EQUAZIONI NONLINEARI [*]

A. SOMMARIVA [†] E M.VENTURIN [‡]

1. Soluzione numerica di equazioni nonlineari. Data una funzione continua $y = f(x)$, si desidera calcolare x^* tale che $f(x^*) = 0$. Questo problema è assai diffuso nel calcolo numerico e richiede in generale l'utilizzo di un *metodo iterativo* per approssimare tali *soluzioni* x^*. In altri termini, partendo da un valore iniziale x_0 si genera una sequenza di valori $x_1, x_2, x_3, \ldots$ che si desidera convergano, magari velocemente, ad una opportuna soluzione x^*.

Un esempio significativo è rappresentato dal caso in cui f sia un polinomio p_n di grado n, ed è ben noto da un teorema di Galois che per $n \geq 5$, non esistono formule risolutive per il calcolo degli zeri dell'equazione $p_n(x) = 0$ che richiedano un numero finito di operazioni. Ciò non vieta di approssimare tali radici con un metodo numerico compiendo un errore assoluto e/o relativo inferiore a un valore prestabilito detto *tolleranza*.

Osserviamo che quelle polinomiali non sono gli unici esempi di equazioni nonlineari. Si consideri ad esempio il problema di calcolare gli zeri di $f(x) = \sin x - x$, cioè quei valori x^* per cui $\sin x^* - x^* = 0$. Visto che

$$\sin x^* - x^* = 0 \Leftrightarrow \sin x^* = x^*$$

e

$$|\sin x| \leq 1$$

risulta evidente che se esiste un tale zero, necessariamente

$$x^* \in [-1, 1].$$

Eseguiamo il codice Matlab/Octave

```
>> x=-10:0.01:10;
>> y=sin(x)-x;
>> plot(x,y);
```

Dal plot appare chiaro che se esiste uno zero in $[-10, 10]$ allora sta nell'intervallo $[-1, 1]$. Scriviamo quindi

```
>> x=-1:0.01:1;
>> y=sin(x)-x;
>> plot(x,y);
```

L'unico zero sembra essere $x^* = 0$. In effetti la funzione f è continua e decrescente essendo $f'(x) = \cos x - 1 \leq 0$ (poichè $|\cos x| \leq 1$) ed assume valori positivi e negativi. Quindi ha un unico zero $x^* = 0$ (si osservi che essendo $\sin 0 = 0$ abbiamo $\sin 0 - 0 = 0$).

In un metodo iterativo, ci sono due aspetti specifici di cui è opportuno tenere conto

---

[*] Ultima revisione: 22 giugno 2010

[†] DIPARTIMENTO DI MATEMATICA PURA ED APPLICATA, UNIVERSITÀ DEGLI STUDI DI PADOVA, VIA TRIESTE 63, 35121 PADOVA, ITALIA (ALVISE@MATH.UNIPD.IT)

[‡] DIPARTIMENTO DI INFORMATICA, UNIVERSITÀ DEGLI STUDI DI VERONA, STRADA LE GRAZIE 15, 37134 VERONA, ITALIA (MANOLO.VENTURIN@GMAIL.COM)

1. Garanzia della convergenza alla soluzione: se $\{x_n\}$ è la soluzione generata dal metodo e x^* è uno zero per f, si cercano delle condizioni per cui $x_n \to x^*$.

2. Se $x_n \to x^*$ si cerca la velocità con cui ciò accade. In tal senso, è importante calcolare il cosidetto *ordine di convergenza*. Sia $\{x_k\}$ una successione convergente ad x^* e sia $e_k = x_k - x^*$ l'errore al passo k. Se esiste un numero $p > 0$ e una costante $C \neq 0$ tale che

$$\lim_{n\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = C$$

allora p è chiamato *ordine di convergenza* della successione e C è la *costante asintotica di errore*. Vediamo con un semplice esempio di confrontare in Matlab il caso in cui sia per $k = 0, 1, 2, \ldots$

$$\frac{|e_{k+1}|}{|e_k|^p} = \frac{1}{10},\ |e_0| = 1$$

cioè

$$|e_{k+1}| = \frac{1}{10}\,|e_k|^p,\ |e_0| = 1.$$

Evidentemente

$$\lim_{n\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = \frac{1}{10},$$

e il metodo ha ordine di convergenza p. Se abbiamo $p = 1$ la successione prodotta sarà :

$$e_0 = 1,\ e_1 = 1/10,\ e_2 = 1/100,\ e_3 = 1/1000,\ \ldots$$

mentre per $p = 2$ abbiamo facilmente

$$e_0 = 1,\ e_1 = 1/10,\ e_2 = 1/1000,\ e_3 = 1/10^7, \ldots,$$

per cui si vede che a parità di C, maggiore è p allora minore è l'errore e_k compiuto a parità di k. Per rendercene maggiormente conto digitiamo in Matlab:

```
C=1/10;
e1(1)=1;
e2(1)=1;

M=6;
for index=1:M
  e1(index+1)=C*e1(index);
  e2(index+1)=C*(e2(index))^2;
end

L=length(e1);
semilogy(0:L-1,e1,'rd-'); hold on; semilogy(0:L-1,e2,'ko-');
```

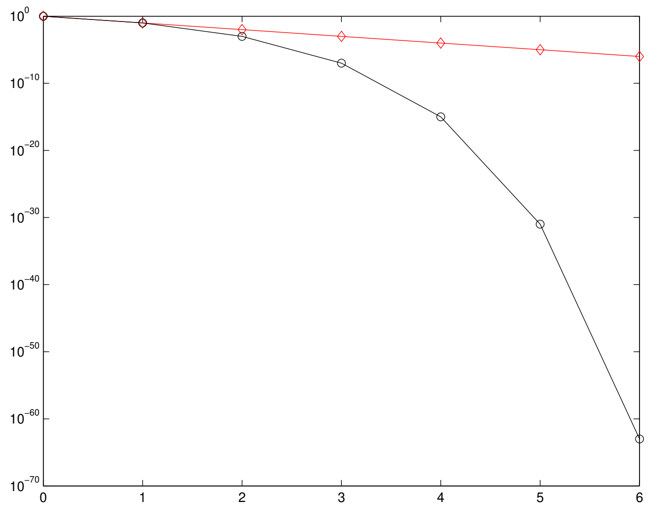

FIGURA 1.1. *Grafico che illustra l'errore di un metodo con convergenza $p = 1$ (in rosso a rombi) e $p = 2$ (in nero a cerchietti), per $C = 1/10$ ed $e_0 = 1$.*

ottenendo il grafico in figura.
Dal punto di vista pratico per avere un'idea dell'ordine di convergenza si effettua un plot semi-logaritmico dell'errore. Se esso appare (circa) come una retta decrescente vuol dire che la (possibile) convergenza ha ordine 1 (e viene detta lineare). Altrimenti se è al di sotto di qualsiasi retta (con coefficiente angolare negativo) si dice *superlineare*, se al di sopra *sublineare*.

