Serie 4 - Soluzione
Equazioni Non Lineari

Esercizio 1

Supponiamo di inserire tutti i seguenti comandi, necessari per risolvere l'esercizio, in uno script di Matlab®.

1. La definizione di $f(x)$ come `@` function e il grafico richiesto si ottengono con i seguenti comandi:

```
f = @(x) x.^3 - (2+exp(1))*x.^2 + (2*exp(1)+1)*x + (1-exp(1)) - cosh(x-1);
x = linspace(0.5, 6.5, 100);
y = f(x);
figure(1);
plot(x,y)
title('f(x)=x.^3 - (2+exp(1))*x.^2 + (2*exp(1)+1)*x + (1-exp(1)) - cosh(x-1)');
xlabel('x');
ylabel('y');
grid on
y0 = zeros(100,1);
hold on
plot(x,y0)
```

   Il grafico risultante è mostrato in Figura 1.

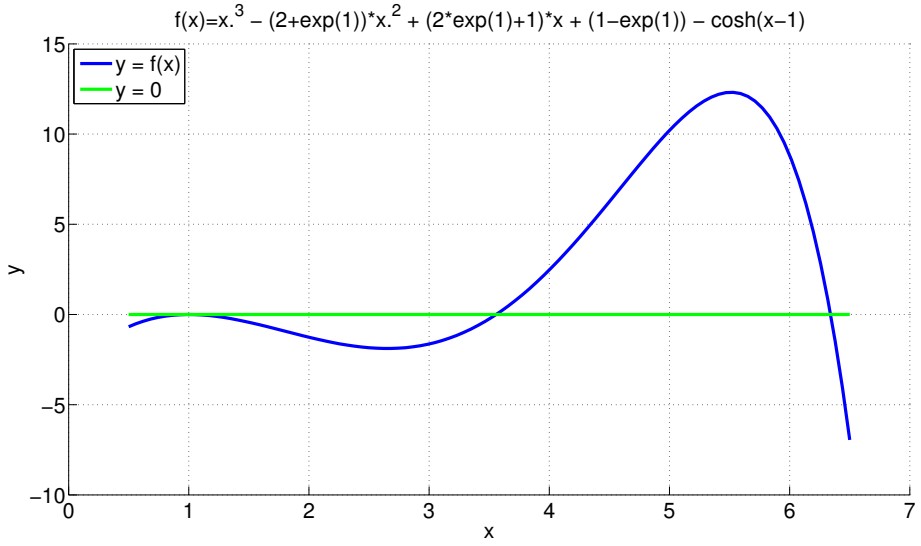

Figura 1: Grafico della funzione $y = f(x)$ in blu e della retta $y = 0$ in verde.

2. La verifica dell'applicabilità del metodo di bisezione si ottiene con i seguenti comandi:

```
disp('la prima radice appartiene all''intervallo [0.5, 1.5]');
if (f(0.5)*f(1.5)) < 0
    disp('si puo'' applicare il metodo di bisezione alla prima radice');
else
    disp('non si puo'' applicare il metodo di bisezione alla prima radice');
end

disp('la seconda radice appartiene all''intervallo [3, 4]');
if (f(3)*f(4)) < 0
    disp('si puo'' applicare il metodo di bisezione alla seconda radice');
else
    disp('non si puo'' applicare il metodo di bisezione alla seconda radice');
end

disp('la terza radice appartiene all''intervallo [6, 6.5]');
if (f(6)*f(6.5)) < 0
    disp('si puo'' applicare il metodo di bisezione alla terza radice');
else
    disp('non si puo'' applicare il metodo di bisezione alla terza radice');
end
```

che stampa a schermo:

```
non si puo' applicare il metodo di bisezione alla prima radice

si puo' applicare il metodo di bisezione alla seconda radice

si puo' applicare il metodo di bisezione alla seconda radice
```

3. Una possibile implementazione della funzione richiesta è la seguente:

```
function [xvect,it]=bisez(a,b,toll,fun)

%
% [xvect, it]=bisez(a,b,toll,fun)
%
% Metodo di bisezione per la risoluzione
% dell'equazione non lineare f(x)=0
%
% Parametri di ingresso:
%
% a,b       Estremi intervallo di ricerca radice
% toll      Tolleranza sul test d'arresto
% f         Funzione definita come inline
%
% Parametri di uscita:
%
% xvect     Vett. contenente tutte le iterate
%           calcolate (l'ultima componente e' la soluzione)
% it        Iterazioni effettuate

it=-1; % in questo modo la prima iterazione del ciclo while calcola x0,
```

```
       % la seconda iterazione di while calcola x1...

xvect=[];
err=toll+1;
nmax=ceil(log2((b - a)/toll) -1 );
fprintf('Massimo numero di iterazioni ammissibili %-d \n',nmax);

if (fun (a) * fun (b) > 0)
    error ('La funzione deve avere segno diverso nei due estremi');
end

while (it < nmax && err > toll)

    it=it+1;
    x = (b+a)/2; %stima dello zero
    fc = fun(x);

    if (fc == 0)
       err=0;
    else
       err=abs(fc);
    end

    xvect=[xvect;x]; % aggiornamento soluzione


    % scelta del nuovo estremo per l'eventuale ciclo successivo
    if (fc*fun(a) > 0),
          a=x;
    else
          b=x;
    end;

end;

if (it==nmax)
    fprintf('ATTENZIONE!')
    fprintf('Massimo numero di iterazioni raggiunte!  Errore sul residuo %-6.4e \n',err);
else
    fprintf('x_%-d soddisfa la tolleranza sul residuo \n', it);
end
fprintf(' Radice calcolata       : %-12.8f \n', xvect(end));
```

