel{x} - h - \cancel{x})^2 + O(h^3)$
$= f(x) - f'(x)h + \frac{f''(x)}{2}h^2 + O(h^3)$

$f(x-2h) = f(x) + f'(x)(\cancel{x} - 2h - \cancel{x}) + \frac{f''(x)}{2!}(\cancel{x} - 2h - \cancel{x})^2 + O(h^3)$
$= f(x) - 2f'(x)h + \frac{4h^2 f''(x)}{2} + O(h^3) =$
$= f(x) - 2f'(x)h + 2h^2 f''(x) + O(h^3)$

avevamo

$-4f(x-h) + f(x-2h) = -3f(x) + 2hf'(x) + O(h^3)$
$-4(f(x) - f'(x)h + \frac{f''(x)}{2}h^2) + f(x) - 2f'(x)h + 2h^2 f''(x)$
$-4f(x) + 4f'(x)h - \cancel{2f''(x)h^2} + f(x) - 2f'(x)h + \cancel{2h^2 f''(x)}$
$-3f(x) + 2f'(x)h = -3f(x) + 2f'(x)h$

$\square$

Esercizio 2

$$u = \frac{1}{2} \cdot b^{1-m}$$

eps è il numero più piccolo che può intercorrere tra 2 numeri.La precisione di macchina è di 16 cifre decimali, $eps = 2.2204e^{-16}$. Nel caso delle IEEE 754 normalizzata con arrotondamento si ha il formato: $1.f$ con f cifra della mantissa. In Doppia precisione m=52, dunque possiamo dire che m=53 e b=2

$$u = \frac{1}{2} \cdot b^{1-m} = \frac{1}{2} \cdot 2^{-52} = 2^{-1} \cdot 2^{-52} = 2^{-53}$$

che equivale (da Matlab) a $1.1102e^{-16}$. La differenza è data dal bit del segno.

Esercizio 3

Il programma proposto porta a manifestare il fenomeno della cancellazione numerica. Questa si presenta quando abbiamo la cancellazione di cifre significative in quanto le operazioni di somma e di differenza non sono sempre ben condizionate, dovremmo ottenere **b** in realtà ma la rappresentazione finita porta con sé un errore che è già presente negli operandi e che viene poi amplificata dalle operazioni di somma e sottrazione.

Esercizio 4

```
function [valore] = Esercizio4(x)
% valore = Esercizio4(x) Metodo per calcolare il seno di un valore x
%                         con x compreso tra -pi e pi.
%                         Restituisce il valore calcolato
if x<-pi || x>pi
    error("valore non consentito");
end
valore = 0;
precedente = x;
j = 0;
while precedente ~= valore
    precedente = valore;
    f=1;
    temp = 2*j+1;
    for i=1:temp
        f=f*i;
    end
    valore = valore+((-1)^j*((x^((2*j)+1))/(f)));
    j = j+1;
end
return
```

Confronto con la funzione $sin(x)$ di Matlab:

```
>> Esercizio4(2)

ans =

    0.9093

>> sin(2)

ans =

    0.9093

>> Esercizio4(-1)

ans =

   -0.8415

>> sin(-1)

ans =

   -0.8415
```

I valori restituiti dalla funzione $sin(x)$ di Matlab sono uguali a quelli della function $Esercizio4$.

Esercizio 5-6

<u>Metodo di Bisezione</u>

```
function x = bisezionedef( a, b, f, tolx )
% x = Bisezione( a, b, f, tolx ) Metodo di bisezione per calcolare
%                                una radice di f(x), interna ad [a,b],
%                                con tolleranza tolx.
if a>=b, error('estremi intervallo errati'), end
if tolx<=0; error('tolleranza non appropriata'), end
fa = feval(f,a);
fb = feval(f,b);
xp=0;
if fa*fb>=0, error('intervallo di confidenza non appropriato'), end
imax = ceil( log2(b-a)-log2(tolx) );
if imax<1, x = (a+b)/2; return, end
fprintf('iteration number %d value %d \n',0,xp);
for i = 1:imax
      x = (a+b)/2;
      fprintf('iteration number %d value %d \n',i,x);
      fx = feval( f, x );
      if abs(x-xp)<=tolx*(1+abs(xp))

      break
      elseif fa*fx<0
        b = x;
      else
        a = x; fa = fx;
      end
      xp = x;
end
return
```

<u>Metodo di Newton</u>

```
function [x,itnumber] = newtondef( f, f1, x0, tolx, maxit )
%
% [x,flag] = newton( f, f1, x0, tolx [, maxit] )
%
% Metodo di Newton per determinare una approssimazione
% della radice di f(x)=0 con tolleranza (mista) tolx, a
% partire da x0, entro maxit iterationi (default = 100).
% f1 implementa f'(x) mentre in uscita flag vale -1, se
% la tolleranza non `e soddisfatta entro maxit iterate o
% la derivata si annulla, altrimenti ritorna il numero
% di iterazioni richieste.
```

```
if nargin<4, error('numero argomenti insufficienti')
elseif nargin==4, maxit = 100;
end
if tolx<eps, error('tolleranza non idonea'), end
x = x0;
itnumber = -1;
fprintf('iteration number %d value %d \n',0,x);
for i = 1:maxit
    fx = feval( f, x );
    f1x = feval( f1, x );
    if f1x==0, break, end
    x = x - fx/f1x;
    fprintf('iteration number %d value %d \n',i,x);
    if abs(x-x0)<=tolx*(1+abs(x0)), itnumber=i+1; break
    else, x0 = x;
    end
end
return
```

Metodo delle Secanti

```
function [niterazioni,x] = secantidef(f,x0,x1,imax,tolx)
fx0=f(x0);
fx1=f(x1);
x=(x0*fx1-x1*fx0)/(fx1-fx0);
fprintf('iteration number %d value %d \n',0,0);
fprintf('iteration number %d value %d \n',1,x);
niterazioni = 2;
for i=2:imax
    if abs(x-x1)<=tolx*(1+abs(x1))
        niterazioni = i;
        break
    end
        x0=x1; fx0=fx1;
        x1=x; fx1=f(x);
        x=(x0*fx1-x1*fx0)/(fx1-fx0);
        fprintf('iteration number %d value %d \n',i,x);
end
return
```

Metodo delle Corde

```
function [x,itnumber] = cordedef(f,tolx,x0,maxIt)
%n è il numero di iterazioni usate per convergere ad un numero
%convergenza è il numero che da quel momento in poi non cambia più se
%iterato ulteriormente.
%se non converge ritorna -1 in itnumber.
syms x;
f1 = matlabFunction(diff(f,x)); %fa la derivata di f
f1x = feval(f1,x0);
if f1x==0
    error("la derivata al denominatore non può essere 0");
end
x = x0;
itnumber = -1;
fprintf("derivata calcolata: %s\n",func2str(f1));
fprintf("valore derivata nel punto %d: %d\n",x0,f1x);
fprintf('iteration number %d value %d \n',0,x);
for i = 1:maxIt
    fx = feval(f,x);
    if f1x==0, break, end
    x = x - fx/f1x;
    fprintf('iteration number %d value %.4d \n',i,x);
     if abs(x-x0)<=tolx*(1+abs(x0)), itnumber=i+1; break

     end
    x0 = x;
end
return
```

Esercizio 7

```
f= @(x)x-cos(x)
f1= @(x)sin(x)+1
```

<u>Newton:</u>

```
tol: 10^-3
newtondef(f,f1,0,10^-3,100)
iteration number 0 value 0
iteration number 1 value 1
iteration number 2 value 7.503639e-01
iteration number 3 value 7.391129e-01
iteration number 4 value 7.390851e-01

ans =

     7.390851333852840e-01

tol: 10^-6
newtondef(f,f1,0,10^-6,100)
iteration number 0 value 0
iteration number 1 value 1
iteration number 2 value 7.503639e-01
iteration number 3 value 7.391129e-01
iteration number 4 value 7.390851e-01
iteration number 5 value 7.390851e-01

ans =

     7.390851332151607e-01

tol: 10^-9
newtondef(f,f1,0,10^-9,100)
iteration number 0 value 0
iteration number 1 value 1
iteration number 2 value 7.503639e-01
iteration number 3 value 7.391129e-01
iteration number 4 value 7.390851e-01
iteration number 5 value 7.390851e-01

ans =

     7.390851332151607e-01

tol: 10^-12
```

```
newtondef(f,f1,0,10^-12,100)
iteration number 0 value 0
iteration number 1 value 1
iteration number 2 value 7.503639e-01
iteration number 3 value 7.391129e-01
iteration number 4 value 7.390851e-01
iteration number 5 value 7.390851e-01
iteration number 6 value 7.390851e-01

ans =

     7.390851332151607e-01

%-----------------------------------------------------------------------%
```

Bisezione:

```
tol: 10^-3
bisezionedef(0,1,f,10^-3)
iteration number 0 value 0
iteration number 1 value 5.000000e-01
iteration number 2 value 7.500000e-01
iteration number 3 value 6.250000e-01
iteration number 4 value 6.875000e-01
iteration number 5 value 7.187500e-01
iteration number 6 value 7.343750e-01
iteration number 7 value 7.421875e-01
iteration number 8 value 7.382813e-01
iteration number 9 value 7.402344e-01
iteration number 10 value 7.392578e-01

ans =

     7.392578125000000e-01


tol: 10^-6
bisezionedef(0,1,f,10^-6)
iteration number 0 value 0
iteration number 1 value 5.000000e-01
iteration number 2 value 7.500000e-01
iteration number 3 value 6.250000e-01
iteration number 4 value 6.875000e-01
iteration number 5 value 7.187500e-01
iteration number 6 value 7.343750e-01
iteration number 7 value 7.421875e-01
```

```
iteration number 8 value 7.382813e-01
iteration number 9 value 7.402344e-01
iteration number 10 value 7.392578e-01
iteration number 11 value 7.387695e-01
iteration number 12 value 7.390137e-01
iteration number 13 value 7.391357e-01
iteration number 14 value 7.390747e-01
iteration number 15 value 7.391052e-01
iteration number 16 value 7.390900e-01
iteration number 17 value 7.390823e-01
iteration number 18 value 7.390862e-01
iteration number 19 value 7.390842e-01
iteration number 20 value 7.390852e-01

ans =

     7.390851974487305e-01


tol: 10^-9
bisezionedef(0,1,f,10^-9)
iteration number 0 value 0
iteration number 1 value 5.000000e-01
iteration number 2 value 7.500000e-01
iteration number 3 value 6.250000e-01
iteration number 4 value 6.875000e-01
iteration number 5 value 7.187500e-01
iteration number 6 value 7.343750e-01
iteration number 7 value 7.421875e-01
iteration number 8 value 7.382813e-01
iteration number 9 value 7.402344e-01
iteration number 10 value 7.392578e-01
iteration number 11 value 7.387695e-01
iteration number 12 value 7.390137e-01
iteration number 13 value 7.391357e-01
iteration number 14 value 7.390747e-01
iteration number 15 value 7.391052e-01
iteration number 16 value 7.390900e-01
iteration number 17 value 7.390823e-01
iteration number 18 value 7.390862e-01
iteration number 19 value 7.390842e-01
iteration number 20 value 7.390852e-01
iteration number 21 value 7.390847e-01
iteration number 22 value 7.390850e-01
iteration number 23 value 7.390851e-01
iteration number 24 value 7.390851e-01
```

```
iteration number 25 value 7.390851e-01
iteration number 26 value 7.390851e-01
iteration number 27 value 7.390851e-01
iteration number 28 value 7.390851e-01
iteration number 29 value 7.390851e-01
iteration number 30 value 7.390851e-01

ans =

     7.390851331874728e-01


tol: 10^-12
bisezionedef(0,1,f,10^-12)
iteration number 0 value 0
iteration number 1 value 5.000000e-01
iteration number 2 value 7.500000e-01
iteration number 3 value 6.250000e-01
iteration number 4 value 6.875000e-01
iteration number 5 value 7.187500e-01
iteration number 6 value 7.343750e-01
iteration number 7 value 7.421875e-01
iteration number 8 value 7.382813e-01
iteration number 9 value 7.402344e-01
iteration number 10 value 7.392578e-01
iteration number 11 value 7.387695e-01
iteration number 12 value 7.390137e-01
iteration number 13 value 7.391357e-01
iteration number 14 value 7.390747e-01
iteration number 15 value 7.391052e-01
iteration number 16 value 7.390900e-01
iteration number 17 value 7.390823e-01
iteration number 18 value 7.390862e-01
iteration number 19 value 7.390842e-01
iteration number 20 value 7.390852e-01
iteration number 21 value 7.390847e-01
iteration number 22 value 7.390850e-01
iteration number 23 value 7.390851e-01
iteration number 24 value 7.390851e-01
iteration number 25 value 7.390851e-01
iteration number 26 value 7.390851e-01
iteration number 27 value 7.390851e-01
iteration number 28 value 7.390851e-01
iteration number 29 value 7.390851e-01
iteration number 30 value 7.390851e-01
iteration number 31 value 7.390851e-01
```

```
iteration number 32 value 7.390851e-01
iteration number 33 value 7.390851e-01
iteration number 34 value 7.390851e-01
iteration number 35 value 7.390851e-01
iteration number 36 value 7.390851e-01
iteration number 37 value 7.390851e-01
iteration number 38 value 7.390851e-01
iteration number 39 value 7.390851e-01
iteration number 40 value 7.390851e-01

ans =

     7.390851332156672e-01


%----------------------------------------------------------------------%
```