2. Metodo di bisezione. Sia $f : [a, b] \to \mathbb{R}$ una funzione continua e supponiamo $f(a) \cdot f(b) < 0$. Di conseguenza per il *teorema degli zeri di una funzione continua* l'intervallo (a, b) contiene almeno uno zero di f. Supponiamo per semplicità che sia uno solo, come nel caso in cui f sia crescente o decrescente.

Il metodo di bisezione può essere definito in forma algoritmica (cf. [2], p. 408):

1. si genera una successione di intervalli (a_k, b_k) con
$$f(a_k) \cdot f(b_k) < 0,$$
$$[a_k, b_k] \subset [a_{k-1}, b_{k-1}],$$
$$|b_k - a_k| = \frac{1}{2}|b_{k-1} - a_{k-1}|.$$
2. Date due tolleranze $\epsilon_1 > 0$, $\epsilon_2 > 0$ si arresta l'algoritmo o quando
$$|b_k - a_k| \le \epsilon_1$$
o quando
$$|f((a_k + b_k)/2)| \le \epsilon_2 \tag{2.1}$$
o quando $k >$ nmax, ove nmax è un numero massimo di iterazioni. Successivamente descriveremo ed implementeremo una variante di (2.1) all'interno del nostro criterio di arresto.

Fissata una tolleranza ϵ, per avere un'errore assoluto sulla soluzione inferiore ad ϵ necessitano al piú

$$n \geq \frac{\log(b-a) - \log \epsilon}{\log 2}$$

iterazioni del metodo.

2.1. Implementazione Matlab del metodo di bisezione. Forniamo ora un'implementazione Matlab del metodo di bisezione per calcolare uno zero x^* di $f(x) = 0$ con $f : [a, b] \to \mathbb{R}$ funzione continua. Tale procedura che chiameremo `bisezione` è definita come segue

```
function [c,k,semiampiezza,wres]=bisezione(a,b,tolintv,tolres,maxit,f)
```

Le variabili di output sono

1. `c` è un vettore relativo alla sequenza di approssimazioni della soluzione dell'equazione nonlineare $f(x) = 0$ fornita dal metodo di bisezione;
2. `k` l'iterazione in cui termina il metodo di bisezione;
3. `semiampiezza` è un vettore contenente la semi-ampiezza di ogni intervallo $[a_j, b_j]$ analizzato dal metodo di bisezione;
4. `wres` è un vettore la cui componente j-sima consiste del residuo pesato

   $$|f(c_j) \cdot (b_j - a_j)/(f(b_j) - f(a_j))| := |f(c_j) \cdot w_j|$$

   con c_j punto medio di $[a_{j-1}, b_{j-1}]$ (il punto c_j coincide con a_j oppure b_j) e

   $$w_j := \left(\frac{f(b_j) - f(a_j)}{(b_j - a_j)}\right)^{-1}.$$

   Se la funzione è ripida, allora $|w_j|$ è piccolo e quindi visto che il residuo

   $$|f(x_j)|$$

   non è un buon test (il residuo può essere grande pure se siamo vicini alla soluzione), lo pesa tenendo conto della pendenza; se invece la funzione è molto piatta nuovamente il residuo non è un buon test (il residuo può essere piccolo pure se siamo lontani dalla soluzione) nuovamente lo pesa tenendo conto che $|w_j|$ è grande;

mentre quelle di input

1. `a` estremo minore dell'intervallo contenente lo zero di f;
2. `b` estremo maggiore dell'intervallo contenente lo zero di f;
3. `tolintv` massima semi-ampiezza ammissibile dell'ultimo intervallo analizzato dal metodo di bisezione;
4. `tolres` tolleranza del residuo pesato;
5. `maxit` numero massimo di iterazioni del metodo di bisezione;
6. `f` funzione di cui si vuole calcolare l'unico zero contenuto in $[a, b]$; la funzione f deve essere registrata in un m-file come ad esempio

```
function y=f(x)
y=sin(x)-x;
```