4. Con i seguenti comandi si ottengono le operazioni richeste:

```
toll = 1e-12;

disp('Calcolo seconda radice')
a1 = 3; b1 = 4;
[x1,it1]=bisez(a1,b1,toll,f);

disp('Calcolo terza radice')
a2 = 6; b2 = 6.5;
[x2,it2]=bisez(a2,b2,toll,f
```

e la seguente stampa a schermo:

```
Calcolo seconda radice
Massimo numero di iterazioni ammissibili 39
 ATTENZIONE ! Massimo numero di iterazioni raggiunte!  Errore sul residuo 9.2903e-13
 Radice calcolata       : 3.55780416

Calcolo terza radice
Massimo numero di iterazioni ammissibili 38
 ATTENZIONE ! Massimo numero di iterazioni raggiunte!  Errore sul residuo 1.2349e-11
 Radice calcolata       : 6.34045080
```

Esercizio 2

Supponiamo di inserire tutti i seguenti comandi, necessari per risolvere l'esercizio, in uno script di Matlab®.

1. La definizione di $f(x)$ e $f'(x)$ come `@` function e i loro grafici si ottengono con i seguenti comandi:

```
f = @(x) x.^3 - (2+exp(1))*x.^2 + (2*exp(1)+1)*x + (1-exp(1)) - cosh(x-1);
x = linspace(0.5, 6.5, 100);
y = f(x);
figure(1);
plot(x,y)
title('f(x)=x^3 - (2+e)x^2 + (2e+1)x + (1-e) - cosh(x-1)');
xlabel('x');
ylabel('y');
grid on
y0 = zeros(100,1);
hold on
plot(x,y0)

df = @(x) 3*x.^2 - 2*(2+exp(1))*x + (2*exp(1)+1) - sinh(x-1);
dy = df(x);
figure(2)
plot(x,dy,'b', x,y, 'r', x,y0, 'g')
title('df(x)=3x^2 - 2(2+e)x + (2e+1) - sinh(x-1)');
xlabel('x');
ylabel('y');
legend('y=df(x)', 'y=f(x)', 'y=0', 'Location', 'SouthWest')
grid on
```

   Il grafico risultante è mostrato in Figura 2.

   Osserviamo che la radice $\alpha_1 \in (0.5, 1.5)$ non è sicuramente una radice semplice, in quanto anche $f'(\alpha_1) = 0$. La seconda radice $\alpha_2 \in (3, 4)$ e la terza $\alpha_3 \in (6, 6.5)$ sono semplici in quanto $f'(\alpha_2)$ e $f'(\alpha_3)$ sono visibilmente diverse da zero. Utilizzando il metodo di Newton classico per calcolare α_1 ci aspettiamo di avere una convergenza lineare, mentre per α_2 e α_3 ci spettiamo di avere una convergenza quadratica.

   Verifichiamo la molteplicità della radice $\alpha_1 = 1$ tramite i seguenti comandi:

```
% discussione sulle radici dal grafico di f e df
```

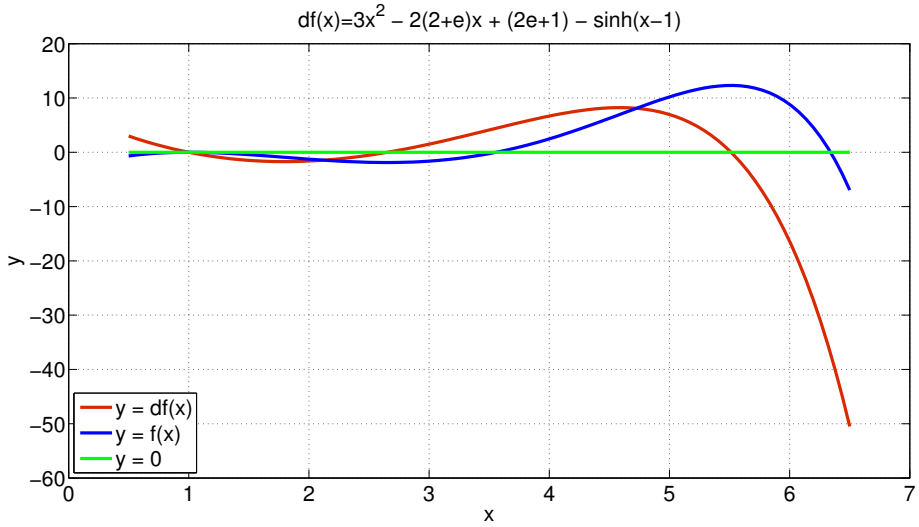

Figura 2: Grafico della funzione $f(x)$ in blu, della sua derivata $f'(x)$ in rosso e della retta $y = 0$ in verde.

```
d2f = @(x) 6*x - 2*(2+exp(1)) - cosh(x-1);
alpha1 = 1;
if (df(alpha1) == 0)
    if (d2f(alpha1) == 0)
        disp('la radice alpha1 ha molteplicita'' maggiore di due')
    else
        disp('la radice alpha1 ha molteplicita uguale a due')
    end
else
    disp('la radice alpha1 ha molteplicita'' uguale ad uno')
end
```

e otteniamo la seguente stampa a schermo:

```
 la radice alpha1 ha molteplicita uguale a due
```

