Calcolo Numerico

(A.A. 2013-2014)

Esercitazione n.5,6,7,8

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

28,31 Marzo, 4,8 Aprile 2014

Equazioni non lineari

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

$$f(x) = 0$$

Le **soluzioni** ξ 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 ξ si passa alla seconda fase che consiste nella costruzione di un'opportuna successione $\{x_n\}$ di approssimazioni di ξ che converge alla radice ξ 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

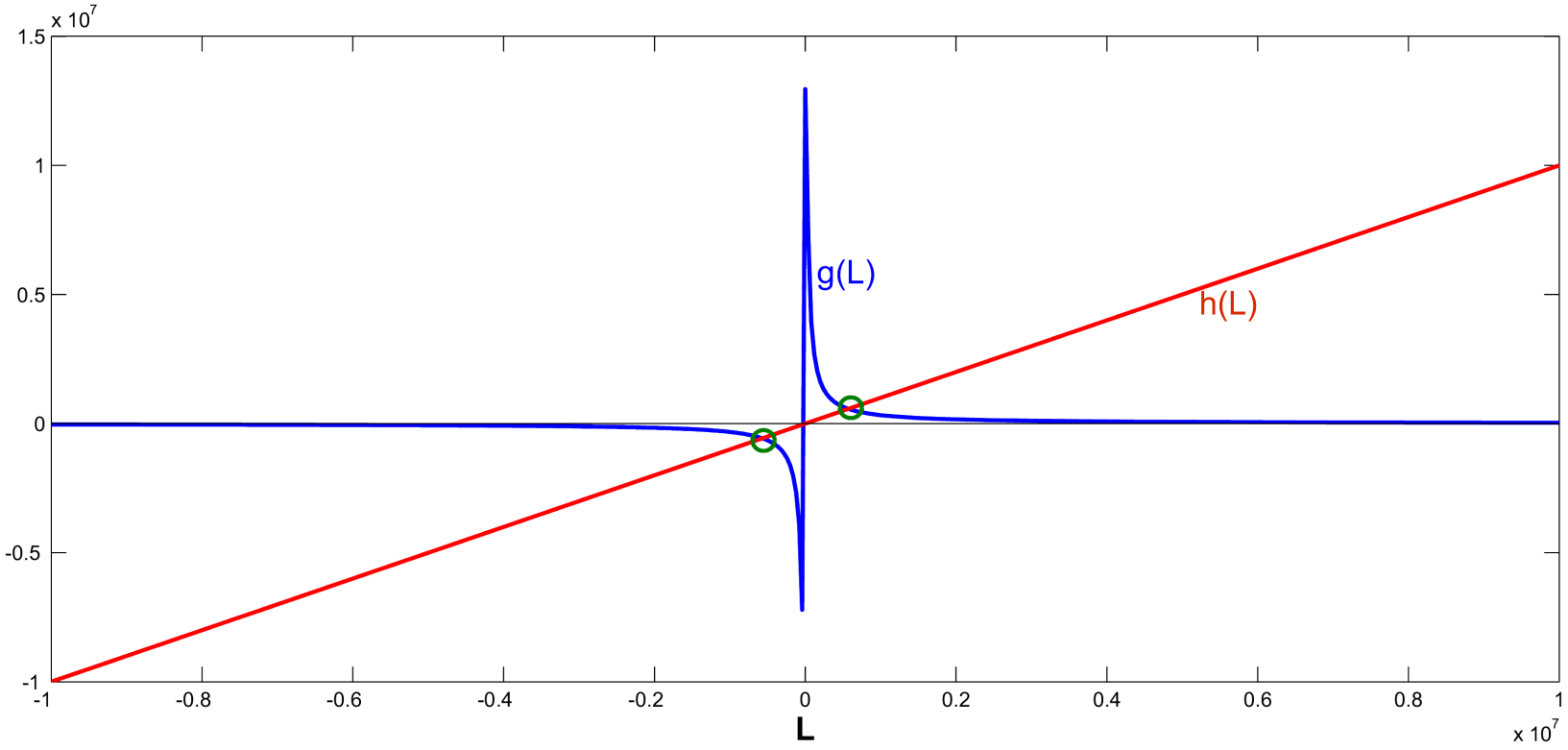
Utilizziamo il **metodo grafico** per stabilire quante sono le radici reali di $f(L)$, con $f(L) = L - \frac{a_g T^2}{2\pi} \tanh\left(\frac{2\pi d}{L}\right)$.