Un codice che implementa il metodo di bisezione è il seguente:

```
function [c,k,semilunghezza,residuopesato]=bisezione(a,b,tolintv,...
    tolres,maxit,f)

%--------------------------------------------------------------------------
% NOTA 1.
%
% SE L'UTENTE VUOLE PASSARE LA FUNZIONE g SALVATA IN g.m, SCRIVERA'
%    [...]=bisezione(...,maxit,'g').
%
%--------------------------------------------------------------------------
% NOTA 2.
%
% SI PUO' DEFINIRE LA FUNZIONE 'g' ANCHE USANDO "inline".
%
% ESEMPIO:
%
% >> g=inline('sin(x)-x');
% >> [c,k,semilunghezza,residuopesato]=bisezione(-0.45,0.4,10^(-6),...
%    10^(-6),40,'g');
%         [IT]:  1 [c]: -0.02500 [a]:-0.025[AMP]: 2.13e-01 [WRES]:1.54e-03
%    ...
%         [IT]: 18 [c]: 0.40000 [a]:0.400[AMP]: 1.62e-06 [WRES]:1.85e-01
% >>
%--------------------------------------------------------------------------


% PONI "a < b" ANCHE IN CASO DI ERRORE DELL'UTENTE.
if b < a
    s=b; b=a; a=s;
end

fa=feval(f,a);
fb=feval(f,b);

% SE UNO TRA a E b E' LA SOLUZIONE, ALLORA ESCI DOPO AVER
% ASSEGNATO L'OUTPUT.

if fa == 0
    c=a; k=0; semilunghezza=(b-a)/2; residuopesato=0;
    return;
end

if fb == 0
    c=b; k=0; semilunghezza=(b-a)/2; residuopesato=0;
    return;
end

% SE IL NUMERO DI ITERAZIONI NON E' ECCESSIVO ALLORA PROCEDI.
```

```
for index=1:maxit

    c(index)=(a+b)/2;
    fc=feval(f,c(index));

    % CALCOLA IL NUOVO INTERVALLO [a,b].
    if sign(fc) == sign(fa)
        a=c(index); fa=fc;  %SUB INTV. DX.
    else
        b=c(index); fb=fc;  %SUB INTV. SX.
    end

    % CALCOLA SEMILUNGHEZZA E RESIDUO PESATO.
    semilunghezza(index)=(b-a)/2;
    den=(fb-fa);
    if den == 0
       den=eps; % SE IL DENOMINATORE "den" E' NULLO, PONGO "den=eps".
    end
    w=(b-a)/den;
    residuopesato(index)=abs(fc*w);

    % SE QUALCHE TEST DI ARRESTO E' VERIFICATO, ESCI DALLA FUNZIONE.
    if (residuopesato(index) < tolres) | ...
             (semilunghezza(index) < tolintv) | (fc == 0)
        k=index;
        fprintf('\n');
        return;
    end

   fprintf('\n \t [IT]:%3.0f [c]: %5.5f [a]:%3.3f',index,c(index),a)
   fprintf('[AMP]: %2.2e [WRES]:%2.2e',semilunghezza(index),...
        residuopesato(index));

end

k=maxit;

fprintf('\n')
```

Descriviamo passo passo il programma (a meno di dettagli che lasciamo al lettore).

1. Supposto $I_0 = [a_0, b_0] \equiv [a, b]$ con $f(a) \cdot f(b) < 0$, calcoliamo c_1 punto medio di I_0. Se $f(c_1) = 0$ si esce, determinando i parametri richiesti dal metodo. Altrimenti si determina l'intervallo $I_1 = [a_1, b_1]$ con $a_1 = c_1$ e $b_1 = b_0$ oppure $b_1 = c_1$ e $a_1 = a_o$ in modo che sia $f(a_1) \cdot f(b_1) < 0$.

2. Registrata la semi-ampiezza di I_1 nella prima componente del vettore `semilunghezza` e il residuo pesato

$$\rho_1 := |f(c_1) \cdot (b_1 - a_1)/(f(b_1) - f(a_1))|$$

nella prima componente del vettore `residuopesato`, si testa se i criteri di arresto sono verificati, cioè se la semi-ampiezza dell'intervallo è inferiore a `tolintv` o il residuo pesato è minore di `tolres` o si è raggiunto un numero massimo di iterazioni `maxit`. Se uno solo di questi test viene soddisfatto si esce dalla routine altrimenti si continua.

3. Supposto $I_k = [a_k, b_k]$ con $f(a_k) \cdot f(b_k) < 0$, calcoliamo c_{k+1} punto medio di I_k. Se $f(c_{k+1}) = 0$ si esce, determinando i parametri richiesti dal metodo. Altrimenti si considera l'intervallo $I_{k+1} = [a_{k+1}, b_{k+1}]$ con $a_{k+1} = c_{k+1}$ e $b_{k+1} = b_k$ oppure $b_{k+1} = c_{k+1}$ e $a_{k+1} = a_k$ in modo che sia $f(a_{k+1}) \cdot f(b_{k+1}) < 0$.

4. Registrata la semi-ampiezza di I_{k+1} nella $k+1$-sima componente del vettore `semiamp` e il residuo pesato

$$\rho_{k+1} := |f(c_{k+1}) \cdot (b_{k+1} - a_{k+1})/(f(b_{k+1}) - f(a_{k+1}))|$$

nella $k+1$-sima componente del vettore `wres`, si testa se i criteri di arresto sono verificati, cioè se la semi-ampiezza dell'intervallo I_{k+1} è inferiore a `tolintv` o il residuo pesato è minore di `tolresiduo` o si è raggiunto un numero massimo di iterazioni `maxit`. Se uno solo di questi test viene soddisfatto si esce dalla funzione altrimenti si continua.

Notiamo alcune cose:

1. Per far eseguire ai programmi esattamente 50 iterazioni, si può impostare il numero massimo di iterazioni a 50 e le tolleranze sulle semi-ampiezze e i residui uguali a 0 (o perfino a -1).

2. si supponga sia `f` una funzione registrata in un m-file come ad esempio

```
function y=f(x)
y=sin(x)-x;
```

e la si voglia valutare in un punto x. La chiamata

```
y=feval('f',x)
```

valuta la funzione f nel punto x. Se f è funzione vettoriale, allora x può essere un vettore. Vediamo un esempio, supposto che in f sia registrata la funzione sopra menzionata.

```
>> y=feval('f',x)

y =

   -3.1416
>> y=feval(f,x)
??? Input argument 'x' is undefined.

Error in ==> C:\MATLAB6p1\work\f.m
On line 2  ==> y=sin(x)-x;
>>
```

Notiamo che la mancanza di apici in f genera un errore di sintassi. Nella chiamata della procedura di bisezione, la variabile f viene passata come stringa 'f' e quindi all'interno della procedura di bisezione, f è per l'appunto una stringa, da cui si può scrivere

```
fa=feval(f,a);
fb=feval(f,b);
```

invece di

```
fa=feval('f',a);
fb=feval('f',b);
```

3. A volte può essere comodo definire una funzione g senza scriverla su un file `g.m`, bensì utilizzando il comando `inline`. Un esempio è il seguente

   ```
   >> g=inline('sin(x)-x');
   >> [c,k,semilunghezza,residuopesato]=bisezione(-0.45,0.4,...
           10^(-6),10^(-6),40,'g');

      [IT]:  1 [c]: -0.02500 [a]:-0.025[AMP]: 2.13e-01 [WRES]:1.54e-03
      [IT]:  2 [c]: 0.18750 [a]:0.188[AMP]: 1.06e-01 [WRES]:5.52e-02
             ... ... ...
      [IT]: 17 [c]: 0.39999 [a]:0.400[AMP]: 3.24e-06 [WRES]:1.85e-01
      [IT]: 18 [c]: 0.40000 [a]:0.400[AMP]: 1.62e-06 [WRES]:1.85e-01
   >>
   ```

4. Per disegnare il grafico degli errori relativi a `rad2`
   ```
   function [fx]=rad2(x)
   fx=x^2-2;
   ```
   si esegua

   ```
   [x,k,semiamp,wres]=bisezione(1,2,0,0,50,'rad2');
   x_fin=x(length(x));
   semilogy(abs(x-x_fin)/x_fin,'r-');
   ```

   Tale codice produce il grafico degli errori relativi al metodo. Osserviamo che `semilogy` chiede nell'ordine il vettore di ascisse e il vettore di ordinate. Qualora ci sia un solo vettore lo prende quale vettore delle ordinate, con ascisse il vettore degli indici da 1 alla lunghezza del vettore delle ordinate.