2. Una possibile implementazione della funzione richiesta è la seguente:

```
function [xvect,it]=newton(x0,nmax,toll,fun,dfun, mol)

% [xvect,it]=newton(x0,nmax,toll,fun,dfun)
%
% Metodo di Newton per la ricerca degli zeri della
% funzione fun. Test d'arresto basato sul controllo
% della differenza tra due iterate successive.
%
% Parametri di ingresso:
%
% x0          Punto di partenza
% nmax        Numero massimo di iterazioni
```

```
% toll       Tolleranza sul test d'arresto
% fun dfun   inline functions contenenti la funzione e la sua derivata
% mol        Se presente, permette di effettuare il metodo di Newton
%            modificato
%
% Parametri di uscita:
%
% xvect      Vett. contenente tutte le iterate calcolate
%            (l'ultima componente e' la soluzione)
% it         Iterazioni effettuate

if (nargin == 5)
    mol = 1;
end

err = toll+1;
it = 0;
xvect = x0;
xv = x0;

while (it< nmax && err> toll)
   dfx = dfun(xv);
   if dfx == 0
      error(' Arresto per azzeramento di dfun');
   else
      xn = xv - mol*fun(xv)/dfx;
      err = abs(xn-xv);
      xvect = [xvect; xn];
      it = it+1;
      xv = xn;
   end
end

if (it < nmax)
    fprintf(' Convergenza al passo k : %d \n',it);
else
    fprintf(' E` stato raggiunto il numero massimo di passi k : %d \n',it);
end
fprintf(' Radice calcolata       : %-12.8f \n', xvect(end));
```

3. Con i seguenti comandi si ottengono le operazioni richeste:

```
toll = 1e-6;
nmax = 100;

disp('Calcolo della radice con molteplicita'' 2')
x01 = 0.5;
disp('Metodo di Newton semplice');
[xvect_1,it_1]=newton(x01,nmax,toll,f,df);
disp('Metodo di Newton modificato');
[xvect_1m,it_1m]=newton(x01,nmax,toll,f,df,2);


disp('Calcolo della prima radice con molteplicita'' 1')
x02 = 3;
[xvect_2,it_2]=newton(x02,nmax,toll,f,df);
```

```
disp('Calcolo della seconda radice con molteplicita'' 1')
x03 = 6;
[xvect_3,it_3]=newton(x03,nmax,toll,f,df);

alpha1 = 1; % soluzione esatta

% valutazione errore con Newton e Newton modificato
sol_1 = alpha1*ones(length(xvect_1),1);
sol_1m = alpha1*ones(length(xvect_1m),1);

err_1 = abs(xvect_1 - sol_1);
err_1m = abs(xvect_1m - sol_1m);

it_1vec = 0:it_1;
it_1mvec = 0:it_1m;

figure(3);
semilogy(it_1vec,err_1,'b', it_1mvec,err_1m,'r');
grid on
title('Confronto convergenza metodo Newton e Newton modificato')
legend('Newton','Newton modificato','Location', 'NorthEast');
xlabel('iterazioni');
ylabel('errore');
```

Il precedente codice produce a schermo:

```
Calcolo della radice con molteplicita' 2
Metodo di Newton semplice
 Convergenza al passo k : 20
 Radice calcolata       : 0.99999942
Metodo di Newton modificato
 Convergenza al passo k : 4
 Radice calcolata       : 1.00000000

Calcolo della prima radice con molteplicita' 1
 Convergenza al passo k : 6
 Radice calcolata       : 3.55780416

Calcolo della seconda radice con molteplicita' 1
 Convergenza al passo k : 6
 Radice calcolata       : 6.34045080
```

Il grafico che mostra il confronto tra la convergenza del metodo di Newton classico e quello modificato, per il calcolo della radice doppia α_2 è mostrato in Figura 3.

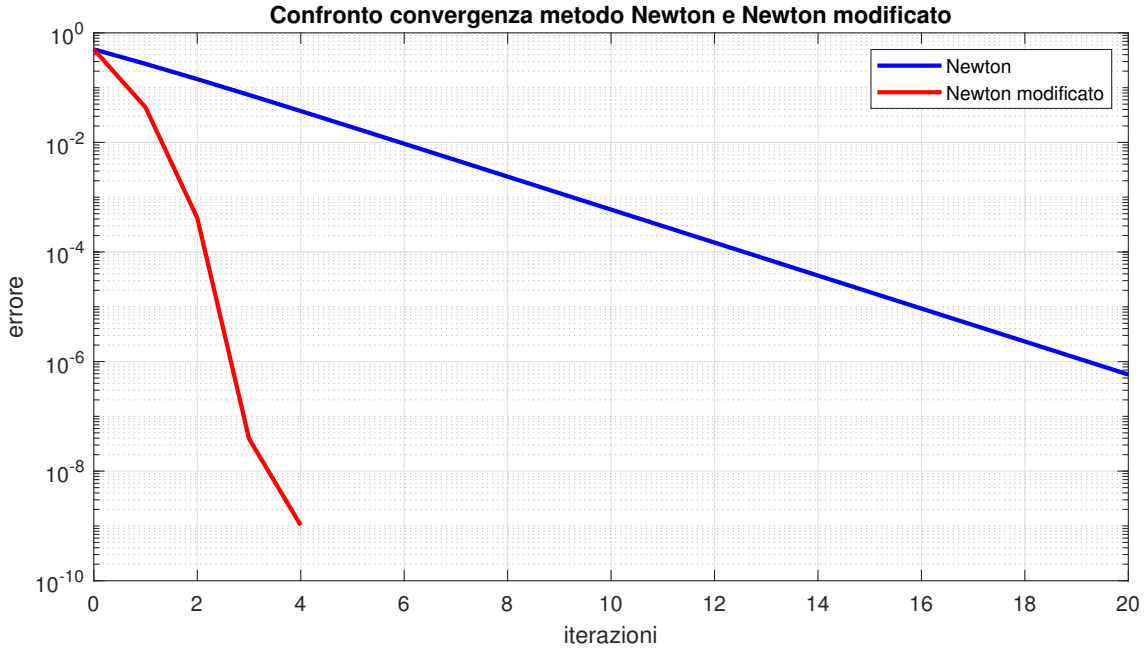

Figura 3: Esercizio 2: confronto tra la convergenza del metodo di Newton classico e quello modificato per α_2

Esercizio 3

Supponiamo di inserire tutti i seguenti comandi, necessari per risolvere l'esercizio, in uno script di Matlab®.

