

# Calcolo Numerico

(A.A. 2014-2015)

**Lab n.4, 5, 6, 7**

**Equazioni non lineari: Metodi di Bisezione, Newton-Raphson, Secanti**

**5,12,19,24 Novembre 2014**

# Equazioni non lineari

Un'**equazione non lineare** è un'equazione del tipo

$$f(x) = 0$$

Le **soluzioni**  $\xi$  dell'equazione, cioè quei valori tali che

$$f(\xi) = 0$$

vengono chiamate **radici** dell'equazione non lineare o **zeri** della funzione  $f$ .

Ci limiteremo al caso di **radici reali**:  $\xi \in \mathbf{R}$ .

# Separazione delle radici

In genere, le **equazioni non lineari** che nascono nelle applicazioni non possono essere **risolte analiticamente**. Per **approssimare le radici** è necessario ricorrere a un **metodo numerico**.

**Prima** di utilizzare un metodo numerico bisogna sapere:

- **quante** sono le radici (reali);
- **dove** si trovano approssimativamente;
- se ci sono delle **simmetrie**.

Per rispondere a queste domande si può ricorrere alla **tabulazione** o al **grafico** della funzione  $f$ .

Una volta separata una radice  $\xi$  si passa alla **seconda fase** che consiste nella costruzione di un'opportuna **successione**  $\{x_n\}$  **di approssimazioni** di  $\xi$  che converge alla radice  $\xi$  al divergere di  $n$ .

## Equazioni non lineari

La lunghezza d'onda di uno tsunami  $L$ , per una certa profondità dell'acqua  $d$  soddisfa la seguente **equazione non lineare**

$$L = \frac{a_g T^2}{2\pi} \tanh\left(\frac{2\pi d}{L}\right)$$

con  $a_g$  e  $T$  rispettivamente l'accelerazione di gravità e il periodo. Sapendo che  $T = 2880s$  e  $d = 4000m$  (valore tipico dell'Oceano Indiano), produrre una stima del valore di  $L$  con precisione almeno  $10^{-5}$ .

## Separazione delle radici

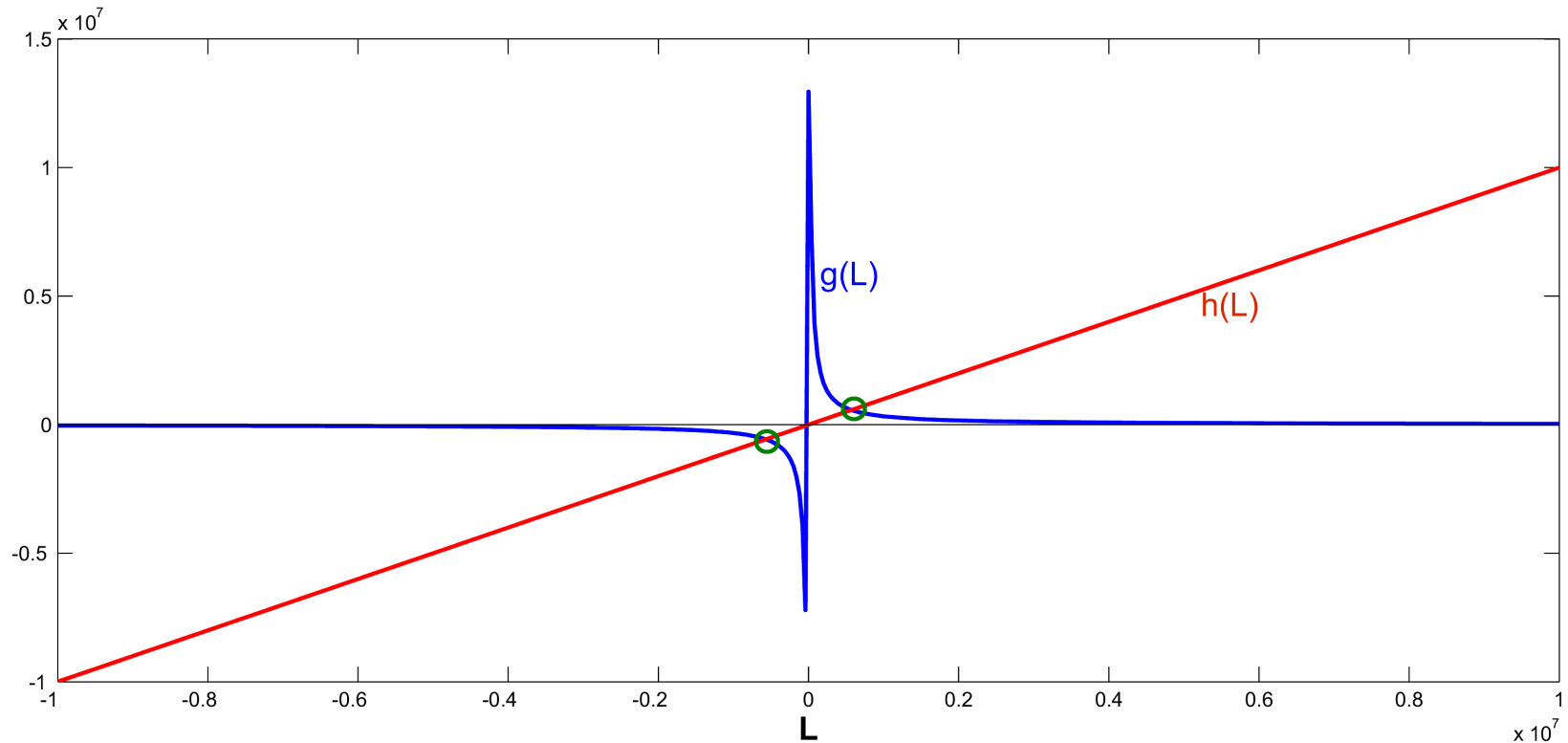
Utilizziamo il **metodo grafico** per stabilire quante sono le radici reali di  $f(L)$ , con  $f(L) = L - \frac{a_g T^2}{2\pi} \tanh\left(\frac{2\pi d}{L}\right)$ .