Da provare Salvare su un file **f.m** la function

```
function fx=f(x)
fx=sin(x)-x;
```

e quindi digitare dal workspace di Matlab/Octave

```
[x,k,semiamp,wres]=bisezione(-3,2,0,0,50,'f');
```

Il risultato come previsto è $x = 0$. Più precisamente

```
[IT]: 1 [c]: -0.50000 [z1]:-0.500 [AMP]:1.25e+000 [WRES]:4.63e-002
[IT]: 2 [c]:  0.75000 [z1]:-0.500 [AMP]:6.25e-001 [WRES]:9.61e-001
[IT]: 3 [c]:  0.12500 [z1]:-0.500 [AMP]:3.13e-001 [WRES]:9.73e-003
[IT]: 4 [c]: -0.18750 [z1]:-0.188 [AMP]:1.56e-001 [WRES]:2.41e-001
```

```
[IT]: 5 [c]: -0.03125 [z1]:-0.031 [AMP]:7.81e-002 [WRES]:2.41e-003
[IT]: 6 [c]:  0.04688 [z1]:-0.031 [AMP]:3.91e-002 [WRES]:6.03e-002
[IT]: 7 [c]:  0.00781 [z1]:-0.031 [AMP]:1.95e-002 [WRES]:6.01e-004
[IT]: 8 [c]: -0.01172 [z1]:-0.012 [AMP]:9.77e-003 [WRES]:1.51e-002
[IT]: 9 [c]: -0.00195 [z1]:-0.002 [AMP]:4.88e-003 [WRES]:1.50e-004
[IT]:10 [c]:  0.00293 [z1]:-0.002 [AMP]:2.44e-003 [WRES]:3.77e-003
......
[IT]:40 [c]: -0.00000 [z1]:-0.000 [AMP]:2.27e-012 [WRES]:4.55e-012
[IT]:41 [c]: -0.00000 [z1]:-0.000 [AMP]:1.14e-012 [WRES]:2.27e-012
[IT]:42 [c]: -0.00000 [z1]:-0.000 [AMP]:5.68e-013 [WRES]:0.00e+000
[IT]:43 [c]: -0.00000 [z1]:-0.000 [AMP]:2.84e-013 [WRES]:5.68e-013
[IT]:44 [c]: -0.00000 [z1]:-0.000 [AMP]:1.42e-013 [WRES]:0.00e+000
[IT]:45 [c]: -0.00000 [z1]:-0.000 [AMP]:7.11e-014 [WRES]:0.00e+000
[IT]:46 [c]: -0.00000 [z1]:-0.000 [AMP]:3.55e-014 [WRES]:0.00e+000
[IT]:47 [c]: -0.00000 [z1]:-0.000 [AMP]:1.78e-014 [WRES]:3.55e-014
[IT]:48 [c]: -0.00000 [z1]:-0.000 [AMP]:8.88e-015 [WRES]:0.00e+000
[IT]:49 [c]: -0.00000 [z1]:-0.000 [AMP]:4.44e-015 [WRES]:8.88e-015
[IT]:50 [c]: -0.00000 [z1]:-0.000 [AMP]:2.22e-015 [WRES]:4.44e-015
```

Notiamo che a volte sul workspace compare

```
error: `c' undefined near line 3 column 8
error: evaluating argument list element number 1
error: evaluating binary operator `-' near line 3, column 10
error: evaluating assignment expression near line 3, column 3
error: ...
```

In molti casi è sufficiente rilanciare il comando

```
[c,k,semiamp,wres]=bisezione(-3,2,0,0,50,'f')
```

per avere i risultati desiderati.

3. Esercizio per casa. Calcolare con i metodi di bisezione e Newton un'approssimazione dello zero α di

$$5x = \exp{(-x)} \tag{3.1}$$
$$x^4 - 2 = 0 \tag{3.2}$$
$$x \log{(x)} - 1 = 0 \tag{3.3}$$
$$(x-1)\log{(x)} = 0 \tag{3.4}$$
$$m = x - E\sin{(x)},\ m = 0.8,\ E = 2 \tag{3.5}$$

Verificare che per le approssimazioni $x^* \approx \alpha$ fornite sia effettivamente $f(x^*) \approx 0$. Posto come x^* il risultato finale, si disegni il grafico degli errori $|x^{(k)} - x^*|$ in scala semilogaritmica. Da questa dedurre, se possibile, l'ordine di convergenza del metodo e giustificare teoricamente la risposta. Alcuni suggerimenti:

1. si controlli sempre che la directory corrente contenga i files Matlab/Octave da usare; ad esempio se voglio usare `bisezione` e una funzione `f.m` devo controllare che siano presenti nella directory corrente come nell'esempio

   ```
   >> ls
   ```

```
Dg1.m   dg.m   g.m   rad.m
bisezione.m   drad2.m   g1.m   rad2.m
bisezione.m~   f.m   newton.m
>>
```

A volte appaiono files con la *tilda*, che sono alcuni backups automatici di Matlab.