1. La definizione della funzione e il grafico richiesto si ottengono con i seguenti comandi:

```
a = -1;  b = 6;
x = linspace( a, b, 1000 );
f = @(x) atan(7*( x - pi/2)) + sin((x-pi/2).^3);
y = f(x);
y0 = zeros(length(x),1);
figure(1);
plot(x,y,'b',x,y0,'r')
title('f(x) = atan(7*(x - pi/2)) + sin((x-pi/2)^3)')
xlabel('x')
ylabel('y')
legend('y = f(x)', 'y = 0', 'Location', 'SouthEast')
grid on;
```

   Il grafico della funzione $f(x)$ è mostrato in Figura 4.

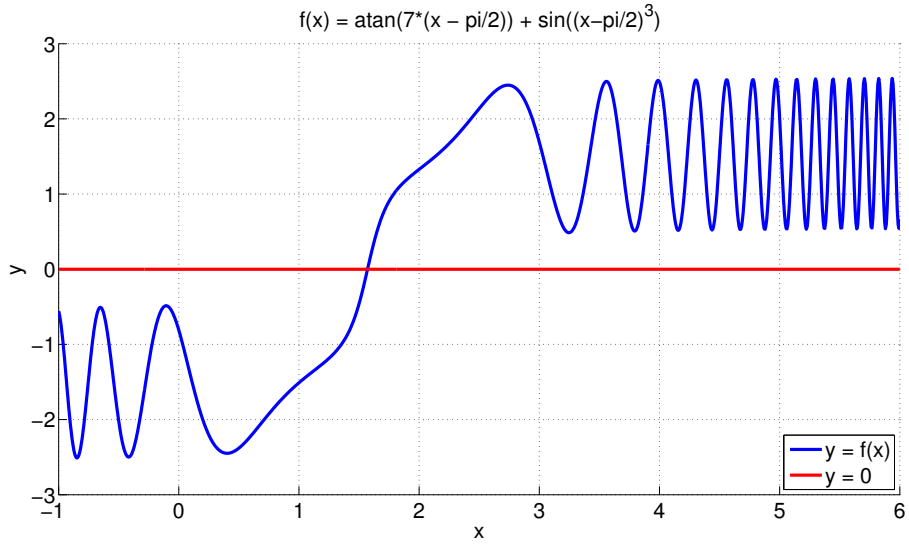

Figura 4: Grafico della funzione $f(x)$ in blu e della retta $y = 0$ in rosso.

2. Calcolo della derivata prima di $f(x)$ e verifica che lo zero α sia semplice:

```
alpha = pi/2; % valore esatto dello zero di f(x)

df = @(x) 7 ./ ( 1 + 49 * ( x-pi/2 ).^2 ) + 3 * (x-pi/2).^2 .* cos( (x-pi/2).^3 );

% controllo che alpha = pi/2 sia uno zero semplice
if (df(alpha) ~= 0)
    disp('lo zero alpha = pi/2  e'' semplice')
end
```

a schermo otteniamo la stampa:

```
 Lo zero alpha = pi/2  e' semplice
```

Calcolo dello zero α con il metodo di Newton, nei casi richiesti:

```
nmax = 1000;
toll = 1e-10;

disp('Metodo di Newton con X0 = 1.5')
x0 = 1.5;
[xvect,it] = newton(x0,nmax,toll,f,df);
err = abs(xvect(end)-alpha)

disp('Metodo di Newton con X0 = 4')
x0 = 4;
[xvect,it] = newton(x0,nmax,toll,f,df);
err = abs(xvect(end)-alpha)
```

a schermo otteniamo la stampa dei risultati:

```
Metodo di Newton con X0 = 1.5
Convergenza al passo k : 4
Radice calcolata       : 1.57079633

err =

     0

Metodo di Newton con X0 = 4
E` stato raggiunto il numero massimo di passi k : 1000
Radice calcolata       : 23.95331549

err =

   22.3825
```

Si osservi che nel secondo caso, l'uso di $x^{(0)} = 4$ non permette al metodo di Newton di giungere a convergenza e il metodo si arresta quando viene raggiunto il numero massimo di iterazioni. Questo avviene perché la convergenza del metodo di Newton è garantita solo qualora il punto di partenza $x^{(0)}$ sia sufficientemente vicino allo zero cercato.

3. Verifichiamo che il metodo di bisezione può essere utilizzato, e in caso affermativo calcoliamo α con i seguenti comandi:

```
 if (f(a)*f(b)) < 0
    disp('Si puo'' applicare il metodo di bisezione');
    nmax_b = 1000;
    toll_b = ( b - a ) / (2^31);
    [xvect,it] = bisez(a,b,toll_b,f);
    err = abs(xvect(end)-alpha)
else
    disp('Non si puo'' applicare il metodo di bisezione');
end
```

a schermo otteniamo la stampa:

```
Si puo' applicare il metodo di bisezione
Massimo numero di iterazioni ammissibili 30
x_29 soddisfa la tolleranza sul residuo
 Radice calcolata       : 1.57079633

err =

   6.0771e-11
```