Poniamo

$$h(L) = L \quad \text{e} \quad g(L) = \frac{a_g T^2}{2\pi} \tanh\left(\frac{2\pi d}{L}\right)$$

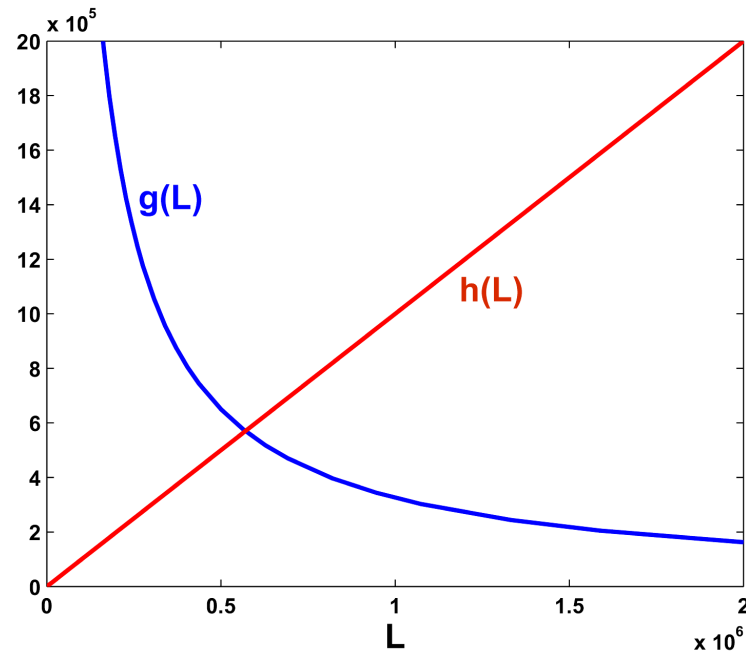
e visualizziamo i punti di intersezione tra le due funzioni.

Dal grafico si evince che le intersezioni sono due, una positiva e una negativa.



Poichè la lunghezza d'onda deve essere un numero positivo, si scarta la radice negativa.

Restringendo il grafico delle funzioni nell'intervallo  $[0, 2 \cdot 10^6]$ , si osserva che la radice positiva è contenuta nell'intervallo  $[.5 \cdot 10^6, 1 \cdot 10^6]$  ( $[500, 1000]Km$ )



```
>> T = 2880;
>> ag = 9.81;
>> h = @(L) [L];
>> g = @(L) [(ag*T^2)/(2*pi)*tanh(2*pi*d/L)];
>> figure, fplot(g,[0 2*10^6])
>> hold on, fplot(h,[0 2*10^6])
>> xlabel('L')
>> axis([0 2000000 -1 2000000])
```

Visualizzando le due funzioni per  $L \in [0.5 \cdot 10^6, 1 \cdot 10^6]$ , si può ridurre ulteriormente l'intervallo in cui cercare la radice dell'equazione non lineare considerata, per esempio  $[0.5 \cdot 10^6, 0.6 \cdot 10^6]$

A questo punto è possibile selezionare un metodo numerico per il calcolo della radice di  $f(L) = 0$  nell'intervallo  $I = [0.5 \cdot 10^6, 0.6 \cdot 10^6]$



# Metodo di bisezione (o metodo dicotomico)

## Ipotesi di applicabilità :

- è stato **separato** un intervallo  $I = [a, b]$  in cui c'è un'**unica radice**  $\xi$ ;
- la funzione  $f$  è **continua** in  $I$ :  $f \in C^0[a, b]$ ;
- $f(a)f(b) < 0$ .

## Algoritmo:

$$a_0 = a, \quad b_0 = b$$

per  $k = 1, 2, 3, \dots$

$$x_k = \frac{a_{k-1} + b_{k-1}}{2} \quad (\text{punto medio di } [a_{k-1}, b_{k-1}])$$

se  $f(x_k) = 0$ , allora stop

se  $f(a_{k-1})f(x_k) < 0$ , allora  $[a_k, b_k] = [a_{k-1}, x_k]$

se  $f(x_k)f(b_{k-1}) < 0$ , allora  $[a_k, b_k] = [x_k, b_{k-1}]$

**Criteri di arresto a posteriori**

$$\begin{cases} |e_k| \simeq |x_k - x_{k-1}| < \varepsilon \\ |f(x_k)| < \varepsilon \end{cases}$$

**Criterio di arresto a priori:** La **stima a priori** del numero di iterazioni  $K$  necessario per ottenere un **errore minore** di  $\varepsilon$  è

$$|e_k| < \frac{b-a}{2^k} < \varepsilon \quad \Rightarrow \quad K > \frac{\log(b-a) - \log(\varepsilon)}{\log 2}$$

# Metodo di bisezione

Si verifica facilmente che la funzione  $f(L) = L - \frac{a_g T^2}{2\pi} \tanh\left(\frac{2\pi d}{L}\right)$  è una funzione continua in tutto il dominio di definizione ed in particolare nell'intervallo  $I = [a, b] = [0.5 \cdot 10^6, 0.6 \cdot 10^6]$ . Inoltre risulta

$$f(0.5 \cdot 10^6) = -150396.83597 \quad \text{e} \quad f(0.6 \cdot 10^6) = 57863.27995$$

(Oss.  $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ )

da cui

$$f(a)f(b) < 0.$$

Quindi, sono soddisfatte le condizioni di applicabilità del **metodo di bisezione**

Si calcola il punto

$$x_1 = \frac{a + b}{2} = \frac{0.5 \cdot 10^6 + 0.6 \cdot 10^6}{2} = 0.55 \cdot 10^6$$

si valuta la funzione  $f$  nel punto  $x_1$ , cioè  $f(x_1) = -41356.18896 < 0$

quindi, si definisce il nuovo intervallo

$$I_1 = [x_1, b] = [0.55 \cdot 10^6, 0.6 \cdot 10^6]$$

che contiene la radice positiva di  $f$ .

L'errore è

$$e_1 = |x_1 - x_0| = |x_1 - a| = 50000.$$

Si calcola il punto  $x_2 = \frac{x_1+b}{2} = \frac{0.55 \cdot 10^6 + 0.6 \cdot 10^6}{2} = 0.575 \cdot 10^6$

si valuta la funzione  $f$  nel punto  $x_2$ , cioè  $f(x_2) = 9321.48847 > 0$ .

Il nuovo intervallo è  $I_2 = [x_1, x_2] = [0.55 \cdot 10^6, 0.575 \cdot 10^6]$

mentre l'errore diventa

$$e_2 = |x_2 - x_1| = 25000.$$

Procedendo in questo modo si ha

$$x_3 = \frac{x_1 + x_2}{2} = \frac{0.55 \cdot 10^6 + 0.575 \cdot 10^6}{2} = 0.5625 \cdot 10^6$$

$$f(x_3) = -15732.61210 < 0,$$

$$e_3 = |x_3 - x_2| = |0.5625 \cdot 10^6 - 0.575 \cdot 10^6| = 12500$$

e  $I_3 = [x_3, x_2] = [0.5625 \cdot 10^6, 0.575 \cdot 10^6]$

$$x_4 = \frac{x_3 + x_2}{2} = \frac{0.5625 \cdot 10^6 + 0.575 \cdot 10^6}{2} = 0.56875 \cdot 10^6$$

$$f(x_4) = -3136.71786 < 0,$$

$$e_4 = |x_4 - x_3| = |0.56875 \cdot 10^6 - 0.5625 \cdot 10^6| = 6250$$

e  $I_4 = [x_4, x_2]$

e così via.