2. alcuni utenti di Octave possono incontrare problemi con l'editor `Kate` che aggiunge caratteri indesiderati al testo; quale soluzione si utilizzi invece `pico`di `Kate`;
3. si ricordi che la chiamata di bisezione è del tipo

   ```
   [c,k,semiamp,wres]=bisezione(-3,2,0,0,50,'f')
   ```

   e non

   ```
   [c,k,semiamp,wres]=bisezione(-3,2,0,0,50,f)
   ```

   (si scriva cioè `'f'` e non `f` per passare la funzione da studiare).
4. può essere utile descrivere la funzione `f` come inline; ad esempio

   ```
   >> f=inline('sin(x)-x');
   >> [c,k,semiamp,wres]=bisezione(-0,35,0.45,0.001,0.001,50,f)
   ```

5. si osservi che alcune funzioni non sono scritte come $f(x) = 0$ che è la formulazione richiesta; a tale scopo portare tutti i termini a primo membro e determinare la f.
6. per determinare gli estremi di bisezione a, b a volte può essere utile eseguire il grafico della funzione.

NOTA 3.1. *Il metodo di bisezione non ha, secondo le definizioni date, ordine di convergenza 1, come molti studenti ritengono. Vediamo il seguente test, relativamente all'equazione*

$$\sin(x) - x = 0.$$

Apriamo il file `test_bisezione`

```
% BISEZIONE E ORDINE DI CONVERGENZA.

f=inline('sin(x)-x'); % HA ZERO IN "sol=0".
a=-3; b=2;
tolintv=10^(-6);
tolres=10^(-6);
maxit=20;
[c,k,semilunghezza,residuopesato]=bisezione(a,b,tolintv,...
    tolres,maxit,f);
e_n=-c(1:end-1); % e_n=c_n-sol=c_n-sol=-c_n.
e_nplus1=-c(2:end);   % e_nplus1=c_{n+1}-sol=c_{n+1}-0=c_{n+1}.
ratio=abs(e_nplus1./e_n);

fprintf('\n ----- VALORI DI  abs( e_(n+1)/e_n ) -----')

ratio'
```

Per prima cosa osserviamo che la soluzione dell'equazione $\sin(x) - x = 0$ *è* $x^* = 0$ *Otteniamo, osservato che* $e_n = c_n - x^* = c_n$ *e quindi* $e_{n+1}/e_n = c_{n+1}/c_n$. *Lanciamo il programma appena descritto e otteniamo*

```
>> test_bisezione

   [IT]:  1 [c]: -0.50000 [a]:-0.500[AMP]: 1.25e+00 [WRES]:4.63e-02
   [IT]:  2 [c]: 0.75000 [a]:-0.500[AMP]: 6.25e-01 [WRES]:9.61e-01
   [IT]:  3 [c]: 0.12500 [a]:-0.500[AMP]: 3.12e-01 [WRES]:9.73e-03
   [IT]:  4 [c]: -0.18750 [a]:-0.188[AMP]: 1.56e-01 [WRES]:2.41e-01
   [IT]:  5 [c]: -0.03125 [a]:-0.031[AMP]: 7.81e-02 [WRES]:2.41e-03
   [IT]:  6 [c]: 0.04688 [a]:-0.031[AMP]: 3.91e-02 [WRES]:6.03e-02
   [IT]:  7 [c]: 0.00781 [a]:-0.031[AMP]: 1.95e-02 [WRES]:6.01e-04
   [IT]:  8 [c]: -0.01172 [a]:-0.012[AMP]: 9.77e-03 [WRES]:1.51e-02
   [IT]:  9 [c]: -0.00195 [a]:-0.002[AMP]: 4.88e-03 [WRES]:1.50e-04
   [IT]: 10 [c]: 0.00293 [a]:-0.002[AMP]: 2.44e-03 [WRES]:3.77e-03
   [IT]: 11 [c]: 0.00049 [a]:-0.002[AMP]: 1.22e-03 [WRES]:3.76e-05
   [IT]: 12 [c]: -0.00073 [a]:-0.001[AMP]: 6.10e-04 [WRES]:9.42e-04
   [IT]: 13 [c]: -0.00012 [a]:-0.000[AMP]: 3.05e-04 [WRES]:9.39e-06
   [IT]: 14 [c]: 0.00018 [a]:-0.000[AMP]: 1.53e-04 [WRES]:2.35e-04
   [IT]: 15 [c]: 0.00003 [a]:-0.000[AMP]: 7.63e-05 [WRES]:2.35e-06
   [IT]: 16 [c]: -0.00005 [a]:-0.000[AMP]: 3.81e-05 [WRES]:5.89e-05

 ----- VALORI DI  abs( e_(n+1)/e_n ) -----
ans =

    1.5000
    0.1667
    1.5000
    0.1667
    ....
    1.5000
    0.1667

  >>
```

La successione di rapporti $|e_{n+1}/e_n|$ *ovviamente non ha limite per* $n \to +\infty$ *poichè assume alternativamente valori* 1.5 *e* $1/6 \approx 0.1667$. *Quindi non si può affermare che l'ordine di bisezione sia* 1.

Tenendo in mente quanto appena visto, detto $|e_{n+1}|/|e_n|$ *il fattore di riduzione alla* n*-sima iterazione, e indicato come fattore di riduzione medio dopo* n *iterazioni la quantità*

$$k_n(e_0) = (\frac{|e_n|}{|e_{n-1}|} \cdots \frac{|e_1|}{|e_0|})^{1/n} = (\frac{|e_n|}{|e_0|})^{1/n}$$

verifichiamo che per $n \to \infty$ *il fattore di riduzione medio tende a* $1/2$ *(il che ovviamente non vuol dire che la riduzione sia uguale al massimo* $1/2$ *come erroneamente ritenuto da molti studenti).*

```
>> % e_n E' IL VETTORE IN MEMORIA DOPO test_bisezione.
>>
>> e0=abs(e_n(1));
>> en=abs(e_n(2:end));
>> for ii=1:length(en)
```

```
kn(ii)=(en(ii)/e0)^(1/ii);
end
>> kn'

ans =

    1.5000
    0.5000
    ....
    0.5000
    0.5441
    0.5000
    0.5380

>>
```

Per convincersene analiticamente, osserviamo che nel nostro esempio

- *se n è pari allora per un certo m si ha $n = 2m$ ed è*

$$k_{2m}(e_0) = ((3/2)^m (1/6)^m)^{1/(2m)} = ((3/12)^m)^{1/(2m)} = 1/2$$

- *se n è dispari allora per un certo m si ha $n = 2m + 1$ ed è dopo facili conti*

$$k_{2m+1} = (3/2)^{1/(2m+1)} (1/4^m)^{1/(2m+1)} = (3/2)^{1/(2m+1)} 2^{-(2m)/(2m+1)} \rightarrow 1/2.$$