Secanti:

```
tol: 10^-3
secantidef(f,0,1,100,10^-3)
iteration number 0 value 0
iteration number 1 value 6.850734e-01
iteration number 2 value 7.362990e-01
iteration number 3 value 7.391194e-01
iteration number 4 value 7.390851e-01

ans =

     5

tol: 10^-6
secantidef(f,0,1,100,10^-6)
iteration number 0 value 0
iteration number 1 value 6.850734e-01
iteration number 2 value 7.362990e-01
iteration number 3 value 7.391194e-01
iteration number 4 value 7.390851e-01
iteration number 5 value 7.390851e-01

ans =

     6

tol: 10^-9
secantidef(f,0,1,100,10^-9)
```

```
iteration number 0 value 0
iteration number 1 value 6.850734e-01
iteration number 2 value 7.362990e-01
iteration number 3 value 7.391194e-01
iteration number 4 value 7.390851e-01
iteration number 5 value 7.390851e-01
iteration number 6 value 7.390851e-01

ans =

     7

tol: 10^-12
secantidef(f,0,1,100,10^-12)
iteration number 0 value 0
iteration number 1 value 6.850734e-01
iteration number 2 value 7.362990e-01
iteration number 3 value 7.391194e-01
iteration number 4 value 7.390851e-01
iteration number 5 value 7.390851e-01
iteration number 6 value 7.390851e-01

ans =

     7

%-----------------------------------------------------------------------%
```

Corde:

```
tol: 10^-3
cordedef (f,10^-3,0,100)
derivata calcolata: @(x)sin(x)+1.0
valore derivata nel punto 0: 1
iteration number 0 value 0
iteration number 1 value 0001
iteration number 2 value 5.4030e-01
iteration number 3 value 8.5755e-01
iteration number 4 value 6.5429e-01
iteration number 5 value 7.9348e-01
iteration number 6 value 7.0137e-01
iteration number 7 value 7.6396e-01
iteration number 8 value 7.2210e-01
iteration number 9 value 7.5042e-01
iteration number 10 value 7.3140e-01
iteration number 11 value 7.4424e-01
```

```
iteration number 12 value 7.3560e-01
iteration number 13 value 7.4143e-01
iteration number 14 value 7.3751e-01
iteration number 15 value 7.4015e-01
iteration number 16 value 7.3837e-01
iteration number 17 value 7.3957e-01

ans =

     7.395672022122561e-01

tol: 10^-6
cordedef (f,10^-6,0,100)
derivata calcolata: @(x)sin(x)+1.0
valore derivata nel punto 0: 1
iteration number 0 value 0
iteration number 1 value 0001
iteration number 2 value 5.4030e-01
iteration number 3 value 8.5755e-01
iteration number 4 value 6.5429e-01
iteration number 5 value 7.9348e-01
iteration number 6 value 7.0137e-01
iteration number 7 value 7.6396e-01
iteration number 8 value 7.2210e-01
iteration number 9 value 7.5042e-01
iteration number 10 value 7.3140e-01
iteration number 11 value 7.4424e-01
iteration number 12 value 7.3560e-01
iteration number 13 value 7.4143e-01
iteration number 14 value 7.3751e-01
iteration number 15 value 7.4015e-01
iteration number 16 value 7.3837e-01
iteration number 17 value 7.3957e-01
iteration number 18 value 7.3876e-01
iteration number 19 value 7.3930e-01
iteration number 20 value 7.3894e-01
iteration number 21 value 7.3918e-01
iteration number 22 value 7.3902e-01
iteration number 23 value 7.3913e-01
iteration number 24 value 7.3905e-01
iteration number 25 value 7.3911e-01
iteration number 26 value 7.3907e-01
iteration number 27 value 7.3909e-01
iteration number 28 value 7.3908e-01
iteration number 29 value 7.3909e-01
iteration number 30 value 7.3908e-01
```

```
iteration number 31 value 7.3909e-01
iteration number 32 value 7.3908e-01
iteration number 33 value 7.3909e-01
iteration number 34 value 7.3908e-01

ans =

     7.390845495752126e-01


tol: 10^-9
cordedef (f,10^-9,0,100)
derivata calcolata: @(x)sin(x)+1.0
valore derivata nel punto 0: 1
iteration number 0 value 0
iteration number 1 value 0001
iteration number 2 value 5.4030e-01
iteration number 3 value 8.5755e-01
iteration number 4 value 6.5429e-01
iteration number 5 value 7.9348e-01
iteration number 6 value 7.0137e-01
iteration number 7 value 7.6396e-01
iteration number 8 value 7.2210e-01
iteration number 9 value 7.5042e-01
iteration number 10 value 7.3140e-01
iteration number 11 value 7.4424e-01
iteration number 12 value 7.3560e-01
iteration number 13 value 7.4143e-01
iteration number 14 value 7.3751e-01
iteration number 15 value 7.4015e-01
iteration number 16 value 7.3837e-01
iteration number 17 value 7.3957e-01
iteration number 18 value 7.3876e-01
iteration number 19 value 7.3930e-01
iteration number 20 value 7.3894e-01
iteration number 21 value 7.3918e-01
iteration number 22 value 7.3902e-01
iteration number 23 value 7.3913e-01
iteration number 24 value 7.3905e-01
iteration number 25 value 7.3911e-01
iteration number 26 value 7.3907e-01
iteration number 27 value 7.3909e-01
iteration number 28 value 7.3908e-01
iteration number 29 value 7.3909e-01
iteration number 30 value 7.3908e-01
iteration number 31 value 7.3909e-01
```

```
iteration number 32 value 7.3908e-01
iteration number 33 value 7.3909e-01
iteration number 34 value 7.3908e-01
iteration number 35 value 7.3909e-01
iteration number 36 value 7.3908e-01
iteration number 37 value 7.3909e-01
iteration number 38 value 7.3909e-01
iteration number 39 value 7.3909e-01
iteration number 40 value 7.3909e-01
iteration number 41 value 7.3909e-01
iteration number 42 value 7.3909e-01
iteration number 43 value 7.3909e-01
iteration number 44 value 7.3909e-01
iteration number 45 value 7.3909e-01
iteration number 46 value 7.3909e-01
iteration number 47 value 7.3909e-01
iteration number 48 value 7.3909e-01
iteration number 49 value 7.3909e-01
iteration number 50 value 7.3909e-01
iteration number 51 value 7.3909e-01
iteration number 52 value 7.3909e-01

ans =

     7.390851327392538e-01


tol: 10^-12
cordedef (f,10^-12,0,100)
derivata calcolata: @(x)sin(x)+1.0
valore derivata nel punto 0: 1
iteration number 0 value 0
iteration number 1 value 0001
iteration number 2 value 5.4030e-01
iteration number 3 value 8.5755e-01
iteration number 4 value 6.5429e-01
iteration number 5 value 7.9348e-01
iteration number 6 value 7.0137e-01
iteration number 7 value 7.6396e-01
iteration number 8 value 7.2210e-01
iteration number 9 value 7.5042e-01
iteration number 10 value 7.3140e-01
iteration number 11 value 7.4424e-01
iteration number 12 value 7.3560e-01
iteration number 13 value 7.4143e-01
iteration number 14 value 7.3751e-01
```

```
iteration number 15 value 7.4015e-01
iteration number 16 value 7.3837e-01
iteration number 17 value 7.3957e-01
iteration number 18 value 7.3876e-01
iteration number 19 value 7.3930e-01
iteration number 20 value 7.3894e-01
iteration number 21 value 7.3918e-01
iteration number 22 value 7.3902e-01
iteration number 23 value 7.3913e-01
iteration number 24 value 7.3905e-01
iteration number 25 value 7.3911e-01
iteration number 26 value 7.3907e-01
iteration number 27 value 7.3909e-01
iteration number 28 value 7.3908e-01
iteration number 29 value 7.3909e-01
iteration number 30 value 7.3908e-01
iteration number 31 value 7.3909e-01
iteration number 32 value 7.3908e-01
iteration number 33 value 7.3909e-01
iteration number 34 value 7.3908e-01
iteration number 35 value 7.3909e-01
iteration number 36 value 7.3908e-01
iteration number 37 value 7.3909e-01
iteration number 38 value 7.3909e-01
iteration number 39 value 7.3909e-01
iteration number 40 value 7.3909e-01
iteration number 41 value 7.3909e-01
iteration number 42 value 7.3909e-01
iteration number 43 value 7.3909e-01
iteration number 44 value 7.3909e-01
iteration number 45 value 7.3909e-01
iteration number 46 value 7.3909e-01
iteration number 47 value 7.3909e-01
iteration number 48 value 7.3909e-01
iteration number 49 value 7.3909e-01
iteration number 50 value 7.3909e-01
iteration number 51 value 7.3909e-01
iteration number 52 value 7.3909e-01
iteration number 53 value 7.3909e-01
iteration number 54 value 7.3909e-01
iteration number 55 value 7.3909e-01
iteration number 56 value 7.3909e-01
iteration number 57 value 7.3909e-01
iteration number 58 value 7.3909e-01
iteration number 59 value 7.3909e-01
iteration number 60 value 7.3909e-01
```

```
iteration number 61 value 7.3909e-01
iteration number 62 value 7.3909e-01
iteration number 63 value 7.3909e-01
iteration number 64 value 7.3909e-01
iteration number 65 value 7.3909e-01
iteration number 66 value 7.3909e-01
iteration number 67 value 7.3909e-01
iteration number 68 value 7.3909e-01
iteration number 69 value 7.3909e-01

ans =

     7.390851332157368e-01
```

Commento sui risultati: Come si può notare dalle iterazioni il metodo di newton e delle secanti sono quelli che convergono più velocemente e le iterazioni non aumentano molto anche al raddoppiare della tolleranza. Il metodo di bisezione e corde invece raddoppia il numero di iterazioni al raddoppiare della tolleranza.