Si osserva che l'errore  $e_k$  si dimezza ad ogni iterazione, cioè  $\frac{e_{k+1}}{e_k} = \frac{1}{2}$ .

Quindi, richiedendo un errore almeno di

$$\varepsilon = 0.5 \cdot 10^{-5},$$

sono necessarie  $K$  iterazioni con

$$K > \frac{\log(b - a) - \log(\varepsilon)}{\log(2)} = \frac{\log(0.1 \cdot 10^6) - \log(0.5 \cdot 10^{-5})}{\log(2)} \approx 35$$

affinchè il metodo converga alla soluzione con la precisione fissata.



k	estremo inferiore	estremo superiore	$x_k$	$e_k$	$f(x_k)$
1	500000.000000000000	600000.000000000000	550000.000000000000	$50 \cdot 10^3$	41536.18896
2	550000.000000000000	600000.000000000000	575000.000000000000	$25 \cdot 10^3$	9321.48847
3	550000.000000000000	575000.000000000000	562500.000000000000	$12.5 \cdot 10^3$	15732.61210
4	562500.000000000000	575000.000000000000	568750.000000000000	$6.250 \cdot 10^3$	3136.71786
5	568750.000000000000	575000.000000000000	571875.000000000000	$3.125 \cdot 10^3$	3109.31488
6	568750.000000000000	571875.000000000000	570312.500000000000	$1.5625 \cdot 10^3$	9.43440
7	570312.500000000000	571875.000000000000	571093.750000000000	$7.8125 \cdot 10^2$	1551.00265
8	570312.500000000000	571093.750000000000	570703.125000000000	$3.9062 \cdot 10^2$	771.05027
9	570312.500000000000	570703.125000000000	570507.812500000000	$19.5312 \cdot 10^1$	380.87454
10	570312.500000000000	570507.812500000000	570410.156250000000	97.6562	185.73673
11	570312.500000000000	570410.156250000000	570361.328125000000	48.8281	88.15533
12	570312.500000000000	570361.328125000000	570336.914062500000	24.4141	39.36151
13	570312.500000000000	570336.914062500000	570324.707031250000	12.2070	14.96382
14	570312.500000000000	570324.707031250000	570318.603515625000	6.1035	2.76477
15	570312.500000000000	570318.603515625000	570315.551757812500	3.0518	3.33480
16	570315.551757812500	570318.603515625000	570317.077636718750	1.5259	0.28501
17	570317.077636718750	570318.603515625000	570317.840576171875	$7.6 \cdot 10^{-1}$	1.23988
18	570317.077636718750	570317.840576171875	570317.459106445312	$3.8 \cdot 10^{-1}$	0.47744
19	570317.077636718750	570317.459106445312	570317.268371582031	$1.9 \cdot 10^{-1}$	0.09622
20	570317.077636718750	570317.268371582031	570317.173004150391	$9.5 \cdot 10^{-2}$	0.09439
21	570317.173004150391	570317.268371582031	570317.220687866211	$4.8 \cdot 10^{-2}$	0.00091
22	570317.173004150391	570317.220687866211	570317.196846008301	$2.3 \cdot 10^{-2}$	0.04674
23	570317.196846008301	570317.220687866211	570317.208766937261	$1.192 \cdot 10^{-2}$	0.02292
24	570317.208766937261	570317.220687866211	570317.214727401731	$6 \cdot 10^{-3}$	0.01100
25	570317.214727401731	570317.220687866211	570317.217707633971	$3 \cdot 10^{-3}$	0.00505
26	570317.217707633971	570317.220687866211	570317.219197750091	$1.5 \cdot 10^{-3}$	0.00207
27	570317.219197750091	570317.220687866211	570317.219942808151	$7.4 \cdot 10^{-4}$	0.00059
28	570317.219942808151	570317.220687866211	570317.220315337181	$3.7 \cdot 10^{-4}$	0.00017
29	570317.219942808151	570317.220315337181	570317.220129072671	$1.8 \cdot 10^{-4}$	0.00021
30	570317.220129072671	570317.220315337181	570317.220222204921	$9.3 \cdot 10^{-5}$	0.00002
31	570317.220222204921	570317.220315337181	570317.220268771051	$4.7 \cdot 10^{-5}$	0.00007
32	570317.220222204921	570317.220268771051	570317.220245487991	$2.3 \cdot 10^{-5}$	0.00003
33	570317.220222204921	570317.220245487991	570317.220233846461	$1.2 \cdot 10^{-5}$	0.000003
34	570317.220222204921	570317.220233846461	570317.220228025691	$0.6 \cdot 10^{-5}$	0.000009
35	570317.220228025691	570317.220233846461	570317.220230936071	$0.3 \cdot 10^{-5}$	0.000003

Se come criterio di arresto avessimo usato

$$|f(x_k)| \leq \varepsilon,$$

la soluzione prodotta sarebbe quella corrispondente a  $k = 33$  nella tabella precedente.

# Script Matlab: Metodo di Bisezione

Implementiamo l'algoritmo utilizzando il **criterio di arresto a priori**: il numero di iterazioni **K** per ottenere un errore minore di  $\varepsilon$  è

$$K > \frac{\log(b - a) - \log(\varepsilon)}{\log 2}$$

Introduciamo i seguenti valori

- il periodo **T = 2880** e la profondità dell'acqua **d = 4000**
- gli estremi dell'intervallo **I = [0.5·10<sup>6</sup>, 0.6·10<sup>6</sup>]** in cui si vuole cercare la radice
- la precisione  **$\varepsilon = 10^{-5}$**  con cui si vuole cercare la radice

Aprire l'editor di MATLAB e creare un nuovo file .m dal nome

`script_bisezione`

# script\_bisezione

```
ag = 9.81; T= 2880; d=4000;

f = @(L) [L-(ag*T^2)/(2*pi)*tanh(2*pi*d/L)];

% intervallo in cui si vuole cercare la radice
a0 = 0.5*10^6;
b0 = 0.6*10^6;

eps = 10^-5;

n_iter = ceil((log(b0-a0)-log(eps))/log(2));

for n=1:n_iter
    xn = (a0+b0)/2;

    if f(xn)*f(a0)<0
        b0=xn;
```

```
elseif f(xn)*f(b0)<0
    a0=xn;
end
end

fprintf('la radice e'' %5.6f', xn)
```

