# 7. ZERI DI FUNZIONI

***Esercizio 7.1***

*Considerata la seguente funzione:*

$$f(x) = e^x - x^2 - sen(x) - 1 \qquad [7.1]$$

*si chiede di calcolarne gli zeri nell'intervallo* $x \in [-2,2]$.

---

**Metodo grafico**

Un primo metodo per il calcolo degli zeri è banalmente quello grafico. In MATLAB, una volta inserita la funzione basta usare il comando `ezplot(f,[-2,2])` per ottenerla graficamente nell'intervallo di nostro interesse:

```
>> syms x
>> f=exp(x)-x^2-sin(x)-1;
>> ezplot(f,[-2,2]
>> grid
```

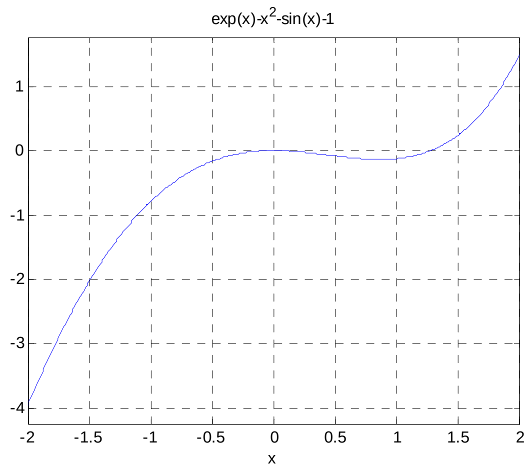


**Figura 7-1: Rappresentazione di** $\boldsymbol{f(x) = e^x - x^2 - sen(x) - 1}$

Dalla figura 7-1, si osserva che esistono due zeri, il primo in un punto $\xi_1 \in [-0.5,0.5]$ ed il secondo in un punto $\xi_2 \in [1,1.5]$. La rappresentazione grafica è però approssimativa, e soprattutto inadatta quando le funzioni siano più complicate della [7.1].
Si noti che l'intervallo in cui cade $\xi_2$ è detto "del teorema degli zeri" perché vale la condizione $f(0.5) \cdot f(1.5) < 0$, cioè vale il seguente:

<u>**Teorema-7.1**</u>: *(Teorema degli zeri)* Se una funzione f(x) di $C^0([a,b])$ è tale per cui $f(a) \cdot f(b) < 0$, allora esiste almeno uno zero di f(x) in (a,b). *(N.B.: <u>è solo condizione sufficiente</u>)*

**Metodo di bisezione**

Un metodo numerico per il calcolo degli zeri di una funzione f(x) in MATLAB è quello “di bisezione”. Esso vale solo quando la funzione f(x) cambia segno in un intervallo; quindi nel nostro caso si può applicare solo per il calcolo dello zero in $\xi_2$.

È possibile utilizzare la routine `bzero` (inclusa nelle routine allegate al presente testo) la cui sintassi è ricavabile dall’help:

```
>> help bzero
    BZERO risolve equazioni non lineari con il metodo di bisezione
    [zerob,fzerob,iterb]=bzero(f,a,b,toll) calcola lo zero della funzione f
    con tolleranza toll appartenente all'intervallo [a,b]. L'intervallo deve
essere tale per cui
    f(a)*f(b)<0.
    f è una stringa che definisce la funzione.
    La funzione restituisce zerob, approssimazione dello zero della funzione,
fzerob, valore che assume la funzione in zerob ed il numero di iterazioni
impiegato (iterb).
```

Nel nostro caso, scriveremo allora:

```
>> [zerob,fzerob,iterb]=bzero(f,0.5,1.5,1e-6)

zerob =

    1.2797

 fzerob =

  6.4097e-007

 iterb =

    20
```

Il numero di iterazioni ottenuto poteva anche essere ricavato a priori osservando che, detto *n* questo valore, vale:

$$n \geq \frac{\log\left(\frac{b-a}{toll}\right)}{\log(2)} \qquad [7.2]$$

che con i nostri numeri fornisce 19.93, cioè circa 20.