Esercizio 8

function per molteplicità e metodi:

```
FUNCTION MOLTEPLICITA':

function [molteplicita,derivata] = molteplicita(f,punto)
syms x
fx = f(punto); %derivata in 0 non deve fare 0
molteplicita = 0;
derivata = f;
fprintf("non derivata: %s valore: %d\n",func2str(f),fx);
while (fx == 0)
    str = diff(f,x);
    f1 = matlabFunction(str);
    fx = f1(punto); %derivata nel punto non deve fare 0
    molteplicita = molteplicita+1;
    fprintf("derivata: %s ; molteplicita: %d; valore in %d: %d\n",str,molteplicita,punto,fx
    derivata = str;
    if (fx ~= 0)
        break
    end
    f = f1;
end
return


FUNCTION AITKEN:

function [x,passi] = Aitken(f,df,x0,epsilon,upper)

f0=feval(f,x0);
d=feval(df,x0);
x1=x0-(f0/d);
f1=feval(f,x1);

d=feval(df,x1);
x2=x1-(f1/d);
f2=feval(f,x2);

count=0;
fprintf('iteration number %d value %d \n',count,x2);

for i=1:upper
   count=count+1;
   x0=(x1*x1-x0*x2)/(2*x1-x2-x0);
```

```
    f0=feval(f,x0);
    d=feval(df,x0);
    if d==0
        passi=count+1;
        break
    end
    x1=x0-(f0/d);
    f1=feval(f,x1);
    d=feval(df,x1);
    if d == 0
        passi=count+1;
        break;
    end
    x2=x1-(f1/d);
    f2=feval(f,x2);
    fprintf('iteration number %d value %d \n',count,x2);
    if abs(x2-x1)<=epsilon*(1+abs(x1))
        passi=count+1;
        break;
    end
end
x=x2;
passi=count+2;
return
```

FUNCTION NEWTON MODIFICATO

```
function [x,itnumber] = newtonmod( f, f1, x0, tolx, maxit,m )
%
% [x,flag] = newtonmod( f, f1, x0, tolx , maxit,m )
%                        Input: f-> funzione su cui applicare il metodo
%                               f1 -> derivata della funzione f
%                               x0 -> punto
%                               tolx -> tolleranza
%                               maxit -> numero massimo di iterazioni
%                               m -> molteplicità
%                       Output: x -> valore di x trovato
%                               itnumber -> numero dell'iterazione
if nargin<4, error('numero argomenti insufficienti')
elseif nargin==4, maxit = 100;
end
if tolx<eps, error('tolleranza non idonea'), end
x = x0;
itnumber = -1;
fprintf('iteration number %d value %d \n',0,x);
for i = 1:maxit
```

```
    fx = feval( f, x );
    f1x = feval( f1, x );
    if f1x==0, break, end
    x = x - m*(fx/f1x);
    fprintf('iteration number %d value %d \n',i,x);
    if abs(x-x0)<=tolx*(1+abs(x0)), itnumber=i+1; break
    else, x0 = x;
    end
end
return
```

la function *newtondef* si trova nell' es. 5-6.

Calcolo della molteplicità:

```
[m,d] = molteplicita(f,1)
non derivata: @(x)(x-1)^3*exp(x-1) valore: 0
derivata: 3*exp(x-1)*(x-1)^2+exp(x-1)*(x-1)^3 ;
molteplicita: 1; valore in 0: 0

derivata: 3*exp(x-1)*(2*x-2)+6*exp(x-1)*(x-1)^2+exp(x-1)*(x-1)^3 ;
molteplicita: 2; valore in 0: 0

derivata: 6*exp(x-1)+9*exp(x-1)*(2*x-2)+9*exp(x-1)*(x-1)^2+exp(x-1)*(x-1)^3 ;
molteplicita: 3; valore in 0: 6

m = 3

d = 6*exp(x-1)+9*exp(x-1)*(2*x-2)+9*exp(x-1)*(x-1)^2+exp(x-1)*(x-1)^3
```

Metodi di approssimazione:

```
NEWTON MODIFICATO con la molteplicità:

tol = 10^-3

>> newtonmod(f,f1,0,10^-3,100,3)
iteration number 0 value 0
iteration number 1 value 1.500000e+00
iteration number 2 value 1.071429e+00
iteration number 3 value 1.001661e+00
iteration number 4 value 1.000001e+00

ans =

   1.000000919274803
```

```
tol: 10^-6
>> newtonmod(f,f1,0,10^-6,100,3)
iteration number 0 value 0
iteration number 1 value 1.500000e+00
iteration number 2 value 1.071429e+00
iteration number 3 value 1.001661e+00
iteration number 4 value 1.000001e+00
iteration number 5 value 1.000000e+00

ans =

   1.000000000000282

tol: 10^-9
>> newtonmod(f,f1,0,10^-9,100,3)
iteration number 0 value 0
iteration number 1 value 1.500000e+00
iteration number 2 value 1.071429e+00
iteration number 3 value 1.001661e+00
iteration number 4 value 1.000001e+00
iteration number 5 value 1.000000e+00
iteration number 6 value 1

ans =

     1
tol: 10^-12

>> newtonmod(f,f1,0,10^-12,100,3)
iteration number 0 value 0
iteration number 1 value 1.500000e+00
iteration number 2 value 1.071429e+00
iteration number 3 value 1.001661e+00
iteration number 4 value 1.000001e+00
iteration number 5 value 1.000000e+00
iteration number 6 value 1

ans =

     1
```

NEWTON NORMALE:

tol: 10^-3

```
>> newtondef (f,f1,0,10^-3,100)
iteration number 0 value 0
iteration number 1 value 5.000000e-01
iteration number 2 value 7.000000e-01
iteration number 3 value 8.111111e-01
iteration number 4 value 8.783048e-01
iteration number 5 value 9.205850e-01
iteration number 6 value 9.477764e-01
iteration number 7 value 9.654927e-01
iteration number 8 value 9.771290e-01
iteration number 9 value 9.848112e-01
iteration number 10 value 9.898999e-01
iteration number 11 value 9.932780e-01
iteration number 12 value 9.955237e-01
iteration number 13 value 9.970180e-01

ans =

   0.997018019368188
```

tol: 10^-6

```
>> newtondef (f,f1,0,10^-6,100)
iteration number 0 value 0
iteration number 1 value 5.000000e-01
iteration number 2 value 7.000000e-01
iteration number 3 value 8.111111e-01
iteration number 4 value 8.783048e-01
iteration number 5 value 9.205850e-01
iteration number 6 value 9.477764e-01
iteration number 7 value 9.654927e-01
iteration number 8 value 9.771290e-01
iteration number 9 value 9.848112e-01
iteration number 10 value 9.898999e-01
iteration number 11 value 9.932780e-01
iteration number 12 value 9.955237e-01
iteration number 13 value 9.970180e-01
iteration number 14 value 9.980130e-01
iteration number 15 value 9.986758e-01
iteration number 16 value 9.991174e-01
iteration number 17 value 9.994117e-01
iteration number 18 value 9.996078e-01
iteration number 19 value 9.997386e-01
```

```
iteration number 20 value 9.998257e-01
iteration number 21 value 9.998838e-01
iteration number 22 value 9.999225e-01
iteration number 23 value 9.999484e-01
iteration number 24 value 9.999656e-01
iteration number 25 value 9.999771e-01
iteration number 26 value 9.999847e-01
iteration number 27 value 9.999898e-01
iteration number 28 value 9.999932e-01
iteration number 29 value 9.999955e-01
iteration number 30 value 9.999970e-01

ans =

   0.999996977916602

tol: 10^-9
>> newtondef (f,f1,0,10^-9,100)
iteration number 0 value 0
iteration number 1 value 5.000000e-01
iteration number 2 value 7.000000e-01
iteration number 3 value 8.111111e-01
iteration number 4 value 8.783048e-01
iteration number 5 value 9.205850e-01
iteration number 6 value 9.477764e-01
iteration number 7 value 9.654927e-01
iteration number 8 value 9.771290e-01
iteration number 9 value 9.848112e-01
iteration number 10 value 9.898999e-01
iteration number 11 value 9.932780e-01
iteration number 12 value 9.955237e-01
iteration number 13 value 9.970180e-01
iteration number 14 value 9.980130e-01
iteration number 15 value 9.986758e-01
iteration number 16 value 9.991174e-01
iteration number 17 value 9.994117e-01
iteration number 18 value 9.996078e-01
iteration number 19 value 9.997386e-01
iteration number 20 value 9.998257e-01
iteration number 21 value 9.998838e-01
iteration number 22 value 9.999225e-01
iteration number 23 value 9.999484e-01
iteration number 24 value 9.999656e-01
iteration number 25 value 9.999771e-01
iteration number 26 value 9.999847e-01
iteration number 27 value 9.999898e-01
```

```
iteration number 28 value 9.999932e-01
iteration number 29 value 9.999955e-01
iteration number 30 value 9.999970e-01
iteration number 31 value 9.999980e-01
iteration number 32 value 9.999987e-01
iteration number 33 value 9.999991e-01
iteration number 34 value 9.999994e-01
iteration number 35 value 9.999996e-01
iteration number 36 value 9.999997e-01
iteration number 37 value 9.999998e-01
iteration number 38 value 9.999999e-01
iteration number 39 value 9.999999e-01
iteration number 40 value 9.999999e-01
iteration number 41 value 1.000000e+00
iteration number 42 value 1.000000e+00
iteration number 43 value 1.000000e+00
iteration number 44 value 1.000000e+00
iteration number 45 value 1.000000e+00
iteration number 46 value 1.000000e+00
iteration number 47 value 1.000000e+00

ans =

   0.999999996932713

tol: 10^-12
>> newtondef (f,f1,0,10^-12,100)
iteration number 0 value 0
iteration number 1 value 5.000000e-01
iteration number 2 value 7.000000e-01
iteration number 3 value 8.111111e-01
iteration number 4 value 8.783048e-01
iteration number 5 value 9.205850e-01
iteration number 6 value 9.477764e-01
iteration number 7 value 9.654927e-01
iteration number 8 value 9.771290e-01
iteration number 9 value 9.848112e-01
iteration number 10 value 9.898999e-01
iteration number 11 value 9.932780e-01
iteration number 12 value 9.955237e-01
iteration number 13 value 9.970180e-01
iteration number 14 value 9.980130e-01
iteration number 15 value 9.986758e-01
iteration number 16 value 9.991174e-01
iteration number 17 value 9.994117e-01
iteration number 18 value 9.996078e-01
```

```
iteration number 19 value 9.997386e-01
iteration number 20 value 9.998257e-01
iteration number 21 value 9.998838e-01
iteration number 22 value 9.999225e-01
iteration number 23 value 9.999484e-01
iteration number 24 value 9.999656e-01
iteration number 25 value 9.999771e-01
iteration number 26 value 9.999847e-01
iteration number 27 value 9.999898e-01
iteration number 28 value 9.999932e-01
iteration number 29 value 9.999955e-01
iteration number 30 value 9.999970e-01
iteration number 31 value 9.999980e-01
iteration number 32 value 9.999987e-01
iteration number 33 value 9.999991e-01
iteration number 34 value 9.999994e-01
iteration number 35 value 9.999996e-01
iteration number 36 value 9.999997e-01
iteration number 37 value 9.999998e-01
iteration number 38 value 9.999999e-01
iteration number 39 value 9.999999e-01
iteration number 40 value 9.999999e-01
iteration number 41 value 1.000000e+00
iteration number 42 value 1.000000e+00
iteration number 43 value 1.000000e+00
iteration number 44 value 1.000000e+00
iteration number 45 value 1.000000e+00
iteration number 46 value 1.000000e+00
iteration number 47 value 1.000000e+00
iteration number 48 value 1.000000e+00
iteration number 49 value 1.000000e+00
iteration number 50 value 1.000000e+00
iteration number 51 value 1.000000e+00
iteration number 52 value 1.000000e+00
iteration number 53 value 1.000000e+00
iteration number 54 value 1.000000e+00
iteration number 55 value 1.000000e+00
iteration number 56 value 1.000000e+00
iteration number 57 value 1.000000e+00
iteration number 58 value 1.000000e+00
iteration number 59 value 1.000000e+00
iteration number 60 value 1.000000e+00
iteration number 61 value 1.000000e+00
iteration number 62 value 1.000000e+00
iteration number 63 value 1.000000e+00
iteration number 64 value 1.000000e+00
```

```
ans =

   0.999999999996887

AITKEN

tol: 10^-3
>> [numero,iterazioni] = aitken(f,f1,0,10^-3,100)
iteration number 0 value 7.000000e-01
iteration number 1 value 9.294450e-01
iteration number 2 value 9.973416e-01

numero =

   0.997341642162314


iterazioni =

     4

tol: 10^-6
>> [numero,iterazioni] = aitken(f,f1,0,10^-6,100)
iteration number 0 value 7.000000e-01
iteration number 1 value 9.294450e-01
iteration number 2 value 9.973416e-01
iteration number 3 value 9.999965e-01

numero =

   0.999996458320337


iterazioni =

     5

tol: 10^-9
>> [numero,iterazioni] = aitken(f,f1,0,10^-9,100)
iteration number 0 value 7.000000e-01
iteration number 1 value 9.294450e-01
iteration number 2 value 9.973416e-01
iteration number 3 value 9.999965e-01
iteration number 4 value 1.000000e+00
```

```
numero =

   0.999999999944272

iterazioni =

     6

tol: 10^-12
con tale tolleranza il metodo di aitken non converge.
```