# Script Matlab: Metodo di Bisezione

Implementiamo un'altra versione dell'algoritmo in cui richiediamo come parametri di input

- il periodo **T** e la profondità dell'acqua **d**
- gli estremi dell'intervallo **I** in cui si vuole cercare la radice
- la precisione  $\varepsilon$  con cui si vuole cercare la radice
- il numero di iterazioni **n\_iter** da effettuare SE non viene specificata la tolleranza  $\varepsilon$

Aprire l'editor di MATLAB e creare un **nuovo file** .m dal nome

`script_bisezione_2`

## script\_bisezione\_2

```
format long;

% parametri relativi alla funzione di cui si vuole cercare una radice
ag = 9.81;
T = input('inserire il periodo T = ');
d = input('inserire la profondita'' dell'' acqua d = ');

% funzione di cui si vuole cercare una radice
f = @(L) [L-(ag*T^2)/(2*pi)*tanh(2*pi*d/L)]

% intervallo in cui si vuole cercare la radice
a0 = input('inserire l''estremo inferiore dell''intervallo a = ');
b0 = input('inserire l''estremo superiore dell''intervallo b = ');
```

```

eps = input('inserire la precisione richiesta per la soluzione, eps = ')

if isempty(eps)
    %se non viene data la precisione,
    %si richiede il numero di iterazioni da eseguire
    n_iter = input('inserire il no. di iterazioni da eseguire, n_iter = ')

    if isempty(n_iter)
        % se non viene dato il numero di iterazioni,
        % si fissa 30 come numero massimo
        n_iter = 30;
    end
else
    % calcola il numero minimo di iterazioni
    % necessarie per raggiungere la precisione richiesta
    n_iter = ceil((log(b0-a0)-log(eps))/log(2));
end

```



```

% inizializzazione per il I passo
a = a0; b = b0; xv = a0;
fprintf('iter\t a_n\t\t\t b_n\t\t\t x_n\t\t |x_n-x_(n-1)|\t\t f(x_n)\n')
figure, hold on, title('errore di troncamento')
% si cerca la soluzione con un numero fissato di iterazioni
for n= 1:n_iter
    xn = (a+b)/2;
    err1 = abs(xn-xv);
    err2 = abs(f(xn));

    if ( f(a)*f(xn) < 0 )
        b = xn;
    elseif ( f(xn)*f(b) < 0 )
        a = xn;
    end
    xv = xn;

    fprintf('%3d\t %15.15f\t %15.15f %15.15f %6.15f\t %6.15f\n',...
        [n a b xn err1 err2])

```

```
    plot(n,err1,'*')  
end  
axis([0 30 0 60000])  
% si osserva che l'errore err1 si dimezza ad ogni iterazione
```

# Output dei dati

Un modo per costruire stringhe di caratteri per l'output di testo è fornito dalle funzioni **fprintf** e **sprintf** mutuata direttamente dal linguaggio C. La sintassi della prima è del tipo

```
fprintf(fid, formato, variabili)
```

dove **formato** è una stringa di testo che tramite l'uso di caratteri speciali indica il tipo di formato dell'output, **variabili** è una lista opzionale di variabili separate da una virgola e che hanno un corrispondente all'interno della stringa formato. Infine **fid** è un identificatore opzionale del file al quale l'output è inviato. Se si omette il fid, le variabili vengono visualizzate nello standard output (*fid* = 1)

## Esempio

```
>> x = 3.56;
>> s = 'ciao';
>> fprintf('un numero decimale %3.2f e una stringa %s\n', x, s)
un numero decimale 3.56 e una stringa ciao
```

Il **formato** è una stringa che contiene i caratteri che si vogliono visualizzare e, nelle posizioni in cui si vuole venga inserito il valore, deve essere indicato uno dei formati preceduti dal carattere `%`. Tali codici di formati sono abitualmente seguiti da due interi separati da un punto (ad esempio `%3.2f`). Il primo numero indica quante colonne si desiderano impegnare in uscita ed il secondo il numero di cifre della **parte frazionaria**. In questo esempio il formato `% 3.2 f` serve a visualizzare un numero decimale, mentre il formato `%s` è utilizzato per le stringhe.

**Descrittori di formato:** specificano tipo, allineamento, cifre significative, ampiezza di campo.

<b>formato</b>	<b>azione</b>
%s	formato stringa
%d	formato numero intero
%f	formato numero decimale
%e	formato in notazione scientifica
%g	formato in forma compatta usando %f o %e

### Esempi:

**%10s** visualizza una stringa di 10 caratteri

**%6.2f** visualizza un numero con 6 cifre di cui 2 decimali

**%-5d** visualizza un intero con 5 cifre allineato a sinistra

Nell'help di MATLAB si trovano tutti i formati per i vari tipi di variabili.

La funzione **sprintf** ha invece la sintassi

```
stringa = sprintf(formato, variabili)
```

e la differenza è che l'output viene reindirizzato su una stringa

```
>>x=3; s='ciao';  
>>stringa = sprintf('un intero %d e una stringa %s\n', x, s)  
>>disp(stringa)  
un intero 3 e una stringa ciao
```

## Caratteri speciali

`\n` -> carattere di ritorno a capo

`\t` -> carattere di tabulazione

**Nota:** La principale differenza tra le funzioni MATLAB **fprintf** e **sprintf** e le equivalenti versioni in C, è data dalla possibilità di dare come argomenti **vettori e matrici**, come vedremo nel prossimo esempio.

# Esempio

```
>> x=[1 4 2 5 3 6];  
>> fprintf('%3d',x)  
  1  4  2  5  3  6  
>> fprintf('%3d\n%3d\n',x(1:2:end),x(2:2:end))  
  1  
  2  
  3  
  4  
  5  
  6  
>> fprintf('%3d%3d\n',x(1:2:end),x(2:2:end))  
  1  2  
  3  4  
  5  6
```



# Esercizi

- Quale comando si deve usare per ottenere la seguente stampa?

```
x=1.00    |    log(x)=0.00  
x=1.25    |    log(x)=0.22  
x=1.50    |    log(x)=0.41  
x=1.75    |    log(x)=0.56  
x=2.00    |    log(x)=0.69
```

- E se volessimo memorizzare i risultati dello script di bisezione su un file di testo?