4. Una possibile implementazione della funzione richiesta è la seguente:

```
function [xvect,it]=biseznewton(a,b,nmax_b,nmax_n,toll,fun,dfun)

% [xvect,it]=biseznewton(a,b,nmax_b,nmax_n,toll,fun,dfun)
%
% Ricerca dello zero di f(x) nell'intervallo [a,b],
% utilizzando il metodo di bisezione per l'avvicinamento allo zero
```

```
% e successivamente il metodo di Newton
%
% Parametri di ingresso:
% a, b       Estremi intervallo di partenza
% nmax_b     Numero di iterazioni per il metodo di Bisezione
% nmax_n     Numero massimo di iterazioni per il metodo di Newton
% toll_n     Tolleranza sul test d'arresto il metodo di Newton
% fun dfun   Macro contenenti la funzione e la sua derivata
%
% Parametri di uscita:
% xvect      Vettore contenente tutte le iterate calcolate
%            (l'ultima componente e' la soluzione)
% it         Iterazioni totali (bisezione + Newton) effettuate

xvect = [];
it = [];

disp('----------------- mi avvicino allo zero con bisezione ---------------')

% Metodo di Bisezione
toll_b = ( b - a ) / ( 2^( nmax_b + 1) );
[ xvect_b,  it_b ] = bisez( a, b, toll_b, fun );
it = it_b;
xvect = [ xvect_b ];

disp('----------------- lancio il metodo di Newton ---------------')

% Metodo di Newton
xv = xvect( end );
[ xvect_n, it_n ] = newton( xv, nmax_n, toll, fun, dfun );
it = it + it_n;

xvect = [ xvect; xvect_n( 2 : end ) ]; % xvect_b(end) e xvect_n(1) sono coincidenti
```

5. Il calcolo di α utilizzando `biseznewton` si ottiene tramite i seguenti comandi:

```
nmax_b = 5;
nmax_n = 1000;
[xvect,it] = biseznewton(a,b,nmax_b,nmax_n,toll,f,df);

disp('L''errore finale e''')
disp(abs(xvect(end)-alpha))
```

e a schermo otteniamo la stampa del risultato:

```
 ----------------- mi avvicino allo zero con bisezione ---------------
 Massimo numero di iterazioni ammissibili 5
 ATTENZIONE ! Massimo numero di iterazioni raggiunte!  Errore sul residuo 3.6872e-01
 Radice calcolata       : 1.51562500
----------------- lancio il metodo di Newton ---------------
 Convergenza al passo k : 4
 Radice calcolata       : 1.57079633

 L'errore finale e'
     0
```

Esercizio 4

Si consideri la funzione $f(x) = \cos^2(2x) - x^2$.

1. Definiamo la *anonymous function* `f = @(x)` e la rappresentiamo graficamente dopo averla valutata su mille punti equispaziati;

```
>> f = @(x) cos(2*x).^2 - x.^2;
>> x = linspace(-pi/2,pi/2,1000);
>> plot(x,f(x),x,zeros(size(x)),'k');
```

   Si individuano graficamente due zeri, simmetrici rispetto all'origine, che chiameremo nel seguito $\alpha > 0$ e $-\alpha < 0$.

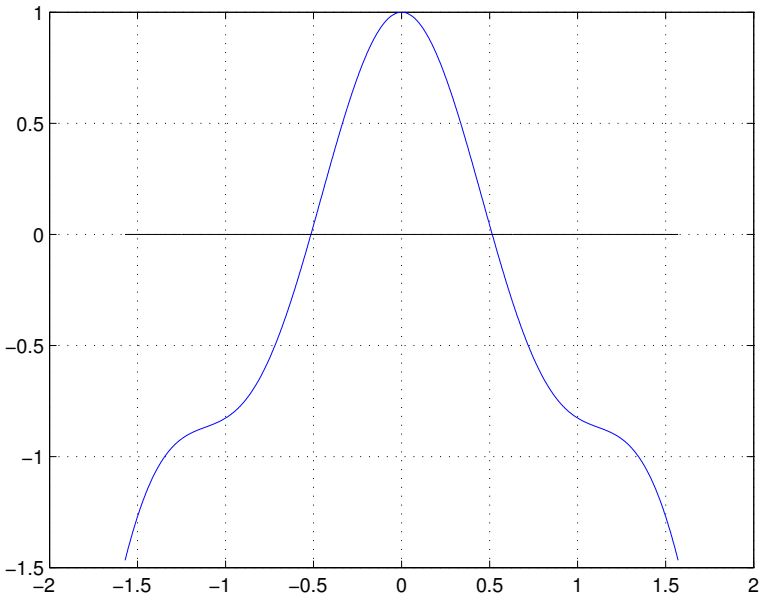

Figura 5: Rappresentazione di $f(x)$ nell'intervallo $[-\pi/2, \pi/2]$.

2. Per il teorema di Ostrowski è richiesto che $|\phi'(\alpha)| < 1$, ovvero:

$$\begin{aligned} |1+Af'(\alpha)| &< 1 \\ -1 < 1+Af'(\alpha) &< 1 \\ -2 < Af'(\alpha) &< 0 \\ 0 < A &< -2/f'(\alpha) \end{aligned}$$

   Si noti che, deducendo da una analisi del grafico della funzione che $f'(\alpha) < 0$, è stato cambiato il segno della disequazione nell'ultimo passaggio.

3. Dobbiamo in questo caso definire la funzione `phi` ed utilizzare la `function ptofis.m` con la seguente sintassi:

```
>> phi = @(x) x + 0.1*(cos(2*x).^2 - x.^2)
>> [succ1,it1] = ptofis(0.1,phi,1000,1e-10,-pi/2,pi/2);
```

dove gli ultimi due argomenti `-pi/2,pi/2` sono gli estremi (su entrambi gli assi cartesiani) dell'intervallo nel quale verrà plottato l'output grafico della funzione (si veda `help ptofis`)

