ESERCITAZIONE MATLAB 3: Equazioni non lineari

1. Si scriva un M-file di tipo function che implementa il metodo di Newton per il calcolo di una radice di una equazione non lineare.

2. Al fine di verificare se il metodo di Newton è stato implementato correttamente lo si utilizzi per calcolare la soluzione di

$$f(x) \equiv x - 1 = 0$$

utilizzando vari valori dell'approssimazione iniziale `x0`. Se la function che implementa il metodo è corretta allora, indipendentemente da `x0`, si deve ottenere la radice applicando, al massimo, 2 iterazioni.

3. Si utilizzi il metodo di Newton implementato per calcolare la soluzione della equazione

$$f(x) \equiv \mathtt{e}^{x-1} - 1 = 0.$$

4. È ben noto che la seguente equazione

$$f(x) \equiv \arctan(x) = 0$$

ammette come unica soluzione $x^* = 0$. Si applichi il metodo di Newton per determinarla e si verifichi che il metodo converge soltanto se l'approssimazione iniziale è "sufficientemente" vicina a x^*.

5. Si utilizzi il metodo di Newton per calcolare la soluzione della seguente equazione

$$f(x) \equiv (x-1)^2 \mathtt{e}^x = 0 \tag{1}$$

utilizzando `x0 = 2` come approssimazione iniziale della radice $x^* = 1$ che ha molteplicità $m = 2$. Si scriva poi un M-file di tipo function, da salvare con il nome `newtonmod.m`, che implementi la versione modificata del metodo di Newton per la approssimazione di radici multiple con molteplicità nota a priori. Si utilizzi quindi `newtonmod` per risolvere (1) e si verifichi che il metodo di Newton modificato converge molto più velocemente. Si ripeta lo stesso esperimento per l'equazione

$$f(x) \equiv (x-1)^3 \mathtt{e}^x = 0. \tag{2}$$

6. Si scriva un M-file di tipo function che implementa il metodo delle secanti per il calcolo di una radice di una equazione non lineare.

SOLUZIONI:

1.
```
function xv = newton(x0,fun,dfun,tolx,itmax)
% xv = newton(x0,fun,dfun,tolx,itmax)
%
% Implementa il metodo di Newton per il calcolo di una radice
% della equazione non lineare
%
% fun(x) = 0
%
% Input:  x0   --> approssimazione iniziale
%         fun  --> funzione di cui si vuole calcolare lo zero
%        dfun  --> derivata prima di fun
%        tolx  --> tolleranza per il criterio di arresto
%       itmax  --> numero massimo di iterazioni consentite
%
% Output:   xv --> vettore contenente la sequenza di approssimazioni
%                  calcolata durante la applicazione del metodo.
%                  L'ultima componente e' la approssimazione
%                  che soddisfa il criterio di arresto
x  = x0;
xv = x0;
dx = tolx+1;
it = 1;

while ((abs(dx)>tolx) & (it<=itmax))

    f   = feval(fun,x);
    if (f==0),   return,  end

    df  = feval(dfun,x);
    if (df==0),
       warning('Derivata prima nulla: impossibile proseguire'),
       return
    end

    dx = -f/df;
    x  = x + dx;
    xv = [xv;x];
    it  = it + 1;

end

if (abs(dx)>tolx),
   warning('Il metodo di Newton non converge o numero massimo di iterazioni insufficiente'),
end
```

2.
```
>> fun1 = @(x) x - 1;
>> dfun1 = @(x) 1;
>> xv = newton(2,fun1,dfun1,1e-14,100)
xv =
     2
     1
>> xv = newton(10,fun1,dfun1,1e-14,100)
xv =
    10
     1
```