# Scrittura su file

Per scrivere su file un insieme di dati di output con un certo formato si utilizzano i comandi **fopen**, **fprintf** e **fclose**:

```
fid = fopen('nomefile','w')
```

dove **fid** è una variabile che identifica il file, **nomefile** definisce il nome del file che è aperto in scrittura (parametro **'w'**)

```
fprintf(fid,formato,variabili)
```

```
fclose(fid)
```

**fclose** chiude il file identificato da fid.

# Script MATLAB: Metodo di Bisezione

usando il ciclo **while**

·  
·  
·

```
eps = input('inserire la precisione richiesta per la soluzione, eps = ');  
if isempty(eps)  
    % se non viene data la precisione,  
    % si fissa eps = .5*10^-5  
    eps = .5*10^-5;  
end  
  
% I passo  
xn = (a0+b0)/2;  
err1 = abs(xn-a0); err2 = abs(f(xn));  
a = a0; b = b0; iter = 0;  
iter = 1;
```

```

% si cerca la soluzione con una certa precisione
while (err1>eps)
    if ( f(a)*f(xn) < 0 )
        b = xn;
    elseif ( f(xn)*f(b) < 0 )
        a = xn;
    end
    xv = xn;
    xn = (a+b)/2;
    iter = iter+1;
    err1 = abs(xn-xv);
    err2 = abs(f(xn));
    fprintf('%3d\t %15.15f\t %15.15f\t %15.15f\t %6.15f\t %6.15f\n', ...
        [iter a b xn err1 err2])
end

```

# Script MATLAB: Metodo di Bisezione

usando un ciclo **while** con doppio criterio di arresto

.  
. .  
.

```
eps = input('inserire la precisione richiesta per la soluzione, eps = ');
if isempty(eps)
    % se non viene data la precisione,
    % si fissa eps = .5*10^-5
    eps = .5*10^-5;
end

% I passo
xn = (a0+b0)/2;
err1 = abs(xn-a0); err2 = abs(f(xn));
a = a0; b = b0; iter=1;
```

```

% si cerca la soluzione tale che l'errore tra due approssimazioni
% successive abbia una certa precisione oppure che la funzione
% assuma un valore abbastanza prossimo a 0
figure, hold on
fprintf('iter\t a_n\t b_n\t ,x_n\t e_n=|x_n-x_(n-1))|\t f(x_n)\n')
while (err1>eps) & (err2>eps)
    if ( f(a)*f(xn) < 0 )
        b = xn;
    elseif ( f(xn)*f(b) < 0 )
        a = xn;
    end
    xv = xn;
    xn = (a+b)/2;
    iter = iter+1;
    err1 = abs(xn-xv);
    err2 = abs(f(xn));
    fprintf('%3d\t %15.15f\t %15.15f %15.15f %6.15f\f    %6.15f\n',...
        [iter a b xn err1 err2])
    % costruisce il grafico dell'errore iterazione dopo iterazione
    plot(iter,err1,'*')
end

```

# Function files

- Matlab permette di definire **nuove funzioni**.
- I comandi che definiscono la funzione devono essere inseriti in un file con estensione **.m**
- La prima linea del file definisce la **sintassi** della funzione

**function [outputs]=myfunc(inputs)**

**Nota.** Il **nome** con cui viene richiamata la funzione dal **Command Window** è quello del **file** in cui è memorizzata. Per questo è consigliato chiamare il file in cui si memorizza la funzione con il nome della funzione

**function [outputs]= myfunc(inputs)**

**outputs** = elenco delle variabili che si richiedono in output **separate da virgole**. Se non si richiede alcuna variabile di output le parentesi [] prima del comando function si possono omettere

**inputs** = elenco delle variabili di input **separate da virgole**

**myfunc** = nome della funzione



# Function files

- Le **variabili di input e di output** possono essere variabili semplici, vettori, matrici o altre strutture dati.
- Il **nome** della funzione deve sempre iniziare con una lettera dell'alfabeto, può contenere numeri e simboli ma non può contenere spazi.
- Una funzione può essere chiamata dal **Command window** oppure all'interno di uno **script** o di un'altra **funzione**.
- la directory in cui è stata salvata la funzione deve essere inclusa nel **path** a meno che la **current directory** non sia la cartella che contiene la funzione

- Una funzione può contenere un' altra funzione (**sotto-funzioni**), mentre uno script può solo richiamare funzioni memorizzate in altri file con estensione **.m**. Una sotto-funzione può essere richiamata solo dalla funzione all'interno della quale è definita.
- Dopo la prima riga di comando, si possono scrivere dei **commenti** alla funzione. Per esempio, si può descrivere cosa fa la funzione, le variabili di input richieste e quelle di output restituite. Tali commenti vanno preceduti ad ogni riga dal simbolo **%** e vengono visualizzati come **help** della funzione

# Esempio

Definire la funzione **stat** che calcoli la media e la deviazione standard di un vettore **x**.

File **stat.m** (nome del file in cui si memorizza la funzione)

---

```
function [mean,stdev]=stat(x)
```

```
% Calcola la media (mean) e la deviazione standard (stdev) degli
```

```
% elementi del vettore x
```

```
n=length(x);
```

```
mean = sum(x)/n;
```

```
stdev = sqrt(sum((x-mean).^2)/n);
```

---

Dal **Command Window**

```
>>x=rand(50,1);
```

```
>>[mean,stdev]=stat(x);
```

**mean** e **stdev** sono ora due variabili contenute nel **Workspace**.

La funzione **stat** può essere richiamata da un altro script o funzione.

# Esempio

E' buona norma **controllare le variabili in input**. Ad esempio, nella funzione **stat** dovremmo assicurarci che **x** sia un vettore

```
function [mean, stdev]=stat(x)
% Calcola la media (mean) e la deviazione standard (stdev) degli
% elementi del vettore x

[m, n] = size(x);
if m>1 & n>1
    error('x deve essere un vettore')
end
mean = sum(x)/n;
stdev = sqrt(sum((x-mean).^2)/n);
```

# Esercizio 1

Trasformare lo **script** relativo al metodo di Bisezione in una **funzione** avente i seguenti parametri di input e output

```
function [xn,err1,err2,iter] = bisezione(f,a,b,eps)
```

Input	Ouput
<b>f</b> funzione	<b>xn</b> la radice trovata
<b>a,b</b> estremi dell'intervallo	<b>err1,err2</b> gli errori effettuati ad ogni passo
<b>eps</b> tolleranza desiderata	<b>iter</b> il numero di iterazioni effettuate

- usare un **ciclo while** con doppio criterio di arresto: il ciclo continua fin quando uno dei due errori (err1, err2) non raggiunge la precisione richiesta
- scrivere uno script che richiami la funzione **bisezione** implementata.