Commento sui risultati: il metodo migliore risulta newton modificato, aitken è molto simile ma con tolleranze troppo piccole non converge, il metodo di newton invece aumenta di molto le iterazione al raddoppiare della tolleranza.

Esercizio 9

```
function [LU,p] = plu(A)
% [LU,p] = plu(A) [INPUT]
%                 A => Matrice quadrata non singolare tale che A è fattorizzabile LU.
%                 [OUTPUT]
%                 LU => matrice che contiene sia le informazioni della triangolare
%                       inferiore che della triangolare superiore
%                 p => vettore che contiene l'informazione della matrice di
%                      permutazione P.
%
%                 Restituisce errore se viene fornita una matrice non quadrata e/o non
%                 singolare.
[m,n] = size(A);
if m~=n, error('matrice non quadrata'); end
p = (1:n);
LU = A;
for i=1:n-1
    [mi,ki] = max(abs(LU(i:n,i)));
    ki = ki+i-1;
    if mi==0
        error("matrice non singolare");
    end
    if ki>i
        LU([i,ki],:) = LU([ki,i],:);
        p([i,ki]) = p([ki,i]);
    end
    LU(i+1:n,i) = LU(i+1:n,i)/LU(i,i);
    LU(i+1:n,i+1:n) = LU(i+1:n,i+1:n)-LU(i+1:n,i)*LU(i,i+1:n);
end
end
```

Esercizio 10

```
function x = mialu(LU,p,b)
%  x = mialu(LU,p,b) [INPUT]
%                    LU => input Matrice fattorizzata LU tramite il codice plu
%                    p => contiene l'informazione della matrice
%                         di permutazione P, ottenuta da plu.m
%                    b => vettori termini noti sistema lineare
%                    [OUTPUT]
%                    x => vettore contenente le soluzioni del sistema A*x=b
%                    Il codice si propone di risolve il sistema di equazioni lineari
%                    LU*x=b dove LU = A ovvero A*x=b
%                    restituisce un errore se la matrice fornita in input
%                    è non quadrata e/o la dimensione della matrice
%                    non coincide con la dimensione dei vettori:
%                    permutazione e soluzione.
[m,n] = size(LU);
if m~=n, error('matrice non quadrata'); end
if length(p) ~= n || length(b) ~= n, error('dimensioni errate'); end
x = b(p);
for i=1:n-1
    x(i+1:n) = x(i+1:n) - LU(i+1:n,i)*x(i);
end
x(n) = x(n)/LU(n,n);
for i=n-1:-1:1
    x(1:i) = x(1:i)-LU(1:i,i+1)*x(i+1);
    x(i) = x(i)/LU(i,i);
end
return
end
```

Esempi per validare il codice:

```
    Esempio1:
A =
     3    -2     1
     1     1     1
     1    -1     1

>> b = [1;1;1];
>> [LU,p] = plu(A)

LU =

   3.000000000000000  -2.000000000000000   1.000000000000000
   0.333333333333333   1.666666666666667   0.666666666666667
   0.333333333333333  -0.200000000000000   0.800000000000000
```

```
p =

     1     2     3

>> [x] = mialu(LU,p,b)

x =

     0
     0
     1
```

Esempio2:

```
>> format rational
>> A = [1 1 1; 3 -1 -1; 4 1 -2]

A =

       1              1              1
       3             -1             -1
       4              1             -2

>> b = [2;2;1];
>> [LU,p] = plu(A)

LU =

       4              1             -2
       3/4           -7/4            1/2
       1/4           -3/7           12/7


p =

       3              2              1

>> [x] = mialu(LU,p,b)

x =

       1
      -1/3
       4/3
```

La controverifica degli esercizi è stata fatta mediante il codice:

```
A = MATRICE;
b = VETTORE TERMINI NOTI;
x = A\b;
```

ovvero:

```
>> A

A =

       3             -2              1
       1              1              1
       1             -1              1

>> b

b =

       1
       1
       1

>> x

x =

       0
       0
       1
```

e nel secondo:

```
A =

       1              1              1
       3             -1             -1
       4              1             -2

>> b

b =

       2
       2
       1
```

```
>> x

x =

      1
     -1/3
      4/3
```

Esercizio 11

```
function [x] = mialdl(A,b)
%   [INPUT]
%            A => matrice sdp fornita in input.Nel codice mano mano si verifica
%                 anche che la matrice sia effettivamente sdp o meno.
%            b => vettore delle soluzioni
%
%   La funzione ritorna nel vettore x la soluzioni del sistema
%   A*x=b ovvero LDL^T*x=b
[m,n] = size(A);
if m~=n, error('matrice non quadrata'); end
if length(b) ~= n, error('dimensioni errate'); end
if A(1,1)<=0
    error("la matrice fornita non è sdp");
end
A(2:n,1) = A(2:n,1)/A(1,1);
for j=2:n
    v = (A(j,1:j-1).').*diag(A(1:j-1,1:j-1));
    A(j,j) = A(j,j)-A(j,1:j-1)*v;
    if A(j,j)<=0
        error("la matrice non è sdp");
    end
    A(j+1:n,j) = (A(j+1:n,j)-A(j+1:n,1:j-1)*v)/A(j,j);
end
%disp(A);
n=length(b);
y=b;
for i=2:n
    for j=1:i-1
        y(i)=y(i)-A(i,j)*y(j);
    end
end
z=y ./ diag(A);
x=z;
```

```
for i=n:-1:1
    for j=1:i-1
        x(j)=x(j)-A(i,j)*x(i);
    end
end
end
```

Esempi per validare il codice:

```
    A =

       1              1              1
       1              3              3
       1              3              6

>> b = [2;2;1]

>> [x] = mialdl(A,b)

x =

       2
       1/3
      -1/3

infatti si ha che

>> x = A\b

x =

       2
       1/3
      -1/3

ESEMPIO 2:

A =

       4              1              0
       1              1              0
       0              0              8

>> b = [3;4;5]
```

```
b =

       3
       4
       5

>> [x] = mialdl(A,b)

x =

      -1/3
      13/3
       5/8

Infatti:
>> x = A\b

x =

      -1/3
      13/3
       5/8
```

Esercizio 12

Dimostrazione che $A^T A$ è sdp data A nonsingolare:
Calcolando $A^T A$ otteniamo

```
A^TA =

    0.6225   -0.1974   -2.5862
   -0.1974    1.7890  -11.4951
   -2.5862  -11.4951   98.5883
```

che è una matrice simmetrica.
Ora mostriamo che è definita positiva, ossia mostriamo che:

$$\forall x \in \mathbb{R}, x \neq 0 : x^T A^T A x > 0$$

infatti

$$\left(x^T A^T\right)(Ax) = b^T b = \sum_{i=0}^{n} b_i^2 > 0, b \neq 0$$

quindi la matrice è sdp.

Risoluzione dei sistemi lineari dati:

```
A =

    0.5635   -0.4651   -0.2978
   -0.4651    0.5746   -1.1147
   -0.2978   -1.1147    9.8619

>>b = [-1.2601;-2.6600;27.0585]

b =

   -1.2601
   -2.6600
   27.0585

>> T=A'*A

ans =

    0.6225   -0.1974   -2.5862
   -0.1974    1.7890  -11.4951
   -2.5862  -11.4951   98.5883

>> A'*A

ans =

     221/355       -616/3121     -2969/1148
    -616/3121      2383/1332     -1161/101
   -2969/1148     -1161/101       1676/17

Calcolo Ax=b dalla prima richiesta
>> [LU,p] = plu(A)

LU =

   0.563500000000000  -0.465100000000000  -0.297800000000000
  -0.528482697426797  -1.360497302573203   9.704517852706299
  -0.825377107364685  -0.140181907750915  -0.000099476178025


p =

     1     3     2
```

```
>> [x] = mialu(LU,p,b)

x =

   0.999999999985679
   1.999999999984078
   2.999999999997768

>> cond(A)

ans =

     1.311278960740176e+06
```

Seconda richiesta: (A'*A)x=(A'*b)

```
>> T2 = A'*b

T2 =

   -7.5309
  -31.1045
  270.1886

>> [x] = mialdl(T,T2)

x =

   1.000050547311859
   2.000056197785106
   3.000007878461599

>> cond(T)

ans =

     1.719439227118773e+12
>> cond(T2)

ans =

     1
```

I numeri di condizionamento ottenuti indicano che sia la matrice A usata per risolvere il sistema 1) che la matrice A'*A (usata per il 2)) sono malcondizionate in quanto $K(a) >>> 1$ e $K(A' * A) >>> 1$. Questo porta ad avere un risultato condizionato da errore ma il sistema 2) in particolare, ha un condizionamento sull'errore molto maggiore rispetto al sistema 1) (più nello specifico ha un ordine di differenza quadratico). Tale errore è "ingigantito" perchè viene effettuato un prodotto matriciale che è già di per se' affetto da errore a sua volta amplificato da un calcolo riprodotto su memoria finita.