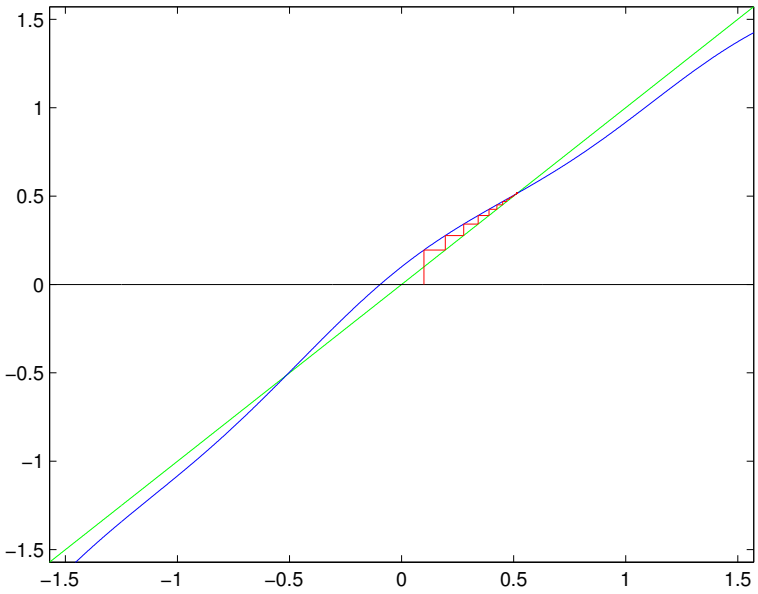

Figura 6: Rappresentazione grafica delle iterate di punto fisso per la funzione dell'Esercizio 4 con $A = 0.1$ e $x^{(0)} = 0.1$. In blu viene rappresentato il grafico della funzione di iterazione $\phi(x)$, in verde la bisettrice degli assi e in rosso le iterate calcolate dall'algoritmo

Il numero di iterazioni effettuate e il valore numerico della soluzione (contenuto nell'ultimo elemento del vettore fornito come output della funzione `succ1`) sono:

```
 Numero di Iterazioni : 65
 Radice calcolata :   0.51493326
```

Valutando $f'(x) = -4\cos(2x)\sin(2x) - 2x$ nello zero calcolato otteniamo l'intervallo di valori ammissibili di A:

```
>> df = @(x) -4*cos(2*x).*sin(2*x)-2*x;
>> df_a = df(succ1(end))
df_a =
   -2.7955
>> Asup = -2/(df_a)
Asup =
```

```
    0.7154
```

Ovvero il metodo converge ad $\alpha > 0$ per $0 < A < 0.7154$. Ad esempio, per $A = 0.6$ e $x^{(0)} = 0.1$ otteniamo il seguente risultato:

```
>> phi6 = @(x) x + 0.6*(cos(2*x).^2 - x.^2);
>> [succ2,it2] = ptofis(0.1,phi6,1000,1e-10,-pi/2, pi/2);
 Numero di Iterazioni : 57
 Radice calcolata :   0.51493326
```

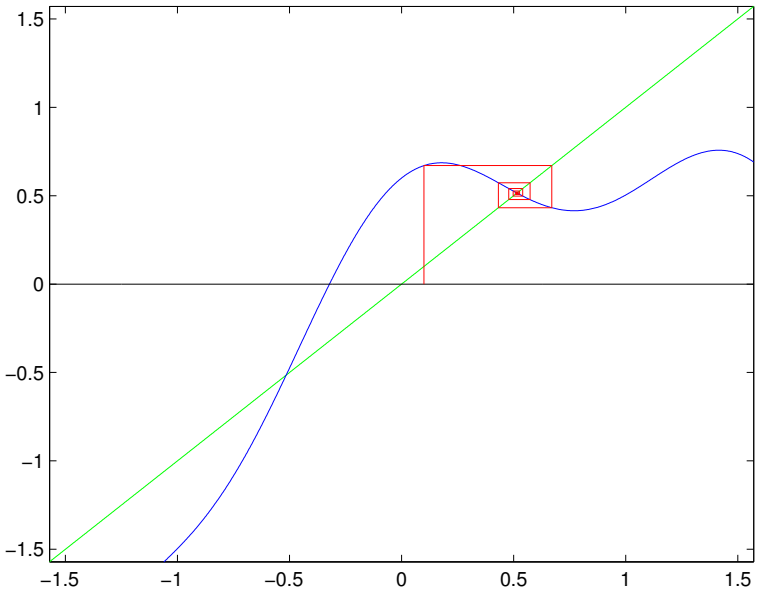

Figura 7: Rappresentazione grafica delle iterate di punto fisso per la funzione dell'Esercizio 4 con $A = 0.6$ e $x^{(0)} = 0.1$

Notiamo che l'algoritmo può convergere per valori del dato iniziale relativamente lontani dalla soluzione, ad esempio per $A = 0.6$ e $x^{(0)} = 2.0$ si ottiene il seguente risultato:

```
>> [succ3,it3] = ptofis(2.0,phi6,1000,1e-10,-pi/2, 2.1);
 Numero di Iterazioni : 58
 Radice calcolata :   0.51493326
```

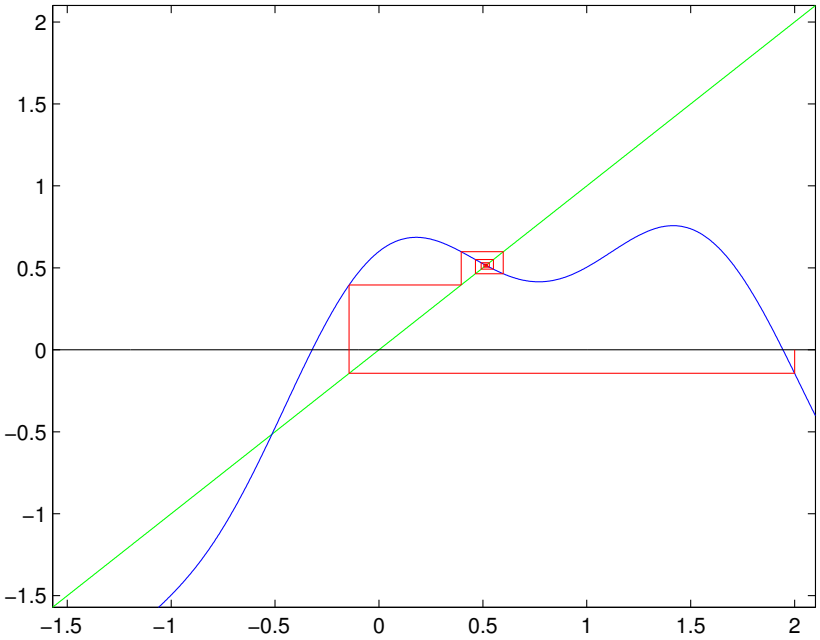

Figura 8: Rappresentazione grafica delle iterate di punto fisso per la funzione dell'Esercizio 4 con $A = 0.6$ e $x^{(0)} = 2.0$.