# Metodo di Newton-Raphson: algoritmo

Ad ogni **iterazione**  $k = 1, 2, \dots$  la **nuova approssimazione**  $x_k$  è data dall'**intersezione** tra la **retta**  $t_{k-1}$ , **tangente** a  $f(x)$  nel punto  $(x_{k-1}, f(x_{k-1}))$  e  $y = 0$ .

$$t_{k-1} \rightarrow y = f(x_{k-1}) + f'(x_{k-1})(x - x_{k-1})$$

$$f(x_{k-1}) + f'(x_{k-1})(x_k - x_{k-1}) = 0$$



**Algoritmo:**

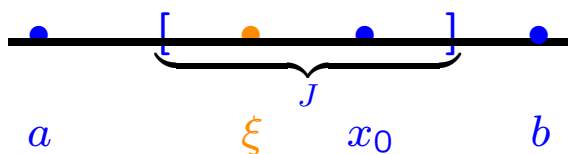
$$\begin{cases} x_0 & \text{dato} \\ x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}, & k = 1, 2, \dots \end{cases}$$

# Metodo di Newton-Raphson: convergenza

Ipotesi di applicabilità :

- è stato **separato** un intervallo  $I = [a, b]$  in cui c'è un'**unica radice**  $\xi$ ;
- $f, f', f''$  sono **continue** in  $I$ :  $f \in C^2[a, b]$ ;
- $f'(x) \neq 0$  per  $x \in [a, b]$ .

$\Rightarrow$  esiste un **intorno**  $J \subseteq I$  di  $\xi$  tale che, se  $x_0 \in J$ , la **successione delle approssimazioni**  $\{x_k\}$  **converge** a  $\xi$ .



Se  $f(x) \in C^3[a, b]$  la convergenza è almeno **quadratica**



# Metodo di Newton-Raphson

$$f(L) = L - \frac{a_g T^2}{2\pi} \tanh\left(\frac{2\pi d}{L}\right)$$

Si osserva che

- $f'(L) = 1 + \frac{a_g T^2 d}{L^2} \frac{1}{\cosh^2\left(\frac{2\pi d}{L}\right)} \neq 0 \quad \forall L \neq 0.$

Inoltre  $f'$  è una funzione continua nello stesso dominio ed in particolare nell'intervallo  $I$  in cui è stata isolata la radice positiva di  $f$ .

- $f''(L) = a_g T^2 d \left( -\frac{2}{L^3 \cosh^2\left(\frac{2\pi d}{L}\right)} + \frac{2\pi d}{L^4} \frac{\sinh\left(\frac{4\pi d}{L}\right)}{\cosh^4\left(\frac{2\pi d}{L}\right)} \right)$

è una funzione continua nel dominio di esistenza di  $f$ .

Inoltre è strettamente negativa nell'intervallo  $I$ .

Scegliamo  $x_0 = b = 0.6 \cdot 10^6$  come approssimazione iniziale della soluzione.

Calcoliamo ora il punto  $x_1$  tale che

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

cioè

$$x_1 = 0.6 \cdot 10^6 - \frac{57863.27995127568}{1.90250514141390} = 569585.74319106806.$$

L'errore  $e_1 = |x_1 - x_0| = 30414.25681 > 0.5 \cdot 10^{-5}$ .

Calcoliamo ora il punto  $x_2$  tale che

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

cioè

$$x_2 = 569585.74319106806 - \frac{-1462.944269931060}{2.001268296457646} = 570316.7517582455.$$

L'errore  $e_2 = |x_2 - x_1| = 731.008567177457730 > 0.5 \cdot 10^{-5}$ .

L'approssimazione ottenuta non ha ancora la precisione richiesta, quindi calcoliamo il punto  $x_3$  tale che

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

cioè

$$x_3 = 570316.7517582455 - \frac{-0.93634280480910}{1.99870815060183} = 570317.2202322469.$$

L'errore  $e_3 = |x_3 - x_2| = 0.468474001390859 > 0.5 \cdot 10^{-5}$ .

$$x_4 = x_3 - \frac{f(x_3)}{f'(x_3)}$$

cioè

$$\begin{aligned} x_4 &= 570317.2202322469 - \frac{-3.834720700979233 \cdot 10^{-7}}{1.998706513054914} = \\ &= 570317.220232438760000. \end{aligned}$$

L'errore  $e_4 = |x_4 - x_3| = 1.918601193287788 \cdot 10^{-7} < 0.5 \cdot 10^{-5}$ .

Riassumendo

$k$	$x_k$	$e_k$	$f(x_k)$
1	569585.74319106806	30414.256808931939	1462.9442699310603
2	570316.75175824552	731.00856717745773	0.936342804809101
3	570317.22023224691	0.4684740013908590	0.000000383472070
4	570317.22023243876	0.0000001918524500	0.0000000000000000

Dopo 4 iterazioni è stata raggiunta la precisione  $\varepsilon$  richiesta per l'errore!

Dopo 3 iterazioni il valore di  $f$  nell'approssimazione alla terza iterazione è molto più piccolo di  $\varepsilon$ !

# Metodo di Newton-Raphson: convergenza

Ipotesi di applicabilità :

- $f(a)f(b) < 0$
- $f, f', f''$  sono **continue** in  $I$ :  $f \in C^2[a, b]$ ;
- $f'(x) \neq 0$  per  $x \in [a, b]$ ;
- $f''(x) \neq 0$  per  $x \in [a, b]$  e  $x_0$  è l'**estremo di Fourier** di  $[a, b]$ .

⇒

- 1) esiste un'**unica radice**  $\xi \in [a, b]$ ;
- 2) la **successione delle approssimazioni**

$$\left\{ x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})} \right\} \quad k = 1, 2, \dots$$

è **monotona** e **converge** a  $\xi$ ;

- 3) se  $f \in C^3[a, b]$ , la convergenza è **quadratica**.