Esercizio 13

```
function [QR] = qrfat(A)
% [QR] = qrfat(A) Metodo per ottenere la fattorizzazione QR
%                   di una matrice A avente come caratteristica
%                   m>=n, con m numero di righe e n numero di colonne.
%                   Restituisce error se non è vero che m>=n oppure
%                   durante la computazione viene verificato che la
%                   matrice non abbia rango massimo.
[m,n] = size(A);
if m<n, error('matrice non valida, m>=n necessario'); end
for i=1:n
   alfa=norm(A(i:m,i));
   if alfa==0
       error("la matrice non ha rango max");
   end
   if A(i,i)>=0
       alfa = -alfa;
   end
   v1 = A(i,i)-alfa;
   A(i,i) = alfa;
   A(i+1:m,i) = A(i+1:m,i)/v1;
   beta = -v1/alfa;
   A(i:m,i+1:n)=A(i:m,i+1:n)-(beta*[1;A(i+1:m,i)])*([1 A(i+1:m,i)']*A(i:m,i+1:n));
end
QR=A;
end
```

Esercizio 14

```
function [x] = miaqr(A,b)
% [x,r] = miaqr(A,b) Metodo per risolvere il sistema Ax=b
%                    dove A è una matrice di tipo QR ottenuta, ad esempio,
%                    dalla function qrfat.m
%                    Condizione necessaria: m>=n, con m numero di righe e
%                    n numero di colonne.
%                    Restituisce error se la matrice QR presenta uno 0 lungo
%                    la diagonale.
%                    Ritorna x vettore soluzione.
[m,n] = size(A);
if m<n, error('matrice non valida, m>=n necessario'); end
QR = A;
x = b;
for i = 1:n
    v = [1;QR(i+1:m,i)];
    beta = -2/(norm(v)^2);
    x(i:m) = x(i:m) + (beta*(v'*x(i:m)))*v;
end
for i=n:-1:1
    if QR(i,i) == 0, error ("matrice non valida"); end
    x(i) = x(i)/QR(i,i);
    x(1:i-1) = x(1:i-1)-x(i)*QR(1:i-1,i);
end
return
end
```

Esempi per testare miaqr e qrfat:

```
Ax = b => QRx=b

Genero matrice causale 4*4 (Caso semplice in cui l'operatore \ funziona correttamente)

>> A = randi([1, 9], [4,4])

A =

     2     7     3     5
     6     7     9     9
     3     5     2     1
     6     1     8     4

>> b = randi([1 ,9], [4,1])
```

```
b =

     1
     9
     1
     7

>> format short e
>> [QR] = qrfat(A)

QR =

  -9.2195e+00  -8.3518e+00  -1.2365e+01  -9.8703e+00
   5.3478e-01   7.3653e+00  -1.7251e-01   3.3352e+00
   2.6739e-01  -1.0438e-01   2.2530e+00   3.7853e+00
   5.3478e-01   8.4079e-01   2.3943e-01   3.5296e-01
```

Risoluzione sistema:

```
>> [x] = miaqr(QR,b)

x =

   6.8889e+00
  -2.3333e+00
  -6.2222e+00
   4.4444e+00
```

Confronto sul risultato:

```
>> A\b

ans =

   6.8889e+00
  -2.3333e+00
  -6.2222e+00
   4.4444e+00
```

Nel caso in cui M>N l'operatore \ non restituisce tutti i valori della soluzione e lo si evince dalla documentazione:

```
    ...... If A is an M-by-N matrix with M < or > N and B is a column
    vector with M components, or a matrix with several such columns,
    then X = A\B is the solution in the least squares sense to the
    under- or overdetermined system of equations AX = B. The
```

```
    effective rank, K, of A is determined from the QR decomposition
    with pivoting. A solution X is computed which has at most K
    nonzero components per column. If K < N this will usually not
    be the same solution as PINV(A)B.  A\EYE(SIZE(A)) produces a
    generalized inverse of A.
```

Ad ogni modo i codici per la fattorizzazione QR e per risolvere il sistema lineare sono stati testati nel caso di una matrice N*N (caso precedente) e i risultati coincidono.

Ora invece proviamo col caso m=5 e n=3

```
>> A = randi([1, 9], [5,3])

A =

     8     8     2
     8     4     2
     1     9     8
     4     2     6
     3     3     5


>> b = randi([1 ,9], [5,1])

b =

     2
     8
     6
     4
     5


>> [QR] = qrfat(A)

QR =

  -1.2410e+01  -9.8310e+00  -6.3660e+00
   3.9197e-01   8.7949e+00   6.8694e+00
   4.8996e-02  -6.8960e-01  -6.7295e+00
   1.9599e-01   1.2683e-01   6.1997e-01
   1.4699e-01  -3.2164e-02   4.0323e-01
```

```
Risoluzione sistema:

>> [x] = miaqr(QR,b)

x =

   5.0295e-01
  -1.2976e-01
   7.0095e-01
  -4.4518e+00
  -7.3770e-01

L'operatore \ restituisce invece:

>> A\b

ans =

   5.0295e-01
  -1.2976e-01
   7.0095e-01

Come ci si aspettava da quanto descritto in precedenza.
```

Esercizio 15

```
A =

     1     3     2
     3     5     4
     5     7     6
     3     6     4
     1     4     2

b =

    14
    26
    38
    29
    17

>> [QR] = qrfat(A)
```

```
QR =

  -6.7082e+00  -1.1180e+01  -8.6461e+00
   3.8920e-01   3.1623e+00   1.0541e+00
   6.4866e-01   5.9714e-01   3.6515e-01
   3.8920e-01  -1.3068e-01   5.3966e-01
   1.2973e-01  -5.8686e-01   1.5100e-01

>> [x] = miaqr(QR,b)

x =

   1.0000e+00
   3.0000e+00
   2.0000e+00
  -1.5543e-15
   7.2442e-15
```

Esercizio 16-17

Es 16-17

① PER UN GENERICO x_j

$$\Phi_{im}(x_j) = L^2_{im}(x_j)\left[1-2(x_j-x_i)L'_{im}(x_i)\right]$$

QUINDI SICCOME $L'_{im}(x_i)=0$

$$\Phi_{im}(x_j) = L^2_{im}(x_j)\left[1-0\right] = \begin{cases} 1 & \text{se } i=j \\ 0 & \text{se } i\neq j \end{cases} \qquad \boxed{\Phi_{im}(x_j)=\delta_{ij}}$$

② PER UN GENERICO x_j

$$\Psi_{im}(x_j) = (x_j-x_i)L^2_{im}(x_j) = \begin{cases} \text{QUANDO } i=j \Rightarrow \underbrace{(x_j-x_i)}_{0}\underbrace{L^2_{im}(x_j)}_{1} = 0 \\ \text{QUANDO } i\neq j \Rightarrow (x_j-x_i)\underbrace{L^2_{im}(x_j)}_{0} = 0 \end{cases} \qquad \boxed{\Psi_{im}(x_j)=0}$$

③ $\Psi_{im}(x) = (x-x_i)L^2_{im}(x)$

$\Psi'_{im}(x) = L^2_{im}(x)$ PER UN GENERICO x_j $\Psi'_{im}(x_j) = L^2_{im}(x_j) = \delta_{im}$ $\qquad \boxed{\Psi'_{im}(x_j)=\delta_{im}}$

④ $\Phi_{im}(x) = L^2_{im}(x)\left[1-2(x-x_i)L'_{im}(x_i)\right]$

$$\Phi'_{im}(x) = \left(L^2_{im}(x)\right)'\left[1-2(x-x_i)L'_{im}(x_i)\right] + L^2_{im}(x)\left[0-0(x-x_i)L'_{im}(x_i)+2(1-0)L'_{im}(x_i)+2(x-x_i)0\right]$$

PER UN GENERICO x_j

$$\Phi'_{im}(x_j) = \underbrace{\left(L^2_{im}(x_j)\right)'}_{0}\left[1-2(x_j-x_i)L'_{im}(x_i)\right] + \underbrace{L^2_{im}(x_j)}_{\delta_{im}}\cdot 0 = 0 \qquad \boxed{\Phi'_{im}(x_j)=0}$$

Esercizio 18

Es 18

PER UN GENERICO x_J

$$P(x_J) = \sum_{i=0}^{m} \left[f_i \, \Phi_{im}(x_J) + f'_i \, \Psi_{im}(x_J) \right]$$

① INNANZITUTTO SAPPIAMO CHE $\Psi_m(x_J) = 0$ COME DIMOSTRATO NELL' ES. PRECEDENTE QUINDI SVILUPPO LA SOMMATORIA:

$$P(x_J) = f_0 \underbrace{\Phi_{0m}(x_J)}_{0} + f_1 \underbrace{\Phi_{1m}(x_J)}_{0} + \dots + f_J \underbrace{\Phi_{Jm}(x_J)}_{1} + \dots + f_m \underbrace{\Phi_{mm}(x_J)}_{0}$$

COME DIMOSTRATO NELL' ES. PRECEDENTE $\Phi_{im}(x_J) = \delta_{iJ}$ QUINDI TUTTE LE COMPONENTI DELLA SOMMATORIA VALGONO 0 TRANNE $\Phi_{Jm}(x_J)$ CHE VALE 1.

QUINDI:

$$P(x_J) = f_J \underbrace{\Phi_{Jm}(x_J)}_{1} = f_J \quad \checkmark$$

② $$P'(x_J) = \sum_{i=0}^{m} \left[f_i \, \Phi'_{im}(x_J) + f'_i \, \Psi'(x_J) \right]$$

COME DIMOSTRATO NELL' ES. PRECEDENTE SAPPIAMO CHE $\Phi'_{im}(x_J) = 0$ E $\Psi'_{im}(x_J) = \delta_{im}$

QUINDI:

$$P'(x_J) = f'_0 \underbrace{\Psi'_{0m}(x_J)}_{0} + f'_1 \underbrace{\Psi'_{1m}(x_J)}_{0} + \dots + f'_J \underbrace{\Psi'_{Jm}(x_J)}_{1} + \dots + f'_m \underbrace{\Psi'_{mm}(x_J)}_{0}$$

IN MANIERA ANALOGA A PRIMA ABBIAMO QUINDI CHE

$$P'(x_J) = f'_J \underbrace{\Psi'_{Jm}(x_J)}_{1} = f'_J \quad \checkmark$$

Esercizio 19

Polinomio interpolante una funzione con Newton:

```
function [y] = newtonInterpolante(x,f,xx)
%   esercizio 19: [y] = newtonInterpolante(x,fun,xx)
%                       Input: x-> vettore contenente le ascisse xi
%                              f -> valore della funzione nei vari punti
%                              xi
%                              xx -> vettore su cui calcolare il valore del
%                              polinomio.
%                       Output -> valore/valori del polinomio calcolati sui
%                       punti xx.
n = length(x)-1;
if nargin <3 || length(x)<1
    error("Input non validi");
end
if length(f)~=length(x)
    error("numero di fi calcolati non corrisponde col numero di ascisse, errore");
end
for i = 1:length(x)-1
    for j = i+1:length(x)
        if (x(i) == x(j))
           error("Ascisse uguali rilevate");
        end
    end
end
for j = 1:n
    for i = n+1:-1:j+1
        f(i) = (f(i)-f(i-1))/(x(i)-x(i-j));
    end
end
n = length(x);
c = f;
y = c(n)*ones(size(xx));
for i = n-1:-1:1
    y = y.*(xx-x(i))+c(i);
end
return
end
```