Potrebbe essere interessante osservare come decresce l’errore ad ogni iterazione. Per far questo, occorre operare sul file della routine `bzero` eliminando opportunamente i “;” nel listato in modo da far sì che MATLAB restituisca anche i dati delle operazioni intermedie.
Apriamo il file bzero eseguendo:

```
>> edit bzero.m
```

ed eliminiamo il “;” nel ciclo for alla fine della riga 56, poi salviamo e rieseguiamo il metodo. Stavolta vediamo il valore di x ad ogni iterazione. Come si osserva, l’errore sale e scende tra le varie iterazioni prima di raggiungere la convergenza: per questo metodo perciò non possiamo scrivere che $e_{n+1} = c \cdot e_n$, dove *c* indica una certa costante.

**Metodo di Newton**

Il metodo di Newton non pone alcuna condizione sull'andamento della funzione in un intervallo per il calcolo dei suoi zeri, perciò è applicabile a $\xi_1$.

Esso è un metodo iterativo così fatto:

$$\begin{cases} x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \\ x_0 \end{cases} \qquad [7.3]$$

e vale

$$\lim_{n\to+\infty} x_n = \zeta \qquad [7.4]$$

con $\zeta: f(\zeta) = 0$.

**<u>Condizione-7.1</u>**: Condizione sufficiente affinché il metodo di Newton converga è che $x_0$ sia tale per cui $f(x_0) \cdot f''(x_0) > 0$.

Nel nostro esempio, il punto -0.5 è buono per soddisfare la condizone-7.1. Infatti f(-0.5)<0 e la concavità della funzione è verso il basso, perciò f''(-0.5)<0; dunque il prodotto tra queste due dà un valore >0.

In MATLAB, possiamo usare la routine `nzero`, che ha la seguente sintassi e restituisce:

```
>>[zeron,fzeron,itern]=nzero(f,-0.5,1e-6,20,1)

zeron =

 -6.1193e-007

 fzeron =

 -1.8718e-013

 itern =

    19
```

Vogliamo ora dimostrare consistenza e convergenza del metodo di Newton.
Esso è in generale un metodo del tipo

$$\begin{cases} x_{n+1} = g(x_n) \\ x_0 \end{cases} \qquad [7.5]$$

cioè è un metodo di punto fisso.
Noi avevamo originariamente un problema di ricerca di radici e per risolverlo abbiamo usato un problema di convergenza di una successione. I due problemi sono compatibili se il secondo è consistente con il primo.

Il problema f(x)=0 impone di ricercare una radice $\zeta: f(\zeta) = 0$. Un problema del tipo [7.5], invece, si risolve giungendo ad un $\eta: \eta = g(\eta)$, cioè quando un'iterazione produce un valore identico a quello prodotto dall'iterazione precedente. La consistenza si ha quando $\zeta = \eta$, cioè quando i due problemi producono risultati identici.

Come facciamo a dire che il metodo di Newton, che è un particolare problema del tipo [7.5], è un metodo consistente?
Se osserviamo la sua forma data dalla [7.3], possiamo dire che esso si arresta quando

$$\eta = \eta - \frac{f(\eta)}{f'(\eta)} \qquad [7.6]$$

Dovendo valere $\zeta = \eta$, allora si ha che $f(\zeta) = f(\eta) = 0$ e dunque la [7.6] diventa

$$\eta = \eta - \frac{0}{f'(\eta)} = \eta$$

cioè un'identità. Allora è dimostrata la consistenza.

Per quanto riguarda la convergenza, essa si ha quando $\lim_{n\to+\infty} e_n = 0$ con $e_n = x_n - \zeta$.
Scrivo il sistema

$$\begin{cases} x_{n+1} = g(x_n) \\ \zeta = \mathrm{g}(\zeta) \end{cases} \qquad [7.7]$$

e sottraggo membro a membro la seconda dalla prima, ottenendo:

$$x_{n+1}\text{-}\zeta = g(x_n)\text{-}g(\zeta) \Rightarrow e_{n+1}{=}g'(x_n)\cdot(x_n\text{-}\zeta) \Rightarrow e_{n+1}{=}g'(x_n)\cdot e_n \qquad [7.8]$$

Affinché ci sia convergenza nell'intervallo [a,b] deve valere $|g'(x_n)| < 1$ per far sì che $e_{n+1}{<}e_n$.