Poichè

$$f''(a) < 0 \text{ e } f''(b) < 0 \text{ mentre } f(a) < 0 \text{ e } f(b) > 0,$$

$a$  è l'estremo di Fourier dell'intervallo  $I = [0.5 \cdot 10^6, 0.6 \cdot 10^6]$

e quindi può essere scelto come approssimazione iniziale della soluzione, cioè  $x_0 = a = 0.5 \cdot 10^6$ :

$k$	$x_k$	$e_k$	$f(x_k)$
1	565429.608054025100000	65429.608054025099000	9811.013667401624800
2	570296.151698269420000	4866.543644244317000	42.110592287266627
3	570317.219844276430000	21.068146007019095	0.000775822554715
4	570317.220232438760000	0.000388162326999	0.000000000000000
5	570317.220232438760000	0.000000000000000	0.000000000000000

## Esercizio 2

Implementare il **metodo di Newton-Raphson** utilizzando un ciclo **for** con un numero fissato di iterazioni da richiedere come parametro di input. Se al passo  $k$  si ottiene la precisione richiesta, usare il comando **break** per interrompere l'esecuzione del ciclo.

```
function [xn,err1,err2,iter] = newton(f,df,x0,eps,n_iter)
```



## Esercizio 3

Restituire come output un vettore **X** contenente le approssimazioni  $x_n$ , ottenute ad ogni passo, e un vettore **E** contenente l'errore  $e_n = |x_n - x_{n-1}|$  tra due approssimazioni successive ; al generico passo **n** avremo quindi

```
x(n) = x_n;  
err1(n) = abs(x(n) - x(n-1))
```

Prestare attenzione agli indici! Il vettore finale **x** conterrà gli elementi della successione delle approssimazioni.

# Istruzione break

**break**: arresta l' esecuzione dei cicli **for** o **while**

(non può essere utilizzata all'esterno di un ciclo)

```
X = rand(10,1);
Y=[];
for i = 1:10
    if X(i) >= 0.7
        break,
    else Y(i)=1/(0.7-X(i));
    end
end
```

E' particolarmente utile nel caso di cicli **for** in quanto permette di interrompere il ciclo anche prima che tutte le iterazioni prefissate siano state eseguite. L'istruzione **break** può essere usata nello script del metodo di Newton-Raphson in alternativa al ciclo **while**.

# Metodo delle secanti

$$\begin{cases} x_0, x_1 & \text{dati} \\ x_k = x_{k-1} - f(x_{k-1}) \frac{x_{k-1} - x_{k-2}}{f(x_{k-1}) - f(x_{k-2})}, & k = 2, \dots \end{cases}$$

## Vantaggi:

- si può usare quando **non si conosce** la derivata di  $f(x)$  o quando  $f(x)$  è **nota per punti**
- ad ogni passo richiede **una sola** valutazione funzionale

## Svantaggi:

- servono **due approssimazioni iniziali**  $x_0$  e  $x_1$
- la scelta di  $x_0$  e  $x_1$  deve essere "**accurata**"

# Metodo delle secanti: convergenza

Ipotesi di applicabilità :

- è stato **separato** un intervallo  $I = [a, b]$  simmetrico intorno alla radice  $\xi$ ;
- $f, f', f''$  sono **continue** in  $I$ :  $f \in C^2[a, b]$ ;
- $f'(x) \neq 0$  per  $x \in [a, b]$ .

$\Rightarrow$  esiste un **intorno**  $J \subseteq I$  di  $\xi$  tale che, se  $x_0, x_1 \in J$ , la **successione delle approssimazioni**  $\{x_k\}$  **converge** a  $\xi$  con convergenza **superlineare**, cioè  $2 > p > 1$ .

Se  $f''(x) \neq 0$  in  $I$ , l'**ordine di convergenza** è

$$p = \frac{1+\sqrt{5}}{2} \Rightarrow E = p \simeq 1.62$$

# Metodo delle secanti

La continuità di  $f, f'$  e  $f''$  e  $f'(L) \neq 0 \quad \forall L \in I$ , assicura la convergenza del metodo delle secanti.

Si scelgono gli estremi dell'intervallo  $I$  come punti iniziali, cioè

$$x_1 = a = 0.5 \cdot 10^6 \quad \text{e} \quad x_2 = 0.6 \cdot 10^6,$$

e si calcola il punto

$$\begin{aligned} x_3 &= x_2 - f(x_2) \frac{x_2 - x_1}{f(x_2) - f(x_1)} = \\ &= 0.6 \cdot 10^6 - \frac{57863.27995127568 \cdot 0.1 \cdot 10^6}{57863.27995127568 + 150396.8359693979} = \\ &= 572215.86106611218 \end{aligned}$$

L'errore  $e_3 = |x_3 - x_2| = 27784.138933887822 > \varepsilon = 0.5 \cdot 10^{-5}$

Calcoliamo il punto

$$\begin{aligned}x_4 &= x_3 - f(x_3) \frac{x_3 - x_2}{f(x_3) - f(x_2)} = \\&= 572215.86106611218 - \frac{3788.546320179361 \cdot (-27784.138933887822)}{3788.546320179361 - 57863.27995127568} = \\&= 570269.26807805535\end{aligned}$$

Continuando si ottiene la seguente tabella

$k$	$x_k$	$e_k$	$f(x_k)$
1	572215.86106611218	27784.13893388782200	3788.5463201793609
2	570269.26807805535	1946.592988056829200	95.846302395802923
3	570317.29971602384	48.031637968495488	0.1588643480 79078
4	570317.22023577173	0.079480252112262	0.000006661517546
5	570317.22023243876	0.000003332970664	0.0000000000000000

Il metodo converge dopo 5 iterazioni

(Oss: la  $f(x_5)$  è zero rispetto alla precisione di macchina.)

## Esercizio 4

Scrivere la **funzione metodo\_secanti** che implementi il metodo delle secanti con i seguenti parametri di input e output

```
function [xn,err1,err2,iter] = metodo_secanti(f,a,b,x0,x1,eps)
```

Input	Ouput
<b>f</b> funzione	<b>xn</b> la radice trovata
<b>a,b</b> estremi dell'intervallo	<b>err1,err2</b> gli errori effettuati ad ogni passo
<b>x0,x1</b> punti iniziali	<b>iter</b> il numero di iterazioni effettuate
<b>eps</b> tolleranza desiderata	

- se i punti iniziali non sono assegnati, usare gli estremi dell'intervallo
- usare un ciclo **while** con il doppio controllo sull'errore o un ciclo **for**
- aggiungere i commenti dopo la dichiarazione della funzione
- stampare il valore della radice e gli errori ad ogni passo
- scrivere uno script che richiami la funzione **metodo\_secanti** implementata



## Esercizio

Confrontare le approssimazioni della radice positiva di  $f$  nell'intervallo  $I = [25133, 570502]$ , usando il metodo di bisezione, di Newton-Raphson e delle secanti, con i risultati delle tabelle precedenti, eventualmente usando le routine Matlab riportate di seguito.