Polinomio interpolante una funzione con Lagrange:

```
function [z] = lagrangeInterpolante(x,f,xx)
%   esercizio 19: [z] = lagrangeInterpolante(x,f,xx)
%                       Input: x-> vettore contenente le ascisse xi
%                              f -> valore della funzione nei vari punti
%                              xi
%                              xx -> vettore su cui calcolare il valore del
%                              polinomio.
%                       Output -> valore/valori del polinomio calcolati sui
%                       punti xx.
%   Calcola le differenze divise e utilizza l'algoritmo di Horner per il
%   calcolo del polinomio interpolante.
n=length(x);
cin = ones(n,size(xx,2));
if nargin <3 || length(x)<1 || length(f)~=length(x)
    error("Input non validi");
end
for i = 1:length(x)-1
    for j = i+1:length(x)
        if (x(i) == x(j))
           error("Ascisse uguali rilevate");
        end
    end
end
for i = 1:n
    if (f(i) ~= 0)  % grazie a questo controllo risparmio in totale k*z*n iterazioni,
                    % dove k indica il numero di occorrenze degli elementi
                    % nulli nel vettore fi e z gli elementi di xx.
        for j=1:n
            if (j~=i)
                cin(i,:) = cin(i,:).*(xx-x(j))/(x(i)-x(j));
            end
        end
    end
end
z = 0;
for i=1:n
    z=z+f(i)*cin(i,:);
end
return
end
```

Esercizio 20

Polinomio di Hermite interpolante una funzione con Newton:

```
function [y] = hermiteInterpolante(x,f,xx)
%   esercizio 20: [y] = hermiteInterpolante(x,f,xx)
%                       Input: x-> vettore contenente le ascisse xi del
%                       tipo [x0 x0 x1 x1 etc..]
%                       f -> valore della funzione nei vari punti
%                       e della derivata: [f(x0) f'(x0) f(x1) f'(x1) etc..]
%                       xx -> vettore su cui calcolare il valore del
%                       polinomio.
%                       Output -> valore/valori del polinomio calcolati sui
%                       punti xx.
n = (length(x)/2)-1;
for i = (2*n+1):-2:3
        f(i) = (f(i)-f(i-2))/(x(i)-x(i-1));
end
for j = 2:2*n+1
    for i = (2*n+2):-1:j+1
        f(i) = (f(i)-f(i-1))/(x(i)-x(i-j));
    end
end
n = length(x);
c = f;
y = c(n)*ones(size(xx));
for i = n-1:-1:1
    y = y.*(xx-x(i))+c(i);
end
end
```

Polinomio di Hermite interpolante una funzione con Lagrange:

```
function [points] = lagrangeHermiteInterpolante(x,f,xx)
%   esercizio 20: [z] = lagrangeHermiteInterpolante(x,f,xx)
%                     Input: x-> vettore contenente le ascisse xi nel
%                     formato duplicato [f(x0) f(x0) f(x1) f(x1) ... ]
%                            f -> valore della funzione nei vari punti
%                            nel formato [f(x0) f'(xo) f(x1) f'(x1) ...]
%                            xx -> vettore su cui calcolare il valore del
%                            polinomio.
%                     Output -> valore/valori del polinomio calcolati sui
%                     punti xx.
if nargin <3 || length(x)<1 || length(f)~=length(x)
    error("Input non validi");
end
ff = f(1:2:end);
ff1 = f(2:2:end);
x = x(1:2:end);
n=length(x);
points = zeros(size(xx));
for i = 1:n
    tempx = x;
    ci = tempx(i);
    tempx(i) = [];
    cin = ones(size(xx));
    for j = 1:length(tempx)
        cin = cin.*(xx-tempx(j));
    end
    denominatore = prod(ci-tempx);
    cin = cin/denominatore;
    cin1 = 0;
    for j = 1:length(tempx)
        cin1 = cin1+prod(ci - tempx([1:j-1 j+1:end]));
     end
    cin1 = cin1/denominatore;
    cinquad = cin.^2;
    points = points + ff(i)*cinquad.*(1-2*cin1*(xx-x(i)))+ff1(i)*cinquad.*(xx-x(i));
end
return
end
```

Esercizio 21

Ascisse di Chobyschev:

```
function [xi] = ascisseChobyschev(a,b,n)
%   [xi] = ascisseChobyschev(a,b,n)
%   calcola le  ascisse di chobyshev per il polinomio di interpolazione di
%   grado n su un generico intervallo [a b].
if a >= b || n~=fix(n) || n<0
    error("dati errati")
end
xi = (a+b)/2 + (b-a)/2 * cos((2*(0:n)+1) * (pi/(2*n+2)));
end
```

Esercizio 22

Function Spline0:

```
function [yy] = spline0(x,f,xx)
% esercizio 22: calcola la spline cubica naturale interpolante la
% funzione.
% [yy] = spline0(x,f,xx)
%               Input=> x: vettore delle ascisse già ordinato
%                       f: vettore contenente il valore della funzione sulle
%                       ascisse
%                       xx: punti in cui calcolare la spline interpolante.
%               Output=> yy: punti interpolatori da passare insieme alle
%               ascisse in modo tale da essere plottati.
% Esempio di utilizzo della function:
% a = 0;  b = 1;
% N = 5;
% x = linspace(a,b,N);  %%% x ha dimensione N
% f = cos(x);
% xx = a:0.05:b;
% yy = spline0(x,f,xx);
% plot(x,f,'o',xx,yy)

if ~issorted(x) || length(x)~=length(f)
    error("Input non validi");
end
y =f;
N = length(x);
n = N-1;
deltax = (x(end)-x(1))/n;
%%% definizione matrice A
A = zeros(n-1,n-1);
```

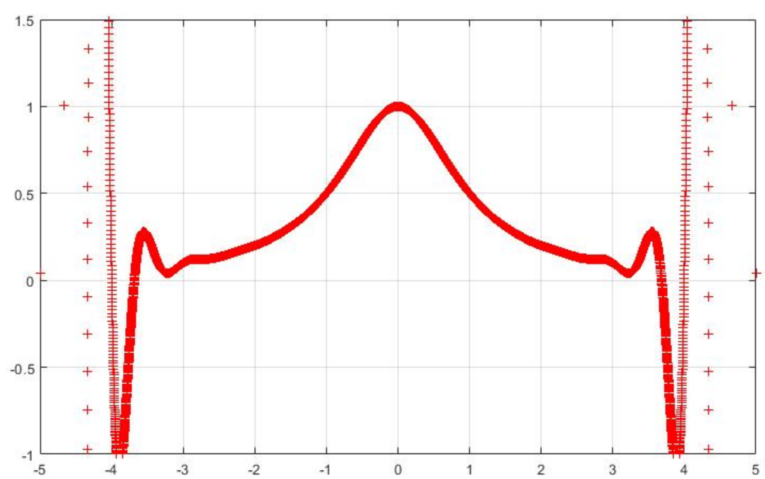
```
h = diff(x);

for i=1:n-1
    for j=1:n-1
        if i==j
            A(i,j) = 2; %%*(2*deltax);
        elseif j == i+1
            A(i,j) = h(i)/(h(i)+h(i+1));%%1/2; %%deltax; %%1/2;
        elseif j == i-1
            A(i,j) = h(i+1)/(h(i)+h(i+1));%%1/2; %%deltax; %%1/2;
        end
    end
end
%%% definizione vettore b
b = zeros(n-1,1);
for i = 1:n-1
    b(i) = 6*(((y(i+2)-y(i+1))/(x(i+2)-x(i+1))) -
              ((y(i+1)-y(i))/(x(i+1)-x(i))))/(x(i+2)-x(i));
end

%%% trovo m
m = A\b; %%OPPURE TRAMITE FATTORIZZAZIONE LU: m = solveLU(A,b);
m = [0;m;0];

for i = 1:n
    r(i) = (m(i) - m(i+1))*deltax/6 - (y(i) - y(i+1))/deltax;
    q(i) = -m(i+1)/6*deltax^2 + y(i+1);
end

%%% definizione s(x) a pezzi
s = zeros(n,N);
for i = 1:n
    s(i,:) = (m(i+1)*(x-x(i)).^3 + m(i)*(x(i+1)-x).^3)/(6*deltax) + q(i)*(x-x(i)) + r(i);
end

xx3 = []; xx1 = []; xx2 = []; yy = [];
for i = 1:n
    vv = pwch(x,y,s(i,:));
    xx3 = xx(xx<=x(i+1));
    xx2 = [xx2,xx1];
    xx1 = setdiff(xx3,xx2);
    yy1 = ppval(vv,xx1);
    yy = [yy,yy1];
end
return
end
```

Esercizio 23-24

```
Punti equispaziati calcolati con:
n = 10001; xx = linspace(-5,5,n);
xline(0, ":"); yline(0, ":");
f = @(x)1./(1+x.^2);
```

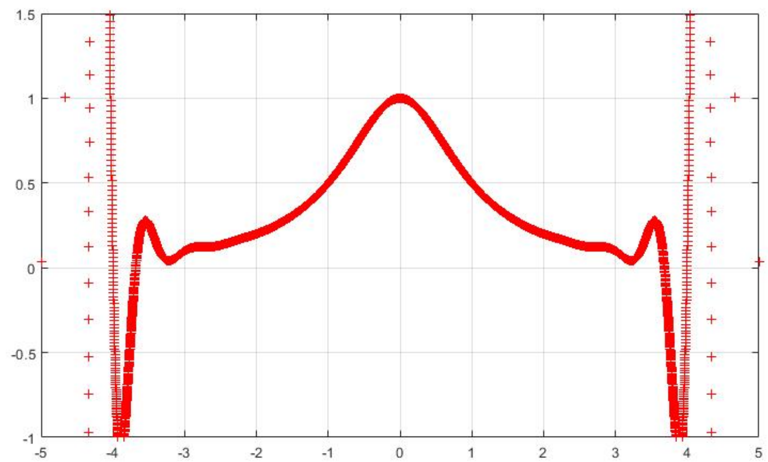
1)q(x) è il polinomio interpolante f(x) su 31 ascisse equidistanti:

```
n = 31; xi = linspace(-5,5,n);
fi = feval(f,xi);
y = newtonInterpolante(xi,fi,xx);
y0 = lagrangeInterpolante(xi,fi,xx);

Newton:
diff = feval(f,xx)-y;
max(abs(diff)) = 2.388280971350641e+03 %errore


plot del polinomio sul grafico:
plot(xx,y,'r+'),drawnow,shg;
axis([-5 5 -1 1.5]); grid on;
```



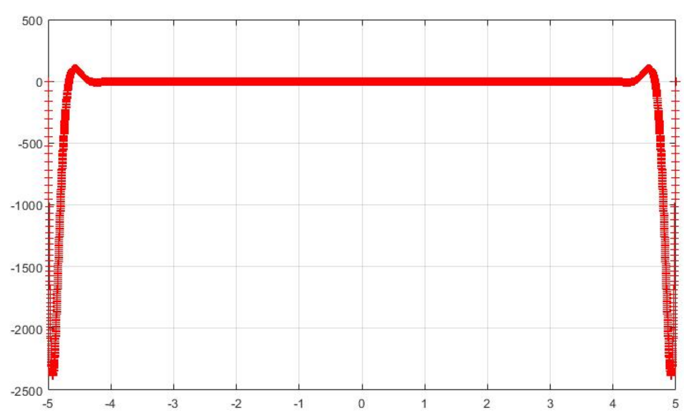
```
Lagrange:
diff = feval(f,xx)-y0;
max(abs(diff)) = 2.388280971350968e+03
```

```
plot del polinomio sul grafico:
plot(xx,y0,'r+'),drawnow,shg;
axis([-5 5 -1 1.5]);grid on;
```