Supponiamo ora che l'errore scenda con una legge del tipo:

$$e_{n+1} = c \cdot {e_n}^p \qquad [7.8]$$

Se così fosse, *c* potrebbe essere una costante qualunque dato che la convergenza dipenderebbe molto di più dalla potenza *p* dell'esponenziale piuttosto che dal termine moltiplicativo. La convergenza sarebbe tanto più veloce quanto più grande fosse *p*. *p* definirebbe quindi un ordine di convergenza, ed una volta definito questo parametro è possibile calcolare *c* come:

$$\lim_{n\to+\infty} \frac{e_{n+1}}{{e_n}^p} = c \qquad [7.9]$$

Nel nostro problema, $e_0 \approx 10^{-1}$. Vogliamo capire l'ordine del metodo di Newton nell'intervallo $\xi_1$.
Se fosse di ordine 2, cioè *p*=2, come in effetti sappiamo essere questo metodo, allora si avrebbe nelle varie iterazioni:

$$e_1 = c \cdot 10^{-2}$$

$$e_2 = c_2 \cdot 10^{-4}$$

$$e_3 = c_3 \cdot 10^{-8}$$

In sole 3 iterazioni saremmo già giunti ad un errore che rispetta la tolleranza di 1e-6. Eppure il calcolo effettuato in MATLAB richiedeva 19 iterazioni: dunque non può essere *p*=2, ma deve essere *p*=1. Che è successo? Newton è sicuramente un metodo del secondo ordine, ma il valore di potenza *p* è unitario: perché? La spiegazione è banale: in $\xi_1$ la radice è doppia; questo si può dimostrare osservando che f($\xi_1$) $= 0$ e f '($\xi_1$)=0 (ma f''($\xi_1$) è diversa da 0), cioè esiste una tangente orizzontale nel punto $\xi_1$.

È possibile correggere Newton riportandolo ad un metodo con convergenza del secondo ordine con questa correzione applicata alla [7.3]:

$$\begin{cases} x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)} \\ x_0 \end{cases} \qquad [7.10]$$

dove *m* definisce la molteplicità della radice, ed è *m*=2 nel nostro caso.

Ora vediamo come sia possibile calcolare *p* usando MATLAB.

La [7.9] vale anche per le iterate successive, cosicché ci è possibile scrivere che:

$$\lim_{n\to+\infty}\frac{e_{n+1}}{{e_n}^p} = c$$

$$\lim_{n\to+\infty}\frac{e_{n+2}}{{e_{n+1}}^p} = c \qquad [7.11]$$

Eliminando i limiti ed uguagliando le espressioni si ha che è possibile calcolare *p* come:

$$\frac{e_{n+1}}{{e_n}^p} = \frac{e_{n+2}}{{e_{n+1}}^p} \Rightarrow \frac{{e_{n+1}}^p}{{e_n}^p} = \frac{e_{n+2}}{e_{n+1}} \Rightarrow$$

$$\Rightarrow p = \frac{\log(\frac{e_{n+2}}{e_{n+1}})}{\log(\frac{e_{n+1}}{e_n})} \qquad [7.12]$$

Si potrebbe fare un *ciclo for* per trovare diversi valori di *p* nelle varie iterazioni. In MATLAB sarà però più conveniente usare dei vettori, come segue. Generiamo un nuovo m-file che chiamiamo `erroreese7.m` e che contiene il seguente codice:

```
% Codice file erroreese7.m

x0=-0.5;
zero=0;
for i=1:20
    xn=x0-(exp(x0)-x0^2-sin(x0)-1)/(exp(x0)-2*x0-cos(x0));      % è f/f'
    err(i)=abs(xn-zero);
    x0=xn;
end
p=log(err(3:end)./err(2:end-1))./log(err(2:end-1)./err(1:end-2))
```

Invece che calcolare il valore di *p* ad ogni iterazione, facciamo in modo che *p* sia un vettore che contenga in ogni posizione uno specifico valore di una specifica iterazione.
Possiamo richiamare il file `erroreese7.m` ed eseguirlo. Osserviamo che i valori di p riportati da MATLAB sono tutti prossimi a 1, e convergenti a questo valore. La cosa si può verificare anche eseguendo un `plot(p)` ed un `plot(err)`, osservando che l'errore converge a zero mentre *p* converge ad 1 come ci aspettavamo. Cioè, se non modifichiamo il metodo di Newton come richiesto dalla [7.10], esso risulta un metodo del prim'ordine.