Poniamo

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

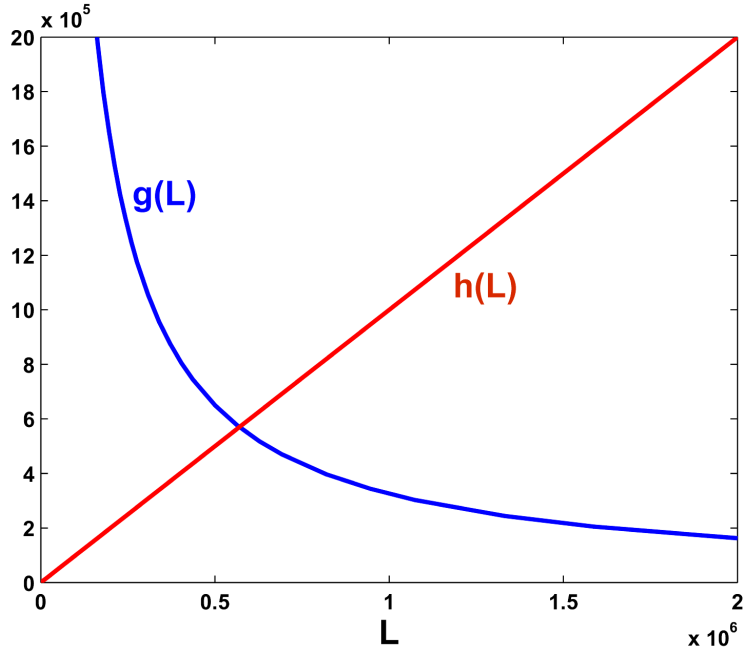
e visualizziamo i punti di intersezione tra le due funzioni.

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 $[0 \quad 2 \cdot 10^6]$, si osserva che la radice positiva è contenuta nell'intervallo $[.5{\cdot}10^6 \quad 1{\cdot}10^6]$ ($[500,\ 1000]Km$)




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

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

Metodo di bisezione (o metodo dicotomico)

Ipotesi di applicabilità :

- è stato **separato** un intervallo $I = [a, b]$ in cui c'è un'**unica radice** ξ;
- 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` $k = 1, 2, 3, \ldots$

$x_k = \frac{a_{k-1}+b_{k-1}}{2}$ (**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 ε 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 $I = [a, b] = [0.5 \cdot 10^6, \quad 0.6 \cdot 10^6]$. 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

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

sono necessarie K iterazioni con

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

affinchè il metodo converga alla soluzione con la precisione fissata.