3.
```
>> format long
>> fun2 = @(x) exp(x-1) - 1;
>> dfun2 = @(x) exp(x-1);
>> xv = newton(5,fun2,dfun2,1e-3,100)
xv =
   5.000000000000000
   4.018315638888734
   3.067199125390316
   2.193738833067194
   1.496824788299695
   1.105284372010052
   1.005352903902150
   1.000014301260955
   1.000000000102263

>> xv = newton(5,fun2,dfun2,1e-10,100)
xv =
   5.000000000000000
   4.018315638888734
   3.067199125390316
   2.193738833067194
   1.496824788299695
   1.105284372010052
   1.005352903902150
   1.000014301260955
   1.000000000102263
   1.000000000000000

>> xv = newton(-1,fun2,dfun2,1e-3,100)
xv =
  -1.000000000000000
   5.389056098930650
   4.401468538738613
   3.434792834728056
   2.522408730867069
   1.740594433718539
   1.217424820918349
   1.022012911758287
   1.000240516089578
   1.000000028921676
```

```
>> xv = newton(-1,fun2,dfun2,1e-10,100)
xv =
  -1.000000000000000
   5.389056098930650
   4.401468538738613
   3.434792834728056
   2.522408730867069
   1.740594433718539
   1.217424820918349
   1.022012911758287
   1.000240516089578
   1.000000028921676
   1.000000000000000
   1.000000000000000
```

4.
```
>> fun3 = @(x) atan(x);;
>> dfun3 = @(x) 1./(1+x.^2);
>> xv = newton(1,fun3,dfun3,1e-10,100)
xv =
   1.000000000000000
  -0.570796326794897
   0.116859903998913
  -0.001061022117045
   0.000000000796310
                   0

>> xv = newton(1.5,fun3,dfun3,1e-10,100);
Warning: Derivata prima nulla: impossibile proseguire
> In newton at 38
>> format short e
>> xv
xv =
   1.5000e+00
  -1.6941e+00
   2.3211e+00
  -5.1141e+00
   3.2296e+01
  -1.5753e+03
   3.8950e+06
  -2.3830e+13
   8.9203e+26
  -1.2499e+54
   2.4540e+108
  -9.4595e+216
```

Si osserva che scegliendo $x_0 = 1.5$, la successione di approssimazioni fornita dal metodo di Newton ha segno alterno ed è divergente in modulo. Dato che $f'(x) \to 0$ per $x \to \pm\infty$, accade che la procedura iterativa viene arrestata poichè, in aritmetica finita, si trova $f'(x_i) = 0$ per un certo indice i;

5. Il codice per la versione modificata del metodo di Newton è il seguente

```
function xv = newtonmod(x0,fun,dfun,tolx,itmax,m)
% xv = newtonmod(x0,fun,dfun,tolx,itmax,m)
%
% Implementa il metodo di Newton per il calcolo di una radice
% della equazione non lineare
%
% fun(x) = 0
%
% Input:  x0   --> approssimazione iniziale
%         fun  --> funzione di cui si vuole calcolare lo zero
%        dfun  --> derivata prima di fun
%        tolx  --> tolleranza per il criterio di arresto
%       itmax  --> numero massimo di iterazioni consentite
%           m  --> molteplicita' della radice
%
% Output:   xv --> vettore contenente la sequenza di approssimazioni
%                  calcolata durante la applicazione del metodo.
%                  L'ultima componente e' la approssimazione
%                  che soddisfa il criterio di arresto
x  = x0;
xv = x0;
dx = tolx+1;
it = 1;

while ((abs(dx)>tolx) & (it<=itmax))

    f   = feval(fun,x);
    if (f==0),   return,  end

    df  = feval(dfun,x);
    if (df==0),
       warning('Derivata prima nulla: impossibile proseguire'),
       return
    end

    dx = -m*f/df;
    x  = x + dx;
    xv = [xv;x];
    it  = it + 1;

end

if (abs(dx)>tolx),
   warning('Il metodo di Newton non converge o numero massimo di iterazioni insufficiente'),
end
```

Esempi di applicazione (nota: in generale, se `x` è un vettore allora digitando `x(end)` si accede alla sua ultima componente):

```
>> format short e
>> fun4 = @(x) (x-1).^2.*exp(x);
>> dfun4 = @(x) (x-1).*(x+1).*exp(x);
>> xv = newton(2,fun4,dfun4,1e-10,100);
>> length(xv)
ans =

    36

>> xv(end)-1
ans =

   7.2006e-11

>> xvmod = newtonmod(2,fun4,dfun4,1e-10,100,2);
>> length(xvmod)
ans =

     7

>> xvmod(end)-1
ans =

     0
```

In Figura 1 sono stati riportati gli errori di convergenza dei due precedenti metodi. I comandi digitati per tracciare il grafico sono i seguenti:

```
>> semilogy([0:length(xv)-1],abs(xv-1),'b-x',[0:length(xvmod)-1],abs(xvmod-1),'r-*')
>> legend('Newton classico','Newton modificato (m=2)')
>> title('f(x) = (x-1)^2 exp(x)')
>> xlabel('i')
>> ylabel('|x_i -1|')
```

Il vantaggio dell'utilizzo di Newton modificato rispetto a Newton classico è ancora più evidente nel caso della equazione (2) (chiaramente specificando tre come ultimo parametro di input per `newtonmod`).