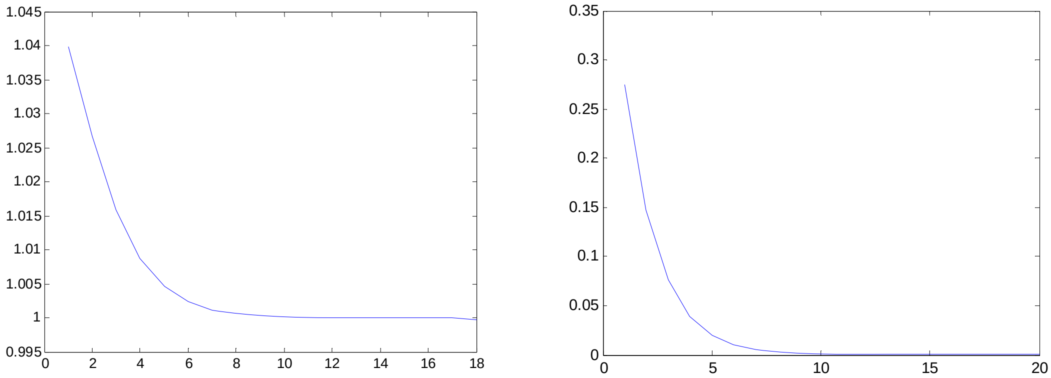


**Figura 7-2: Rappresentazione di *p* (*a sinistra*) e dell'errore *err* (*a destra*) in funzione del numero di iterazioni in ascissa**

Torniamo ora nuovamente all'espressione [7.5], ed alla nostra funzione *f(x)* data dalla [7.1]. Se pongo *f(x)*=0, posso scriverla come un'espressione del tipo *x=g(x)*? Sì, per esempio:

$$f(x) = 0 \Rightarrow e^x - x^2 - sen(x) - 1 = 0 \Rightarrow e^x = x^2 + sen(x) + 1 \Rightarrow$$
$$\Rightarrow x = \ln(x^2 + sen(x) + 1) \qquad [7.13]$$

Allora, posso scrivere un metodo di punto fisso del tipo:

$$\begin{cases} x_{n+1} = \ln(x^2 + sen(x) + 1) \\ x_0 \end{cases} \qquad [7.14]$$

Posso eseguire questo metodo simbolicamente, ma non è detto che sia convergente (mentre la consistenza è sicuramente verificata, visto il modo con cui siamo giunti alla [7.14]).

Implemento questo metodo in due nuovi *m-file*, che chiamo `erroreese72.m` ed `erroreese73.m`:

```
% Codice file erroreese72.m

x0=1.5;
for i=1:20
    xn=log(x0^2+sin(x0)+1);
    %err(i)=abs(xn-zero);
    x0=xn
end
%p=log(err(3:end)./err(2:end-1))./log(err(2:end-1)./err(1:end-2))
```

```
% Codice file erroreese73.m

x0=1.5;
for i=1:100
    xn(i)=log(x0^2+sin(x0)+1);
    x0=xn(end)
end
err=abs(xn-xn(end));err=err(1:end-1);
p=log(err(3:end)./err(2:end-1))./log(err(2:end-1)./err(1:end-2))
```

Se li eseguo, osservo la non convergenza.

Concludiamo mostrando che per i metodi di punto fisso esiste anche questa definizione di ordine.

**<u>Teorema-7.2:</u>** *(Ordine di un metodo di punto fisso)* Sia $\zeta: f(\zeta) = 0$. Se esiste un metodo di punto fisso per il quale è soddisfatta la consistenza, cioè $\zeta = g(\zeta)$ e vale che la prima derivata non nulla della funzione di iterazione è quella del *p*-esimo ordine (cioè $g^{(l)}(\zeta) = 0$ per *l*=1,…,p-1 e $g^{(p)}(\zeta) \neq 0$) allora *p* è l'ordine del metodo di punto fisso.

Vale infine che:

$$\lim_{n \to +\infty} \frac{e_{n+1}}{{e_n}^p} = c = \frac{g^{(p)}(\zeta)}{p!} \qquad [7.13]$$