# Metodo di Bisezione

$$I = [25133, 570502]$$

Si osserva facilmente che il metodo converge alla soluzione con la precisione richiesta dopo 36 iterazioni

k	estremo inf.	estremo sup.	$x_k$	$e_k$	$f(x_k)$
1	297817.500000000	570502.000000000	434159.750000000	136342.25	314664.12181
2	434159.750000000	570502.000000000	502330.875000000	68171.125	145053.08450
3	502330.875000000	570502.000000000	536416.437500000	34085.5625	69892.91937
4	536416.437500000	570502.000000000	553459.218750000	17042.78125	34205.98083
5	553459.218750000	570502.000000000	561980.609375000	8521.39062	16785.70212
6	561980.609375000	570502.000000000	566241.304687500	4260.69531	8175.80282
7	566241.304687500	570502.000000000	568371.652343750	2130.34766	3895.25742
8	568371.652343750	570502.000000000	569436.826171875	1065.17383	1761.00610
9	569436.826171875	570502.000000000	569969.4130859375	532.58691	695.37596

10	569969.41308593750	570502.00000000000	570235.70654296875	266.29346	162.93356
11	570235.70654296875	570502.00000000000	570368.85327148438	133.14673	103.19463
12	570235.70654296875	570368.85327148438	570302.27990722656	66.57336	29.86171
13	570302.27990722656	570368.85327148438	570335.56658935547	33.28668	36.66839
14	570302.27990722656	570335.56658935547	570318.92324829102	16.64334	3.40382
15	570302.27990722656	570318.92324829102	570310.60157775879	8.32167	13.22882
16	570310.60157775879	570318.92324829102	570314.76241302490	4.16083	4.91247
17	570314.76241302490	570318.92324829102	570316.84283065796	2.08042	0.75431
18	570316.84283065796	570318.92324829102	570317.88303947449	1.04021	1.32476
19	570316.84283065796	570317.88303947449	570317.36293506622	0.52010	0.28522
20	570316.84283065796	570317.36293506622	570317.10288286209	0.26005	0.23455
21	570317.10288286209	570317.36293506622	570317.23290896416	0.13003	0.02534
22	570317.10288286209	570317.23290896416	570317.16789591312	0.06501	0.10460
23	570317.16789591312	570317.23290896416	570317.20040243864	0.03251	0.03963
24	570317.20040243864	570317.23290896416	570317.21665570140	0.01625	0.00715
25	570317.21665570140	570317.23290896416	570317.22478233278	0.00813	0.00909
26	570317.21665570140	570317.22478233278	570317.22071901709	0.00406	0.00097
27	570317.21665570140	570317.22071901709	570317.21868735924	0.00203	0.00309
28	570317.21868735924	570317.22071901709	570317.21970318817	0.00102	0.00106
29	570317.21970318817	570317.22071901709	570317.22021110263	0.00051	0.00004
30	570317.22021110263	570317.22071901709	570317.22046505986	0.00025	0.00046
31	570317.22021110263	570317.22046505986	570317.22033808124	0.00013	0.00021
32	570317.22021110263	570317.22033808124	570317.22027459193	0.00006	0.00008
33	570317.22021110263	570317.22027459193	570317.22024284722	0.00003	0.00002
34	570317.22021110263	570317.22024284722	570317.22022697493	0.00002	0.00001
35	570317.22022697493	570317.22024284722	570317.22023491107	0.00001	$0.5 \cdot 10^{-5}$
36	570317.22022697493	570317.22023491107	570317.22023094306	$0.4 \cdot 10^{-5}$	$0.3 \cdot 10^{-5}$

# Metodo di Newton-Raphson

$$I = [25133, 570502]$$

$k$	$x_k$	$e_k =  x_k - x_{k-1} $	$f(x_k)$
1	570317.19038567902	184.809614320984110	0.059654914657585
2	570317.22023243795	0.029846758930944	0.000000001629815
3	570317.22023243876	0.000000000814907	0.000000000000000

# Metodo delle secanti

$$I = [25133, 570502]$$

$k$	$x_k$	$e_k =  x_k - x_{k-1} $	$f(x_k)$
1	570481.52988393593	20.470116064068861	328.35959916887805
2	570317.19369166286	164.33619227306917	0.053047222900204
3	570317.22023625160	0.026544588734396	0.000007620779797
4	570317.22023243876	0.000003812834620	0.0000000000000000

## Esercizio 5

Data l'equazione non lineare  $f(x) = (x^3 - 3x + 2)e^x$  scrivere lo **script esercizio\_finale.m**. In particolare

- disegnare in rosso il grafico della funzione  $f$ ; usare i comandi per aggiungere il titolo, le etichette agli assi;
- richiamare le funzioni che implementano i **metodi di bisezione, Newton-Raphson e il metodo delle secanti** per trovare una radice di  $f$  nell'intervallo  $I = [-3, 1.5]$  con un'approssimazione di  $10^{-5}$ ;
- per ogni metodo creare la seguente tabella (introdurre il comando **fprintf** nella funzione relativa al metodo)

<code>n_iter</code>	<code>xn</code>	<code>err1</code>	<code>f(xn)</code>
---------------------	-----------------	-------------------	--------------------

- riportare sul grafico di  $f$  le radici ottenute con i tre metodi; usare il comando **legend** per identificare le tre radici relative ai tre metodi;
- stampare per ogni metodo il numero finale di iterazioni effettuate e l'errore di troncamento finale **err1**;
- visualizzare un messaggio che dica qual è il metodo che converge più velocemente.

# Esercizio 5 - output tabelle

## Bisezione

n_iter	x_n	err1	f(xn)
1	-0.750000	2.250000	1.808278
2	-1.875000	1.125000	0.158447
3	-2.437500	0.562500	0.451721
4	-2.156250	0.281250	0.180184
5	-2.015625	0.140625	0.018932
6	-1.945312	0.070312	0.067813
7	-1.980469	0.035156	0.023944
8	-1.998047	0.017578	0.002380
9	-2.006836	0.008789	0.008307
10	-2.002441	0.004395	0.002971
11	-2.000244	0.002197	0.000297
12	-1.999146	0.001099	0.001041
13	-1.999695	0.000549	0.000372
14	-1.999969	0.000275	0.000037
15	-2.000107	0.000137	0.000130
16	-2.000038	0.000069	0.000046
17	-2.000004	0.000034	0.000005

## Newton-Raphson

n_iter	x_n	err1	f(xn)
1	-1.000000	2.000000	1.471518
2	-2.000000	1.000000	0.000000

## Secanti

n_iter	x_n	err1	f(xn)
1	-2.240226	3.740226	0.268443
2	-2.000594	0.239632	0.000723
3	-1.999947	0.000647	0.000065
4	-2.000000	0.000053	0.000000



## Esercizio 6

Usare la funzione di Matlab `fzero` (guardare l'`help`) per trovare la radice della funzione  $f$  definita nell'esercizio 5.