E' d'altra parte evidentemente che il fattore di riduzione medio nulla ha a che fare con la classica media dei fattori di riduzione. Infatti

```
>> v=[    1.5000
    0.1667
    1.5000
    ....
    1.5000
    0.1667];
>>
>> %  CUMSUM Cumulative sum of elements.
>> % For vectors, CUMSUM(X) is a vector containing the cumulative sum of
>> % the elements of X.
>>
>> vv=cumsum(v)

vv =

    1.5000
    1.6667
    ....
   13.1669
   13.3336

>> % MEDIA DELLA SOMMA CUMULATIVA.
>>
>> vv./(1:length(vv))'
```

```
ans =

    1.5000
    0.8334
    ....
    0.8778
    0.8334

>>
```

Che in effetti questo sia il limite è di immediata verifica. Calcoliamo le medie al variare di n e vediamone il limite.

- *se n è pari allora per un certo m si ha $n = 2m$ ed è*

$$\textit{media}_{2m} = \frac{m \cdot (3/2) + m \cdot (1/6)}{2m} = \frac{(3/2) + (1/6)}{2} = 10/12 \approx 0.8334.$$

- *se n è dispari allora per un certo m si ha $n = 2m + 1$ ed è dopo facili conti*

$$\begin{aligned}\textit{media}_{2m+1} &= \frac{(m+1) \cdot (3/2) + m \cdot (1/6)}{2m+1} \\ &= \frac{(m+1) \cdot (3/2)}{2m+1} + \frac{m \cdot (1/6)}{2m+1} \to (1/2) \cdot (3/2) + (1/2) \cdot (1/6) \\ &= 10/12 \approx 0.8334. \end{aligned} \tag{3.6}$$

4. Metodo di Newton per la risoluzione di equazioni nonlineari. Supponiamo che f sia derivabile con continuità su un sottinsieme di $\mathbb{R}$ e sia $f^{(1)}(x)$ la derivata prima di f valutata nel generico punto x.

Il *metodo di Newton* genera la successione

$$x_{k+1} = x_k + h_k, \;\; h_k = -f(x_k)/f^{(1)}(x_k), \;\; k = 0, 1, \ldots \tag{4.1}$$

supposto che sia $f^{(1)}(x_k) \neq 0$ per $k = 0, 1, \ldots$.

Dal punto di vista algoritmico quanto visto pu'ò essere brevemente riassunto dal seguente pseudo-codice

ALGORITMO $y =$ `newton` (x,τ)

1. $y = x$.
2. Calcola $f^{(1)}(y)$.
3. Se $f^{(1)}(y)$ non è 0, poni $s = -f(y)/f^{(1)}(y)$.
4. $y = y + s$.
5. Do while $\|s\| > \tau$.
   (a) Calcola $f^{(1)}(y)$.
   (b) Se $f^{(1)}(y)$ non è 0, poni $s = -f(y)/f^{(1)}(y)$.
   (c) $y = y + s$.

dove x è il punto iniziale e τ una tolleranza prefissata.

4.1. Alcuni risultati sulla convergenza locale e globale di Newton. Per quanto riguarda la velocità di convergenza si dimostra il seguente teorema di *convergenza locale* (cf. [1], p. 60)

TEOREMA 4.1. *Sia $x^* \in (a, b)$ uno zero semplice di $f : [a, b] \to \mathbb{R}$. Si supponga inoltre $f \in C^2([a, b])$. Allora per $x_0 \in [a, b]$ sufficientemente vicino a x^* le iterazioni del metodo di Newton*

$$x_{k+1} = x_k - f(x_k)/f^{(1)}(x_k), \;\; k = 0, 1, 2, \ldots$$

sono ben definite e convergono quadraticamente a x^*.

DIMOSTRAZIONE. La dimostrazione consiste dapprima nel trovare un intervallo $\mathcal{I}_\infty$ contenente lo zero in cui la derivata esiste e poi un sottointervallo $\mathcal{I}_2$ per cui se $x_n \mathcal{I}_2$ allora l'iterazione x_{n+1} del metodo di Newton fa ancora parte di $\mathcal{I}_\in$. Da stime dell'errore in quest'intervallo si deduce la convergenza quadratica. Vediamo i dettagli della dimostrazione.

Essendo lo zero semplice, per definizione $f^{(1)}(x^*) \neq 0$. Di conseguenza esiste un intervallo simmetrico chiuso $\mathcal{I}_1 = [x^* - \delta, x^* + \delta]$ di x^* in cui $f^{(1)} \neq 0$. Osserviamo che dalla formula di Taylor centrata in x_n e dalla definizione della successione del metodo di Newton, se $x_n \in \mathcal{I}_1$, allora per un certo $\xi_n \in \mathcal{I}_1$ (che in effetti sta nel più piccolo intervallo aperto contenente x^* e x_n)

$$0 = f(x^*) = f(x_n) + f^{(1)}(x_n)(x^* - x_n) + f^{(2)}(\xi_n)(x^* - x_n)^2/2$$

e quindi dividendo ambo i membri per $f^{(1)}(x_n) \neq 0$ (poichè $x_n \in \mathcal{I}_1$), ricordando la definizione del metodo di Newton

$$\begin{aligned} 0 &= \frac{f(x_n)}{f^{(1)}(x_n)} + (x^* - x_n) + \frac{f^{(2)}(\xi_n)}{2 \cdot f^{(1)}(x_n)}(x^* - x_n)^2 \\ &= x^* - x_{n+1} + \frac{f^{(2)}(\xi_n)}{2 \cdot f^{(1)}(x_n)}(x^* - x_n)^2. \end{aligned} \tag{4.2}$$