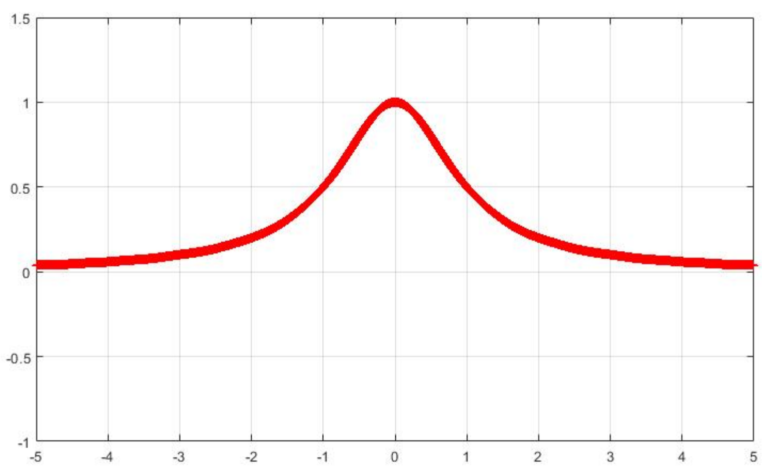
Volendo è possibile ottenere il plot dell'errore:

```
plot dell'errore:  plot(xx,diff,'r+'),drawnow,shg; grid on;
```

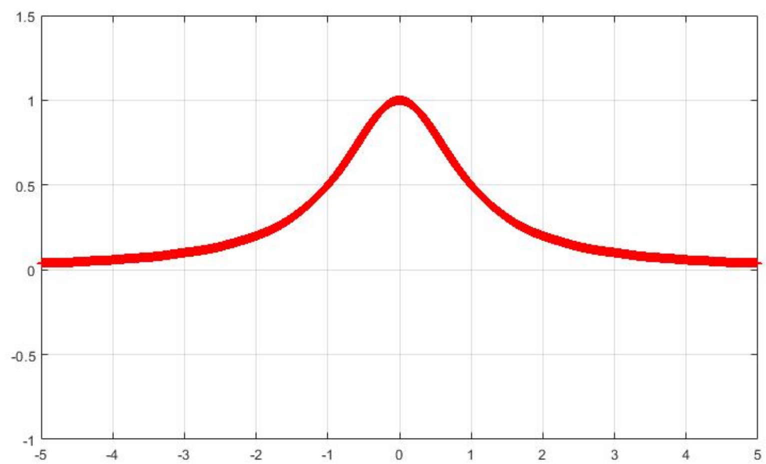


2)q(x) è il polinomio interpolante f(x) su 31 ascisse di Chebyshev:

```
[xi] = ascisseChobyschev(-5,5,30);
fi = feval(f,xi);
y = newtonInterpolante(xi,fi,xx);
y0 = lagrangeInterpolante(xi,fi,xx);

Newton:
diff = feval(f,xx)-y;
max(abs(diff)) = 2.061587838963652e-03 %errore

plot(xx,y,'r+'),drawnow,shg;
axis([-5 5 -1 1.5]); grid on;
```



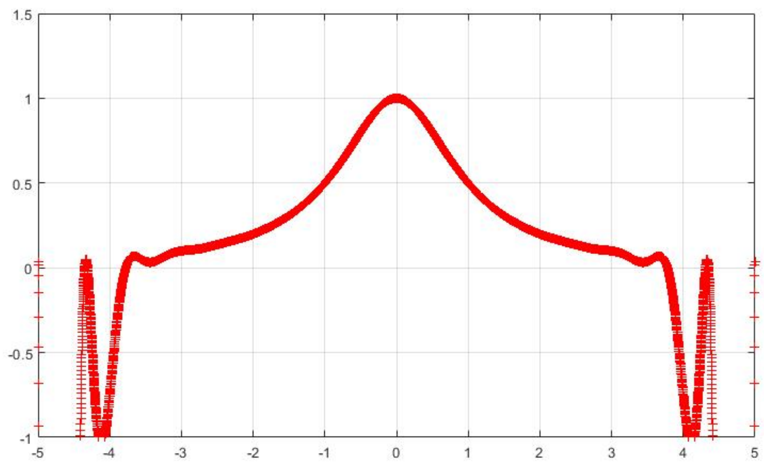
```
Lagrange:
diff = feval(f,xx)-y0;
max(abs(diff)) = 2.061587838963930e-03 %errore

plot(xx,y0,'r+'),drawnow,shg;
axis([-5 5 -1 1.5]); grid on;
```



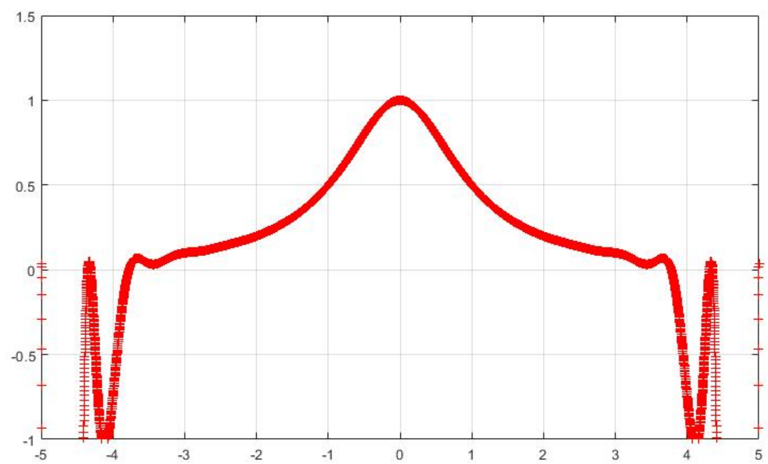
3)q(x) è il polinomio di Hermite interpolante f(x) su 15 ascisse equidistanti:

```
n = 15;
a = -5; b = 5;
   for i = 0:n,
       xi(2*i+1) = a + i*((b-a)/n);
       xi(2*i+2) = xi(2*i+1);
   end

fi = feval(f,xi);
syms x; f1 = matlabFunction(diff(f,x));
fi1 = feval(f1,xi);

for i = 0:n,
       formattedF(2*i+1) = fi(2*i+1);
       formattedF(2*i+2) = fi1(2*i+1);
end
y = hermiteInterpolante(xi,formattedF,xx);
y0 = lagrangeHermiteInterpolante(xi,formattedF,xx);
```

```
%hermite normale
diff = feval(f,xx)-y;
max(abs(diff)) = 0.0107562214439

plot(xx,y,'r+'),drawnow,shg;
axis([-5 5 -1 1.5]); grid on;
```



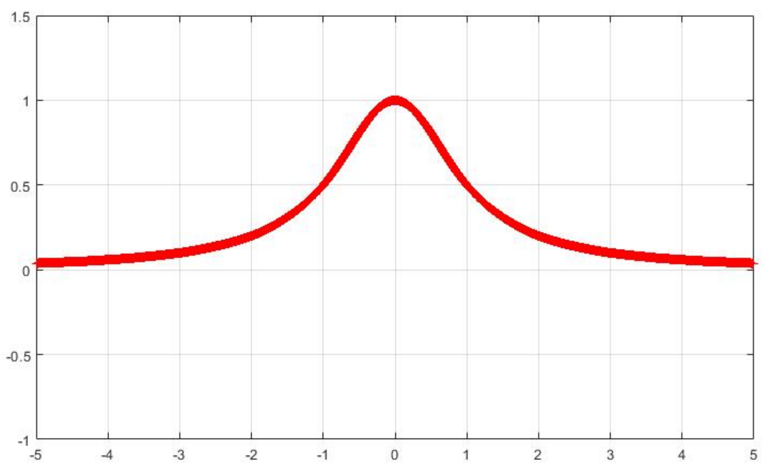
```
%lagrange hermite
diff = feval(f,xx)-y0;
max(abs(diff)) = 0.0107562214439

plot(xx,y0,'r+'),drawnow,shg;
axis([-5 5 -1 1.5]); grid on;
```



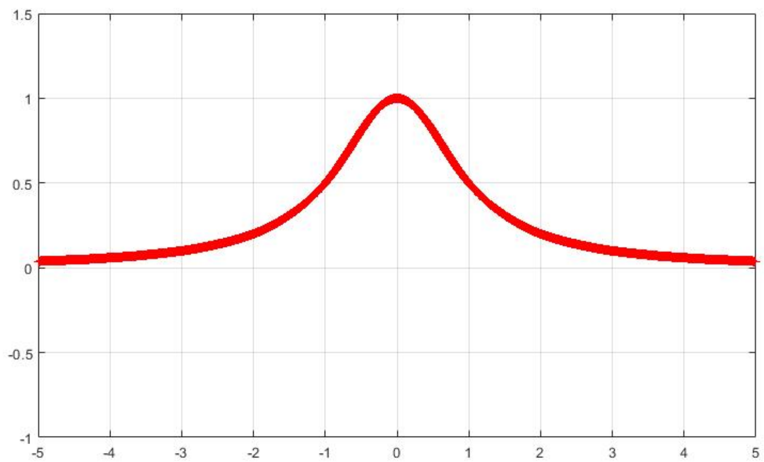
4)q(x) è il polinomio di Hermite interpolante f(x) su 15 ascisse di chobychev:

```
n = 14; %2*n+2 ascisse, da 0 a 14 sono 15 ascisse.
[xtemp] = ascisseChobyschev(-5,5,n);

for i = 0:n
    xi(2*i+1) = xtemp(i+1);
    xi(2*i+2) = xtemp(i+1);
end

fi = feval(f,xi);
syms x; f1 = matlabFunction(diff(f,x));
fi1 = feval(f1,xi);

for i = 0:n,
       formattedF(2*i+1) = fi(2*i+1);
       formattedF(2*i+2) = fi1(2*i+1);
end
```

```
y = hermiteInterpolante(xi,formattedF,xx);
y0 = lagrangeHermiteInterpolante(xi,formattedF,xx);

%hermite normale
diff = feval(f,xx)-y;
max(abs(diff)) = 0.008286208572992

plot(xx,y,'r+'),drawnow,shg;
axis([-5 5 -1 1.5]); grid on;
```

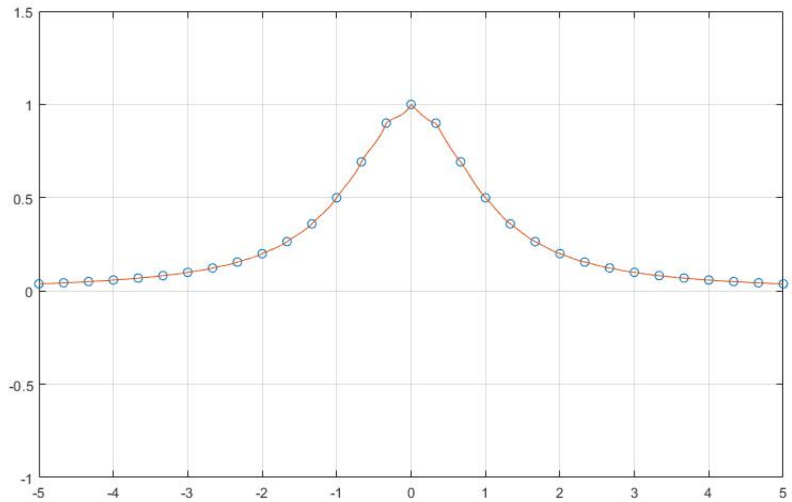


```
%hermite lagrange
diff = feval(f,xx)-y0;
max(abs(diff)) = 0.008286208572992

plot(xx,y,'r+'),drawnow,shg;
axis([-5 5 -1 1.5]); grid on;
```