k	estremo inferiore	estremo superiore	x_k	e_k	$f(x_k)$
1	500000.00000000000	600000.00000000000	550000.00000000000	$50 \cdot 10^{3}$	41536.18896
2	550000.00000000000	600000.00000000000	575000.00000000000	$25 \cdot 10^{3}$	9321.48847
3	550000.00000000000	575000.00000000000	562500.00000000000	$12.5 \cdot 10^{3}$	15732.61210
4	562500.00000000000	575000.00000000000	568750.00000000000	$6.250 \cdot 10^{3}$	3136.71786
5	568750.00000000000	575000.00000000000	571875.00000000000	$3.125 \cdot 10^{3}$	3109.31488
6	568750.00000000000	571875.00000000000	570312.50000000000	$1.5625 \cdot 10^{3}$	9.43440
7	570312.50000000000	571875.00000000000	571093.75000000000	$7.8125 \cdot 10^{2}$	1551.00265
8	570312.50000000000	571093.75000000000	570703.12500000000	$3.9062 \cdot 10^{2}$	771.05027
9	570312.50000000000	570703.12500000000	570507.81250000000	$19.5312 \cdot 10^{1}$	380.87454
10	570312.50000000000	570507.81250000000	570410.15625000000	97.6562	185.73673
11	570312.50000000000	570410.15625000000	570361.32812500000	48.8281	88.15533
12	570312.50000000000	570361.32812500000	570336.91406250000	24.4141	39.36151
13	570312.50000000000	570336.91406250000	570324.70703125000	12.2070	14.96382
14	570312.50000000000	570324.70703125000	570318.60351562500	6.1035	2.76477
15	570312.50000000000	570318.60351562500	570315.55175781250	3.0518	3.33480
16	570315.55175781250	570318.60351562500	570317.07763671875	1.5259	0.28501
17	570317.07763671875	570318.60351562500	570317.84057617187	$7.6 \cdot 10^{-1}$	1.23988
18	570317.07763671875	570317.84057617187	570317.45910644531	$3.8 \cdot 10^{-1}$	0.47744
19	570317.07763671875	570317.45910644531	570317.26837158203	$1.9 \cdot 10^{-1}$	0.09622
20	570317.07763671875	570317.26837158203	570317.17300415039	$9.5 \cdot 10^{-2}$	0.09439
21	570317.17300415039	570317.26837158203	570317.22068786621	$4.8 \cdot 10^{-2}$	0.00091
22	570317.17300415039	570317.22068786621	570317.19684600830	$2.3 \cdot 10^{-2}$	0.04674
23	570317.19684600830	570317.22068786621	570317.20876693726	$1.192 \cdot 10^{-2}$	0.02292
24	570317.20876693726	570317.22068786621	570317.21472740173	$6 \cdot 10^{-3}$	0.01100
25	570317.21472740173	570317.22068786621	570317.21770763397	$3 \cdot 10^{-3}$	0.00505
26	570317.21770763397	570317.22068786621	570317.21919775009	$1.5 \cdot 10^{-3}$	0.00207
27	570317.21919775009	570317.22068786621	570317.21994280815	$7.4 \cdot 10^{-4}$	0.00059
28	570317.21994280815	570317.22068786621	570317.22031533718	$3.7 \cdot 10^{-4}$	0.00017
29	570317.21994280815	570317.22031533718	570317.22012907267	$1.8 \cdot 10^{-4}$	0.00021
30	570317.22012907267	570317.22031533718	570317.22022220492	$9.3 \cdot 10^{-5}$	0.00002
31	570317.22022220492	570317.22031533718	570317.22026877105	$4.7 \cdot 10^{-5}$	0.00007
32	570317.22022220492	570317.22026877105	570317.22024548799	$2.3 \cdot 10^{-5}$	0.00003
33	570317.22022220492	570317.22024548799	570317.22023384646	$1.2 \cdot 10^{-5}$	0.000003
34	570317.22022220492	570317.22023384646	570317.22022802569	$0.6 \cdot 10^{-5}$	0.000009
35	570317.22022802569	570317.22023384646	570317.22023093607	$0.3 \cdot 10^{-5}$	0.000003

Se come criterio di arresto avessimo usato

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

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 $\mathbf{K}$ per ottenere un errore minore di ε è

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

Richiediamo come parametri di input

- il periodo $\mathbf{T}$ e la profondità dell'acqua $\mathbf{d}$
- gli estremi dell'intervallo $\mathbf{I = [a, b]}$ in cui si vuole cercare la radice
- la precisione ε con cui si vuole cercare la radice

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

`script_bisezione`

Script Matlab: Metodo di Bisezione

```
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
```

Script Matlab: Metodo di Bisezione

```
% I passo: definiamo il punto medio xn dell'intervallo [a,b] e
% calcoliamo gli errori al I passo err1=abs(xn-a0) e err2=abs(f(xn))

% dobbiamo cercare una soluzione con un numero fissato di iterazioni
% n_iter => usiamo un ciclo for

for i=2:n_iter
% ad ogni passo dobbiamo definire il nuovo intervallo In =>
% usiamo il test if...elseif...end per stabilire il nuovo estremo di In

% ad ogni passo dobbiamo calcolare gli errori err1 e err2

end
```

```
% I passo
xn = (a0+b0)/2;
err1 = abs(xn-a0);  err2 = abs(f(xn));
a = a0; b = b0;
fprintf('iterazione\t estremo inf\t estremo sup\t x_n\t e_n|\t f(x_n)\n')
fprintf('%3d\t %15.15f\t %15.15f %15.15f %6.15f\t   %6.15f\n',...
        [1 a0 b0 xn err1 err2])
% si cerca la soluzione con un numero fissato di iterazioni
for i= 2:n_iter
    if ( f(a)*f(xn) < 0 )
        b = xn;
    elseif ( f(xn)*f(b) < 0 )
        a = xn;
    end
    xv = xn;
    xn = (a+b)/2;
    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',...
    [i a b xn err1 err2])
end
```

Output dei dati

Un modo per costruire stringhe di caratteri per l'output di testo è fornito dalle funzioni **fprintf** e **sprintf** mutuate 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.

formato	azione
%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**

```
      .
      .
      .

% I passo
xn = (a0+b0)/2;
err1 = abs(xn-a0);  err2 = abs(f(xn));
a = a0; b = b0; iter = 0;

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
```

```
% 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

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

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
```

```
% 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
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
```

Metodo di Newton-Raphson: algoritmo

Ad ogni **iterazione** $k = 1, 2, \ldots$ 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 $y = 0$.