6. Il codice per il metodo delle secanti è il seguente

```
 function xv = secanti(x0,x,fun,tolx,itmax)

% xv = secanti(x0,x,fun,tolx,itmax)
%
% Implementa il metodo di Newton per il calcolo di una radice
% della equazione non lineare
%
% fun(x) = 0
%
```

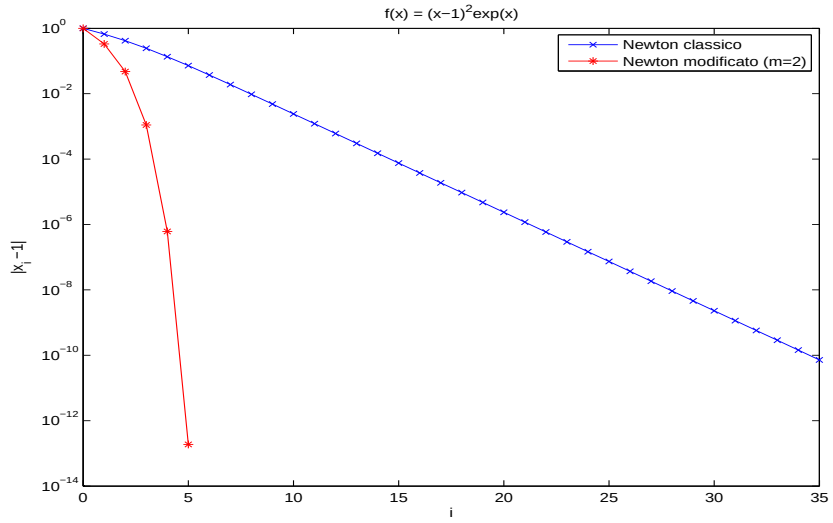

Figure 1: Errori di convergenza per Newton classico e Newton modificato applicati per risolvere $(x-1)^2 \exp(x) = 0$.

```
% Input: x0,x   --> punti di innesco
%         fun   --> funzione di cui si vuole calcolare lo zero
%         tolx  --> tolleranza per il criterio di arresto
%        itmax  --> numero massimo di iterazioni consentite
%
% Output: xv --> vettore delle approssimazioni successive
%                calcolate durante la applicazione del metodo.
%                L'ultima componente e' la approssimazione
%                che soddisfa il criterio di arresto

xv   = x0;
f0   = feval(fun,x0);
if (f0==0), return, end

xv  = [xv;x];
dx = x-x0;
it = 1;

while ((abs(dx)>tolx) & (it<=itmax))
    f   = feval(fun,x);
    if (f==0), return, end

    df  = (f-f0)/dx;
    if (df==0),
       warning('Impossibile proseguire con l''applicazione del metodo delle secanti'),
       return
    end

    x0 = x;
```

```
    f0 = f;
    dx = -f/df;
    x  = x + dx;
    xv = [xv;x];
    it  = it + 1;
end

if (abs(dx)>tolx),
   warning('Il metodo delle secanti non converge o numero massimo di iterazioni insufficiente'),
end
```

Esempi di applicazione (li si confronti con i primi due test del terzo esercizio)

```
>> fun2=@(x) exp(x-1)-1;
>> dfun2=@(x) exp(x-1);
>> format long
>> xv = secanti(5,4,fun2,1e-3,100)
xv =

   5.000000000000000
   4.000000000000000
   3.446998207224081
   2.762959409195180
   2.185791820654120
   1.672543092292245
   1.297892381475660
   1.085525815942989
   1.011951323389289
   1.000502814202586
   1.000002998412371

>> xv = secanti(5,4,fun2,1e-10,100)
xv =

   5.000000000000000
   4.000000000000000
   3.446998207224081
   2.762959409195180
   2.185791820654120
   1.672543092292245
   1.297892381475660
   1.085525815942989
   1.011951323389289
   1.000502814202586
   1.000002998412371
   1.000000000753759
   1.000000000000001
   1.000000000000000
```