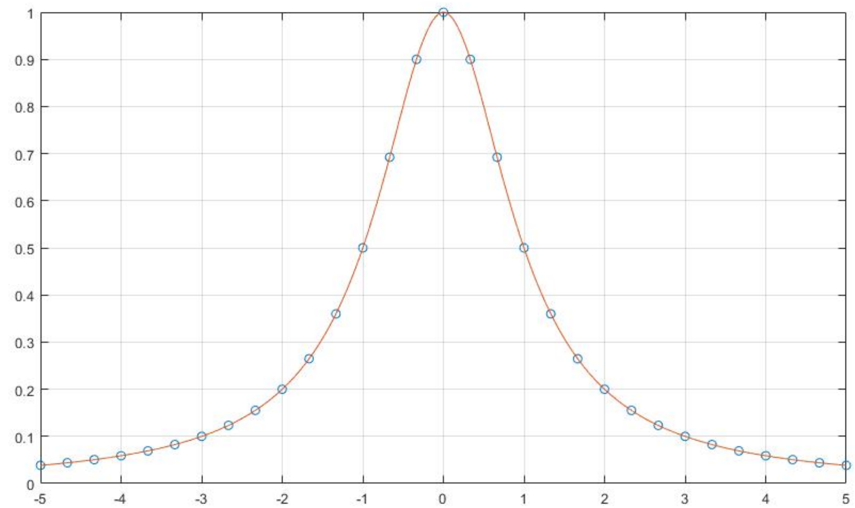
5)q(x) è la spline cubica naturale interpolante f(x) su 31 ascisse equidistanti:

```
n = 31;
xi = linspace(-5,5,n);
fi = feval(f,xi);
yy = spline0(xi,fi,xx);
plot(xi,fi,'o',xx,yy)
axis([-5 5 -1 1.5]); grid on;

errore:

diff = feval(f,xx)-yy;
max(abs(diff)) = 0.038943094048244
```



6)q(x) è la spline cubica not-a-knot interpolante f(x) su 31 ascisse equidistanti:

```
n = 31;
xi = linspace(-5,5,n);
fi = feval(f,xi);
yq = spline(xi,fi,xx);
figure
plot(xi,fi,'o',xx,yq); grid on;

errore:

diff = feval(f,xx)-yq;
max(abs(diff)) = 8.243623335756345e-04
```



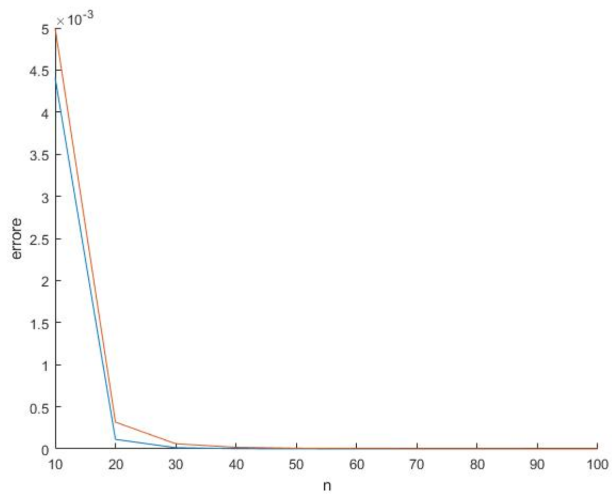
Esercizio 25

```
a = 0; b = 2*pi;
n = 10001; xx = linspace(a,b,n);
f1 = @(x)sin(x);
f2 = @(x)cos(x);


erroreSinKnotKnot = []; erroreCosKnotKnot =[];
erroreSinNaturali = []; erroreCosNaturali =[];
for i = 10:10:100
    xi = linspace(a,b,i);
    fi1 = feval(f1,xi);
    fi2 = feval(f2,xi);

    %knot-a-knot
    ysinKnotKnot = spline(xi,fi1,xx);
    ycosKnotKnot = spline(xi,fi2,xx);

    erroreSinKnotKnot(i/10) = max(abs(feval(f1,xx)-ysinKnotKnot));
    erroreCosKnotKnot(i/10) = max(abs(feval(f2,xx)-ycosKnotKnot));

    %spline naturale
    ysin = spline0(xi,fi1,xx);
    ycos = spline0(xi,fi2,xx);
```

```
    erroreSinNaturali(i/10) = max(abs(feval(f1,xx)-ysin));
    erroreCosNaturali(i/10) = max(abs(feval(f2,xx)-ycos));

end
x = 10:10:100;
xlabel("n");
ylabel("errore");
hold on;
```

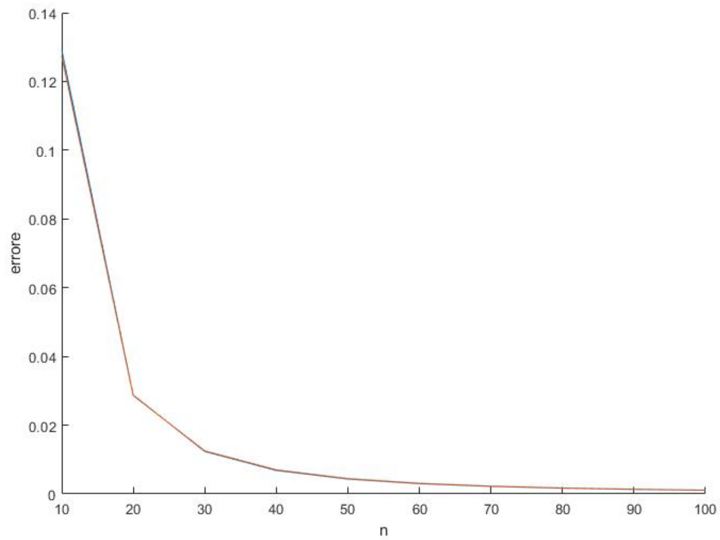
andamento errore nelle knot-a-knot:

```
plot(x,erroreSinKnotKnot(1:end));
plot(x,erroreCosKnotKnot(1:end));
```



andamento errore nelle naturali:

```
plot(x,erroreSinNaturali(1:end));
plot(x,erroreCosNaturali(1:end));
```



Esercizio 26

Coefficienti di Newton-Cotes:

```
function [cin] = coefficientiNC(n)
%esercizio 26: [cin] = coefficientiNC(n)
%                      n: grado per calcolare coefficienti di newton cotes.
cin = ones(1,n);
for i=0:n
    a = poly([0:i-1 i+1:n]);
    b = [a./(n+1:-1:1) 0];
    cin(i+1) = polyval(b,n)/((-1)^(n-i)*factorial(n-i)*factorial(i));
end
return
end
```

Tabulazione:

grado n = 1	1/2	1/2								
grado n = 2	1/3	4/3	1/3							
grado n = 3	3/8	9/8	9/8	3/8						
grado n = 4	14/45	64/45	8/15	64/45	14/45					
grado n = 5	95/288	125/96	125/144	125/144	125/96 95/288					
grado n = 6	41/140	54/35	27/140	68/35	27/140	54/35	41/140			
grado n = 7	1073/3527	810/559	343/640	649/536	649/536	343/640	810/559	1073/3527		
grado n = 9	130/453	1374/869	243/2240	5287/2721	704/1213	704/1213	5594/2879	243/2240	1374/869	130/453]

Esercizio 27

```
function [newtonCotesNormale,sommaSottoIntervalli,erroreCalcolato] ...
                        = approssimazioneIntegraleCon2Intervalli(fun,a,b,n)
%   esercizio 27: approssimazioneIntegraleCon2Intervalli(fun,a,b,n);
%   Suddivide l'intervallo in 2 parti e calcola la funzione nei vari sottointervalli.
%   OUTPUT=> newtonCotesNormale: funzione calcolata tramite coefficienti di newton-cotes sul
%            sommaSottoIntervalli: restituisce la somma dei valori nei 2 intervalli
%            erroreCalcolato: valore assoluto dell'errore tra le 2 forme utilizzate
h=(b-a)/n;
f = feval(fun,linspace(a,b,n+1));
fsottoIntervallo1 = feval(fun,linspace(a,(a+b)/2,n+1));
fsottoIntervallo2 = feval(fun,linspace((a+b)/2,b,n+1));
nc = coefficientiNC(n);
newtonCotesNormale = h*sum(nc.*f);
I1 = sum(nc.*fsottoIntervallo1);
I2 = sum(nc.*fsottoIntervallo2);
sommaSottoIntervalli = h/2*(I2+I1);
erroreCalcolato = abs(newtonCotesNormale-sommaSottoIntervalli);%%/((2.^n)-1);
return
end
```

Esercizio 28

Script usato per tabulare i risultati:

```
a = 0; b = pi;
fun = @(x)exp(sin(x));

for i = 1:9
    if (i~=8)
        [ncNormale,approx,errore] = approssimazioneIntegraleCon2Intervalli(fun,a,b,i);
        fprintf("Valore di n: %d Approssimazione %d Errore: %.e\n",i,approx,errore);
    end
end
```

Tabulazione:

Valore di n	Approssimazione	Errore
1	5.840663e+00	3e+00
2	6.194562e+00	5e-01
3	6.203379e+00	2e-01
4	6.209658e+00	5e-02
5	6.209247e+00	3e-02
6	6.208718e+00	7e-03
7	6.208734e+00	4e-03
9	6.208759e+00	6e-04

Esercizio 29

```
function [I2,np] = simpsonAdattivo(f,a,b,tol,fa,f1,fb)
%   esercizio 29 [I2,np] = simpsonAdattivo(f,a,b,tol,fa,f1,fb)
%   f => funzione su cui effettuare l'approssimazione.
%   a,b => estremi intervallo di integrazione
%   %tol => tolleranza passata
%   ignorare gli ultimi 3 input: vengono usati dalle chiamate ricorsive
%   della function.
global np;
x1 = (a+b)/2;
if nargin<=4
    fa = feval(f,a);
    fb = feval(f,b);
    f1 = feval(f,x1);
    np = 3;
end
h = (b-a)/6;
x2 = (a+x1)/2;
x3 = (x1+b)/2;
```

```
f2 = feval(f ,x2);
f3 = feval(f ,x3);
np = np+2;
I1 = h*(fa+4*f1+fb);
I2 = .5*h*(fa+4*f2+2*f1+4*f3+fb);
e = abs(I2-I1)/15;
if e>tol || np<10
    I2 = simpsonAdattivo(f,a,x1,tol/2,fa,f2,f1) +...
        simpsonAdattivo(f,x1,b,tol/2,f1,f3,fb);
end
%if nargin<=4,fprintf("%i punti utilizzati \n",np); end
end
```

Esercizio 30

Funzione passata a **simpsonAdattivo**

```
function y = cos20(x)
%Funzione cos(x^20)
y = cos(x.^20);
return
end
```

Script usato per tabulare i risultati:

```
format short e;
for i = 2:10
    tol = 10.^(-i);
    [I2,np] = simpsonAdattivo('cos20',a,b,tol);
    fprintf("Tolleranza usata: %.e approssimazione: %.e ,
                   Numero di punti usati: %d \n",tol,I2,np);
end
```

Tabulazione:

Tolleranza usata	Approssimazione	Numero di punti usati
1e-02	2e+00	17
1e-03	2e+00	17
1e-04	1e+00	49
1e-05	1e+00	89
1e-06	1e+00	141
1e-07	1e+00	241
1e-08	1e+00	389
1e-09	1e+00	621
1e-10	1e+00	1069