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

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

$$\downarrow$$

Algoritmo:

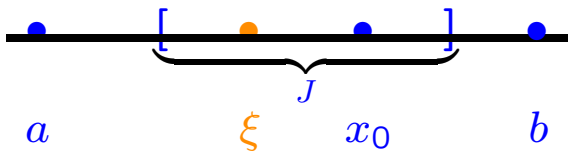
$$\begin{cases} x_0 \quad \text{dato} \\ x_k = x_{k-1} - \dfrac{f(x_{k-1})}{f'(x_{k-1})}, \quad k = 1, 2, \ldots \end{cases}$$

Metodo di Newton-Raphson: convergenza

Ipotesi di applicabilità :

- è stato **separato** un intervallo $I = [a, b]$ in cui c'è un'**unica radice** ξ;
- 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 ξ tale che, se $x_0 \in J$, la **successione delle approssimazioni** $\{x_k\}$ **converge** a ξ.



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

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

  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.

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

  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

$$x_2 = 569585.74319106806 - \frac{-1462.944269931060}{2.00126829645764 6} = 570316.7517582455.$$

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

$$x_4 = 570317.2202322469 - \frac{-3.834720700979233 \cdot 10^{-7}}{1.998706513054914} =$$

$$= 570317.220232438760000.$$

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	0.93634280480910 1
3	570317.22023224691	0.46847400139085 90	0.00000038347207 0
4	570317.22023243876	0.00000019185245 00	0.00000000000000

Dopo 4 iterazioni è stata raggiunta la precisione ε richiesta per l'errore!

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

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 $[a,b]$.

$\Rightarrow$

1) esiste un'**unica radice** $\xi \in [a,b]$;

2) la **successione delle approssimazioni**

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

è **monotona** e **converge** a ξ;

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	$\mathbf{x_k}$	$\mathbf{e_k}$	$\mathbf{f(x_k)}$
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

Script MATLAB: Metodo di Newton-Raphson

Implementare il metodo di Newton-Raphson. Richiedere come parametri di input:

- il punto iniziale $\mathbf{x_0}$
- la precisione richiesta ε

Assegnare $\mathbf{x_0 = a = 500000}$ e $\varepsilon = 0.5 * 10^{-5}$
Memorizzare le approssimazioni $\mathbf{x_k}$, ottenute ad ogni passo, in un vettore **x** e l'errore $\mathbf{e_k = |x_k - x_{k-1}|}$ tra due approssimazioni successive in un vettore **err1**; al generico passo $\mathbf{k}$ avremo quindi

```
x(k) = x_k;
err1(k) = abs(x(k) - x(k-1))
```

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

Script MATLAB: Metodo di Newton-Raphson

```
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)]
% derivata della funzione f
df = @(L)[1+(d*ag*T^2)/(L^2) * 1/((cosh(2*pi*d/L))^2)]

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

```
xn = input('inserisci il punto iniziale x0 = ');
if isempty(xn)
    % se non si fornisce il punto iniziale, si sceglie l'estremo superior
    xn = b;
end
eps = input('inserire la precisione con cui si vuole produrre la soluzion
if isempty(eps)
   % se non viene data la precisione
   % si fissa eps = .5*10^-5
    eps = .5*10^-5;
end
```

```
%inizializzazione dei parametri
iter = 0; err1 = 10; err2 = 10;
% si itera il procedimento finche l'errore tra due approssimazioni
% successive  inferiore alla precisione richiesta
while (err1>eps) & (err2>eps)
  if iter==0
    xv = xn;
  else
    xv = x(iter);
  end
  iter = iter + 1;
  x(iter) = xv-f(xv)/df(xv);
  err1(iter) = abs(x(iter)-xv);
  err2(iter) = abs(f(x(iter)));
end
v_iter = [1:iter];
A = [v_iter' x err1' err2'];
fprintf('iterazione\t\t x_n\t\t\t e_n=|x_n-x_(n-1))|\t\t f(x_n)\n')
fprintf('%3d \t\t %15.15f \t %6.15f \t %6.15f \n',A')
```

Esercizio: 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.

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**.