Di conseguenza, posto $e_n = x_n - x^*$, da 4.2 abbiamo

$$|e_{n+1}| = |\frac{f^{(2)}(\xi_n)}{2 \cdot f^{(1)}(x_n)}||e_n|^2.$$

Per il teorema di Weierstrass esiste finito

$$C = \max_{x \in \mathcal{I}_1} |f^{(2)}(x)| / \min_{y \in I_1} |f^{(1)}(y)|.$$

Possiamo quindi affermare che se $x_n \in I_1$ allora

$$|e_{n+1}| \leq C|e_n|^2.$$

Supponiamo ora che per $\epsilon > 0$ sia $\gamma = \min(\delta, 1/C) - \epsilon > 0$ e che $x_0 \in \mathcal{I}_2 \equiv [x^* - \gamma, x^* + \gamma] \subseteq \mathcal{I}_1$ (si osservi che $\gamma < \delta$). Allora essendo $0 \leq C\gamma = C(\min(\delta, 1/C) - \epsilon) \leq 1 - C\epsilon < 1$ ricaviamo

$$|e_1| \leq C|e_0|^2 = (C|e_0|) \cdot |e_0| \leq (C\gamma)\gamma < \gamma$$

per cui tutte le iterazioni x_n del metodo di Newton stanno nell'intervallo $\mathcal{I}_2 = [x^* - \gamma, x^* + \gamma]$ ed essendo $\mathcal{I}_2 \subseteq \mathcal{I}_1$ la successione descritta dal metodo di Newton è ben definita. Anzi, da $x_0 \in \mathcal{I}_2$ implica $|e_0| \leq \gamma$, ed essendo

$$0 \leq C|e_{n+1}| \leq (C^2|e_n|^2) = (C|e_n|)^2 \leq \ldots \leq (C|e_0|)^{2^{n+1}}$$

con $C|e_0| \le C\gamma < 1$, necessariamente, per il teorema del confronto tra limiti, $|e_{n+1}| \to 0$ cioè $x_{n+1} \to x^*$. Inoltre si ha o che per un certo n^*, se $n \ge n^*$, è $x_n = x^*$ oppure dalla continuità di $f^{(1)}$ e $f^{(2)}$ nell'intervallo $\mathcal{I}_2 = [x^* - \gamma, x^* + \gamma]$

$$\lim_n \frac{|e_{n+1}|}{|e_n|^2} = \lim_n |\frac{f^{(2)}(\xi_n)}{2 \cdot f^{(1)}(x_n)}| = \frac{|f^{(2)}(x^*)|}{2 \cdot |f^{(1)}(x^*)|} \neq 0$$

cioè la convergenza è quadratica.
□

Particolarmente interessante il seguente teorema relativo alla *convergenza* del metodo di Newton (cf. [1, p.62]):

TEOREMA 4.2. *Sia $f \in C^2([a,b])$, con $[a,b]$ intervallo chiuso e limitato. Se*

1. $f(a) \cdot f(b) < 0$;
2. $f^{(1)}(x) > 0$, *per ogni* $x \in [a,b]$;
3. $f^{(2)}(x) > 0$

allora le iterate x_k fornite dal metodo di Newton sono strettamente decrescenti e convergono all'unica soluzione x^* in $[a,b]$, per $x_0 = b$.

NOTA 4.3. *Nelle ipotesi del Teorema 4.2, dal Teorema 4.1, si deduce che la convergenza è quadratica. Si sottolinea che se queste non sono verificate, non si può asserire che la convergenza sia quadratica (ad esempio se uno zero x^{ast} è multiplo allora la convergenza di Newton è solo lineare!).*

NOTA 4.4. *In questa nota consideriamo alcuni casi in cui nonostante il teorema 4.2 non sia direttamente applicabile si possa comunque giungere a provare la convergenza del metodo di Newton.*

Si supponga sia $f^{(1)}(x) < 0$, $f^{(2)}(x) > 0$ per ogni $x \in [a,b]$ e sia $g(x) := f(-x)$. E' chiaro che se $f(x^*) = 0$ si pone i $\tilde{x} = -x^*$ allora $g(\tilde{x}) = f(-\tilde{x}) = f(x^*) = 0$. Inoltre per $x \in [-b,-a]$ abbiamo $-x \in [a,b]$ e

$$g^{(1)}(x) = -f^{(1)}(-x) > 0,\ g^{(2)}(x) = f^{(2)}(-x) > 0.$$

Quindi, posto $x^{(0)} = -a$, le ipotesi del teorema valgono per $g(x)$ ed il metodo di Newton produce quindi una successione $\{x_k\}_k$ decrescente e convergente all'unica soluzione $\tilde{x}$ di $g(x) = 0$ in $[-b,-a]$. Essendo

$$x_{k+1} = x_k - g(x_k)/g^{(1)}(x_k) = x_k - f(-x_k)/(-f^{(1)}(-x_k)) = x_k + f(-x_k)/(f^{(1)}(-x_k))$$

posto $t^{(k)} = -x^{(k)}$ e moltiplicando entrambi i membri della precedente equazione per -1, ricaviamo che la successione

$$t_{k+1} = t_k - f(t_k)/(f^{(1)}(t_k)),\ t_0 = a$$

è crescente e convergente all'unica soluzione x^* di $f(x) = 0$ in $[a,b]$.

Analogamente, se $f^{(1)}(x) < 0$, $f^{(2)}(x) < 0$, al posto di $f(x) = 0$ basta studiare l'equazione $-f(x) = 0$. Se invece $f^{(1)}(x) > 0$, $f^{(2)}(x) < 0$ per ogni $x \in [a,b]$, allora bisogna studiare invece di $f(x) = 0$ l'equazione $h(x) := -f(-x) = 0$. Nuovamente se $h(\tilde{x}) = 0$ allora $-f(-\tilde{x}) = 0$ da cui $x^* = -\tilde{x}$.

Dimostriamo una interessante variante del precedente

TEOREMA 4.5. *Sia $f : [a,b] \to \mathbb{R}$ una funzione che gode delle seguente proprietà*

1. $f(a) < 0$ *e* $f(b) > 0$*;*
2. $f''(x) > 0$ *per ogni* $x \in [a, b]$.

Allora

1. *esiste uno ed un solo* $x^* \in (a, b)$ *tale che* $f(x^*) = 0$. *Tale zero è semplice.*
2. *per* $x_0 \in [x^*, b]$ *il metodo di Newton*

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

   è ben definito.
3. *La successione* $\{x_n\}$ *è decrescente e converge a* x^*.