Per $A = 0.75 > 0.7154$, l'algoritmo di punto fisso non converge, non essendo verificata l'ipotesi del teorema di Ostrowski:

```
>> phi75 = @(x) x + 0.75*(cos(2*x).^2 - x.^2);
>> [succ3,it3] = ptofis(0.1,phi75,1000,1e-10,-pi/2, pi/2);
 Numero massimo di iterazioni raggiunto
```

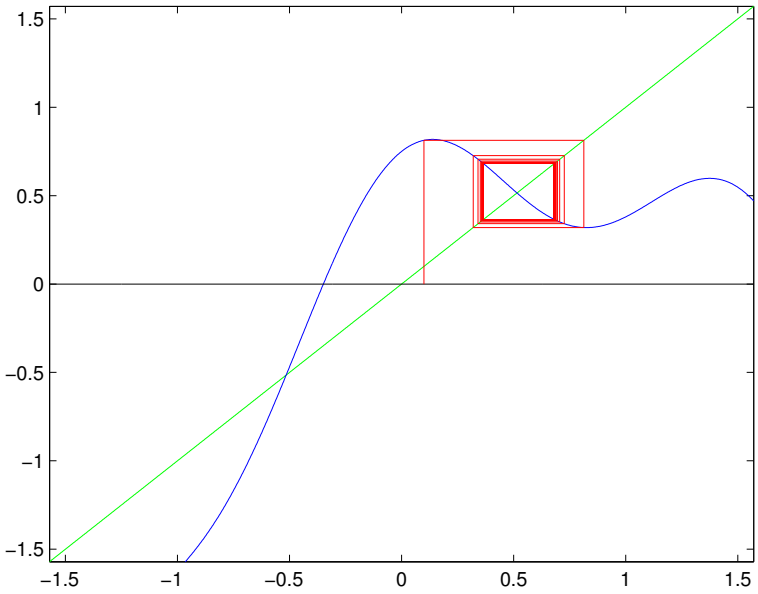

Figura 9: Rappresentazione grafica delle iterate di punto fisso per la funzione dell'Esercizio 4 con $A = 0.75$ e $x^{(0)} = 0.1$; caso in cui l'algoritmo non giunge a convergenza.

4. Dalla stima teorica, per i metodi di punto fisso di ordine 1 il fattore di convergenza è uguale al valore di $\phi'(\alpha)$. Invochiamo la `function stimap.m` (che riceve in ingresso il vettore di valori ottenuti dalle successive iterazioni di un metodo iterativo) per verificare ordine e fattore di convergenza per le successioni ottenute con $A = 0.1$ e $A = 0.6$:

```
>> [p1,c1] = stimap(succ1);
 Ordine stimato       :   1.00000183
 Fattore di riduzione :   0.72047642
>> [p2,c2] = stimap(succ2);
 Ordine stimato       :   0.99999880
 Fattore di riduzione :   0.67730133
```

La convergenza è sempre lineare (ordine uguale a 1), mentre i fattori sono in perfetto accordo con la stima teorica (si noti che per $A = 0.6$ si ha $1 + 0.6f'(\alpha) < 0$ e la convergenza è oscillante, come si vede nell'output grafico in Figura 7):

```
>> 1 + 0.1*df_a
ans =
    0.7204
>> 1 + 0.6*df_a
ans =
   -0.6773
```

5. Si ha un metodo del secondo ordine se $\phi'(\alpha) = 0$, ovvero se $1 + Af'(\alpha) = 0$, e quindi:

$$A = -\frac{1}{f'(\alpha)} \simeq 0.3577$$

Verifichiamo di ottenere effettivamente un metodo del secondo ordine con i comandi:

```
>> A_opt = -1/df_a
A_opt =
    0.3577
>> phi_opt = @(x) x + A_opt*(cos(2*x).^2 - x.^2);
>> [succ_opt,it_opt] = ptofis(0.1,phi_opt,1000,1e-10,-pi/2, pi/2);
 Numero di Iterazioni : 5
 Radice calcolata :   0.51493326
>> [p_opt,c_opt] = stimap(succ_opt);
 Ordine stimato       :   2.00064302
 Fattore di riduzione :   0.31768451
```

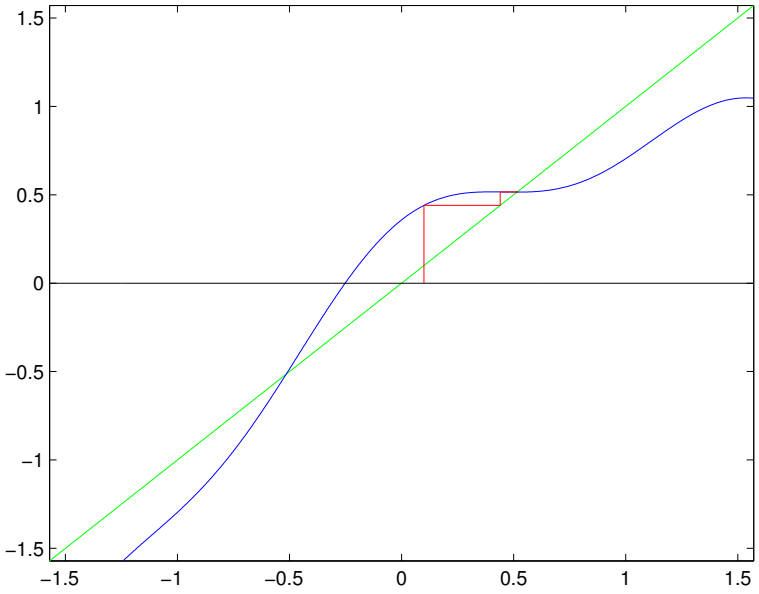

Figura 10: Rappresentazione grafica delle iterate di punto fisso per la funzione dell'Esercizio 4 con $A = A_{opt} = 0.3577$ e $x^{(0)} = 0.1$; caso in cui il metodo ha ordine di convergenza pari a 2.