```
n_iter=input('inserire il numero di iterazioni da eseguire, n_iter =  ');
if isempty(n_iter)
    n_iter = 30; % se non viene dato il numero di iterazioni da eseguire,
                 % si fissa 30 come  numero massimo di iterazioni
end
fprintf('il numero di iterazioni e'' %d \n', n_iter);

fprintf('iterazione\t\t x_n\t\t\t e_n=|x_n-x_(n-1))|\t\t f(x_n)\n')
for iter= 1:n_iter
    xv = xn;
    xn = xv-f(xv)/df(xv);
    err1 = abs(xn-xv);
    err2 = abs(f(xn));
    fprintf('%3d\t\t  %15.15f\t %6.15f\t   %6.15f\n',[iter xn err1 err2])
    if (err1<eps) | (err2<eps)
      break
    end
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 dello 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 sui 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

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

Scrivere uno script che richiami la funzione **bisezione** implementata.

Function Matlab per Bisezione

```
function [xn,err1,err2,iter] = bisezione(f,a,b,eps)
%
% [xn,err1,err2,iter] = bisezione(f,a,b,eps)
% cerca la radice della funzione f nell'intervallo [a,b]
% con precisione eps utilizzando un doppio criterio di arresto
%
% INPUT
% f = espressione della funzione di cui si vuole cercare la radice
% a = estremo inferiore dell'intervallo in cui  stata isolata la radice
% b = estremo superiore dell'intervallo in cui  stata isolata la radice
% eps = limite superiore dell'errore da usare come criterio di arresto
%
% OUTPUT
% xn = approssimazione della radice
% err1 = |xn-x(n-1)|
% err2 = f(xn) valore della funzione nell'approssimazione xn
% iter = numero di iterazioni eseguite
```

```
format long;
xn = (a+b)/2;
%inizializzazione dei parametri
iter = 0; err1 = 10; err2 = 10;
% si cerca la soluzione con una certa precisione oppure la funzione sia
% abbastanza vicina a 0
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])
end
```

Script per la function bisezione

```
format long;

%% parametri relativi alla funzione di cui si vuole cercare una radice
g = 9.81; T = 2880; d = 4000;

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

% intervallo in cui si vuole cercare la radice
a0 = 500000; b0 = 600000;

eps = input('inserire la precisione, eps =  '); % per esempio .5*10^-5;

% chiamo la funzione
[xn,err1,err2,n_iter] = bisezione(f,a0,b0,eps);
```

Metodo delle secanti

$$\begin{cases} x_0, x_1 \quad \text{dati} \\ x_k = x_{k-1} - f(x_{k-1}) \dfrac{x_{k-1} - x_{k-2}}{f(x_{k-1}) - f(x_{k-2})}, \quad k = 2, \ldots \end{cases}$$

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 ξ;
- 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 ξ tale che, se $x_0, x_1 \in J$, la **successione delle approssimazioni** $\{x_k\}$ **converge** a ξ 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 motodo 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

$$x_3 = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)} =$$

$$= 0.6 \cdot 10^6 - \frac{57863.27995127568 \cdot 0.1 \cdot 10^6}{57863.27995127568 + 150396.8359693979} =$$

$$= 572215.86106611218$$

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

Calcoliamo il punto

$$x_4 = x_3 - f(x_3)\frac{x_3 - x_2}{f(x_3) - f(x_2)} =$$

$$= 572215.86106611218 - \frac{3788.546320179361 \cdot (-27784.138933887822)}{3788.546320179361 - 57863.27995127568} =$$

$$= 570269.26807805535$$

Continuando si ottiene la seguente tabella

k	$\mathbf{x_k}$	$\mathbf{e_k}$	$\mathbf{f(x_k)}$
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	0.00000661517546
5	570317.22023243876	0.000003332970664	0.00000000000000

Il metodo converge dopo 5 iterazioni

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

Esercizio

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

Function MATLAB per il Metodo delle secanti

```
function [xn,err1,err2,n_iter] = metodo_secanti(f,a,b,xn1,xn2,eps)
% cerca la radice della funzione f nell'intervallo [a,b] con precisione eps
%
% INPUT
% f       = espressione della funzione di cui si vuole cercare la radice
% a,b     = estremo inferiore e superiore dell'intervallo in cui
%           e' stata isolata la radice
% xn1,xn2 = punti iniziali dell'algoritmo
% eps     = limite superiore dell'errore da usare come criterio di arresto
%
% OUTPUT
% xn     = approssimazione della radice
% err1   = |xn-x(n-1)|
% err2   = f(xn) valore della funzione nell'approssimazione xn
% n_iter = numero di iterazioni eseguite

format long;

if isempty(eps)
    eps = .5*10^-5;
end
fprintf('eps = %6.15g \n', eps);

% come punti iniziali si scelgono gli estremi deggli intervalli
if isempty(xn1)
```

```
  xn1 = a;
end
if isempty(xn2)
  xn2 = b;
end

% 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

%inizializzazione dei parametri
iter = 0; err1 = 10; err2 = 10;

while (err1>eps)
    xn = xn2-f(xn2) *(xn2-xn1)/(f(xn2)-f(xn1));
    iter = iter + 1;
    err1 = abs(xn-xn2);
    err2 = abs(f(xn));

    fprintf('%3d\t  %15.15f   %6.15f   %6.15f\n',[iter xn  err1 err2])
    xn1 = xn2;
    xn2 = xn;
end

n_iter = iter;
```

Separazione delle radici

La separazione delle radici di $f(L)$ può essere fatta anche in modo analitico. Infatti, basta osservare che

$$\lim_{L\to 0} f(L) = -\frac{a_g T^2}{2\pi},$$

in quanto $\tanh(\frac{2\pi d}{L}) \to 1$, mentre

$$\lim_{L\to +\infty} f(L) = +\infty$$

in quanto $\tanh(\frac{2\pi d}{L}) \to 0$.

Inoltre, la derivata prima di f è strettamente positiva per $L > 0$ da cui si deduce che f è una funzione monotona crescente per $L > 0$.

Scegliendo $L = 2\pi d$, si ha

$$f(2\pi d) = 2\pi d - \frac{a_g T^2}{2\pi}\frac{e^2-1}{e^2+1} < 0,$$

mentre, considerando che $\tanh(x) \approx x$, si ha $L = \sqrt{a_g d T}$ e

$$f(\lceil\sqrt{a_g d T}\rceil) = 369.26 > 0.$$

Quindi, l'intervallo

$$I = [\lceil 2\pi d\rceil,\ \lceil\sqrt{a_g d T}\rceil] = [25133, 570502]$$

isola la radice positiva di f.

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	570502.00000000000	434159.7500000000	136342.25	314664.12181
2	434159.750000000	570502.00000000000	502330.8750000000	68171.125	145053.08450
3	502330.875000000	570502.00000000000	536416.4375000000	34085.5625	69892.91937
4	536416.437500000	570502.00000000000	553459.2187500000	17042.78125	34205.98083
5	553459.218750000	570502.00000000000	561980.6093750000	8521.39062	16785.70212
6	561980.609375000	570502.00000000000	566241.3046875000	4260.69531	8175.80282
7	566241.304687500	570502.00000000000	568371.6523437500	2130.34766	3895.25742
8	568371.652343750	570502.00000000000	569436.8261718750	1065.17383	1761.00610
9	569436.826171875	570502.00000000000	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	**570317.22022**697493	0.00002	0.00001
35	570317.22022697493	570317.22024284722	**570317.22023**491107	0.00001	$0.5 \cdot 10^{-5}$
36	570317.22022697493	570317.22023491107	**570317.22023**094306	$0.4 \cdot 10^{-5}$	$0.3 \cdot 10^{-5}$

Metodo di Newton-Raphson

$$I = [25133,\ 570502]$$

k	x_k	$e_k = \lvert x_k - x_{k-1} \rvert$	$f(x_k)$
1	570317.19038567902	184.80961432098411 0	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	$e_k = \lvert x_k - x_{k-1} \rvert$	$f(x_k)$
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	0.000000000000000

Esercizio

Data l'equazione non lineare $f(x) = (x^3 - 3x + 2)e^x$ usare i metodi di bisezione, Newton-Raphson e il metodo delle secanti per trovare una radice nell'intervallo $I = [-3, 1.5]$ con un'approsimazione di 10^{-5}

- disegnare il grafico della funzione f; usare i comandi per aggiungere il titolo, le etichette agli assi e la legenda
- riportare sul grafico le radici ottenute con i tre metodi
- confrontare il numero di iterazioni dei tre metodi e l'errore
- stabilire quale metodo converge più velocemente