Il metodo converge in sole 5 iterazioni e la stima dell'ordine di convergenza è in accordo con le considerazioni teoriche.

6. Il metodo di Newton si può rileggere come metodo di punto fisso con funzione di iterazione:

$$\phi_N = x - \frac{f(x)}{f'(x)}$$

In questo caso, la funzione di iterazione è discontinua in $x = 0$ e oscillante altrove. Ciò implica che l'intervallo $[a, b]$ all'interno del quale scegliere il dato iniziale per assicurare convergenza allo zero $\alpha > 0$ è piccolo.

Per stimare questo intervallo sperimentalmente possiamo procedere nella maniera seguente: variamo il valore di `x0` all'interno di un intervallo a piacere; eseguiamo il metodo di Newton a partire da questo dato iniziale e salviamo il valore della soluzione alla quale il metodo converge; infine rappresentiamo su un grafico i valori dello zero così ottenuto in funzione di `x0`. In pratica:

```
phiN = @(x)x - (cos(2*x).^2-x.^2)./(-4*cos(2*x).*sin(2*x)-2*x)
vett_sol=[];
vett_x0 = [0.01:0.01:1]
for x0 = vett_x0
    [succ,it] = ptofis(x0,phiN,1000,1e-10,-pi/2, pi/2);
    vett_sol = [vett_sol succ(end)];
end
plot(vett_x0,vett_sol,'o--')
```

Dalla Figura 11 si ottiene un intervallo per la scelta del dato iniziale pari a $[0.12,\ 0.88]$.

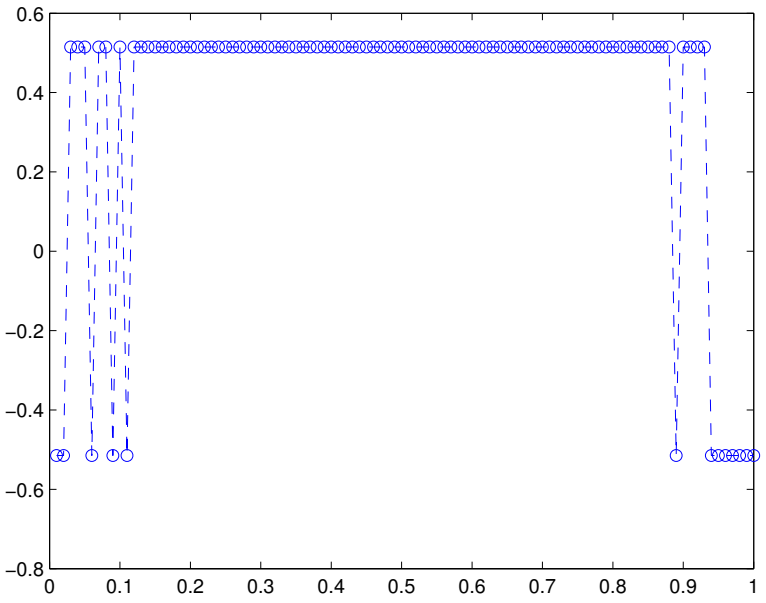

Figura 11: Radice calcolata dal metodo di Newton al variare del dato iniziale. Scegliendo un dato iniziale all'interno dell'intervallo $[0.12,\ 0.88]$ si converge alla radice $\alpha > 0$.

Un risultato del metodo di Newton è il seguente (per $x^{(0)} = 0.8$):

```
>> [succN,itN] = ptofis(0.8,phiN,1000,1e-10,-pi, 2*pi);
 Numero di Iterazioni : 5
 Radice calcolata :   0.51493326
```

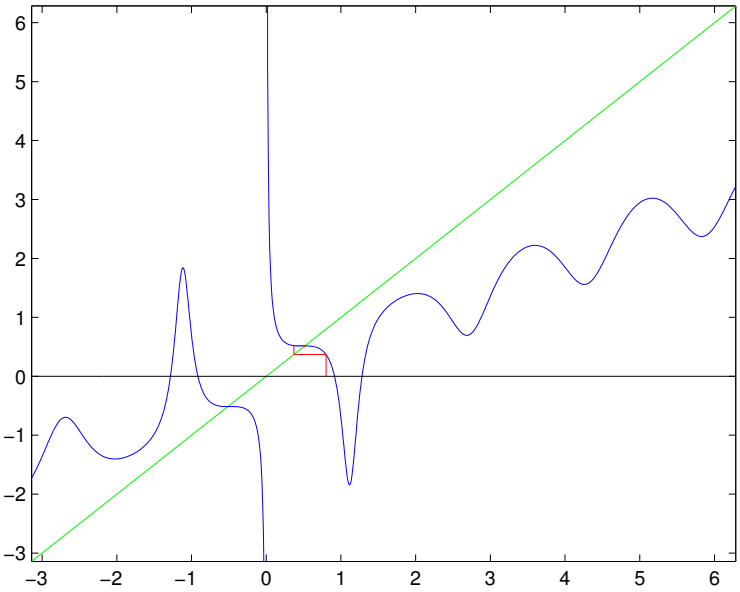

Figura 12: Rappresentazione grafica delle iterate di punto fisso corrispondenti al metodo di Newton. Si noti la forma della funzione di iterazione.