DIMOSTRAZIONE.

1. Essendo $f''(x) > 0$ per ogni $x \in [a, b]$, allora la funzione f' è continua e monotona crescente in $[a, b]$. Di conseguenza,
   - se $f' > 0$ in $[a, b]$ si ha che pure f è monotona crescente e quindi in (a, b) c'è un unico zero x^* poichè $f(a) < 0$ e $f(b) > 0$;
   - se f' non ha segno costante in $[a, b]$ essendo monotona crescente ha un unico zero. Di conseguenza f ha un unico minimo $x_{\min} \in [a, b]$. Quindi dalla monotonia crescente di f' in $[x_{\min}, b]$ e dal fatto che $f(x_{\min}) \leq f(a)$ e $f(x_{\min}) \leq 0 < f(b)$ abbiamo che vi è un unico zero in $x^* \in [a, b]$.

   Osserviamo che per quanto appena visto, $f'(x) > 0$ per ogni $x \in [x^*, b]$ e quindi in particolare che lo zero x^* è semplice.
2. Questo risultato segue dal fatto che la funzione f' è strettamente positiva in $[x^*, b]$.
3. Sia $n \geq 0$ e $x_n \geq x^*$. Dalla formula di Taylor, essendo $f''(x_n) > 0$ deduciamo che per ξ_n nel più piccolo intervallo aperto contenente x_n e $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ si ha

$$\begin{aligned} f(x_{n+1}) &= f(x_n) + f'(x_n) \cdot (x_{n+1} - x_n) + f''(\xi_n) \cdot \frac{(\xi - x_n)^2}{2} \\ &> f(x_n) + f'(x_n) \cdot (x_{n+1} - x_n) = 0 = f(x^*). \end{aligned} \tag{4.3}$$

   Poichè per $x < x^*$ la funzione f è negativa, necessariamente ciò implica che $x_{n+1} > x^*$ Inoltre la successione $\{x_n\}$ è strettamente decrescente perchè da $f(x_n) > 0$ e $f'(x_n) > 0$ abbiamo

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \leq x_n.$$

   Poichè la successione $\{x_n\}$ è decrescente e limitata inferiormente da x^*, essa ammette limite ξ^*. Ma allora per la continuità di f e f'

$$\xi^* = \lim_n x_{n+1} = \lim_n x_n - \lim_n \frac{f(x_n)}{f'(x_n)} = \xi^* - \frac{f(\xi^*)}{f'(\xi^*)}$$

   ed essendo $f'(\xi^*) \neq 0$ poichè tale zero è semplice abbiamo $f(\xi^*) = 0$. Ma x^* è l'unico zero di f e quindi $\xi^* = x^*$ cioè il metodo di Newton converge decrescentemente a x^*.

□

Una versione modificata di questo teorema che stabilisce un teorema di convergenza globale si trova in [2], p. 419:

TEOREMA 4.6. *Sia* $f \in C^2([a, b])$*, con* $[a, b]$ *intervallo chiuso e limitato. Se*

1. $f(a) \cdot f(b) < 0$;
2. $f^{(1)}(x) \neq 0$, *per ogni* $x \in [a, b]$;
3. $f^{(2)}(x) \geq 0$ *o* $f^{(2)}(x) \leq 0$, *per ogni* $x \in [a, b]$;
4. $|f(a)/f^{(1)}(a)| < b - a$ *e* $|f(b)/f^{(1)}(b)| < b - a$,

allora il metodo di Newton converge all'unica soluzione x^* *in* $[a, b]$, *per ogni* $x_0 \in [a, b]$.
Osserviamo che

1. il metodo di Newton non ha sempre convergenza quadratica, come nel caso del problema $f(x) = 0$ in cui x^* sia uno zero avente molteplicità $p > 1$ cioè tale che

$$f(x^*) = f^{(1)}(x^*) = \ldots = f^{(p-1)}(x^*) = 0;$$

   quale esempio si ponga $f(x) = \log(x-1)(x-2)$; si verifica facilmente che $f(2) = f'(2) = 0$ e che la convergenza non è quadratica; per convincersene si applichi il metodo di Newton a tale esempio e si esegua il grafico dell'errore compiuto in scala semilogaritmica;

2. i due teoremi sopracitati forniscono delle condizioni necessarie per la convergenza; in realtà in letteratura ne esistono molte altre (cf. [3], [5], [18]).

4.2. Implementazione del metodo di Newton. Definiamo lo *scarto* (assoluto) di un metodo iterativo come

$$s_{n+1} = x_{n+1} - x_n. \tag{4.4}$$

Un test comunemente utilizzato è che sia

$$|s_{n+1}| \leq \tau \tag{4.5}$$

dove τ è una tolleranza prefissata dall'utente (ad esempio $\tau = 10^{-6}$).

Implementiamo il metodo di Newton mediante una function, usando come criteri di arresto lo scarto (4.4)-(4.5) ed il numero massimo di iterazioni. In seguito testiamo il metodo con diverse combinazioni del punto iniziale x_0, della tolleranza richiesta e del numero massimo di iterazioni e si commentiamo i risultati. Il codice fornisce, come output, i valori della successione e dello scarto ad ogni passo nonchè il numero di iterazioni.

Supponiamo di voler studiare l'equazione $f(x) = 0$ con

$$f(x) = \sin(x) - x$$

e quindi

$$f'(x) = \cos(x) - 1$$

Scriviamo in files `f.m`, `f1.m`, `newton.m` le seguenti funzioni:

```
function y=f(x)
y=sin(x)-x;
```

```
function y=f1(x)
y=cos(x)-1;
```

e sia

```
function [x,k,scarto,errflag]=newton(x0,tol,kmax,funct,dfunct)

%--------------------------------
% METODO DI NEWTON.
%--------------------------------
% INPUT.
%---------
% x0: punto iniziale.
% tol: tolleranza criterio dello step/residuo pesato.
% kmax: numero massimo di iterazioni.
% funct: funzione di cui cercare lo zero "funct(x)=0".
% funct: derivata di "funct".
%--------------------------------
% OUTPUT.
%---------
% x: vettore iterazioni del metodo di Newton.
% k: ultima iterazione.
% scarto: vettore degli scarti -fx/dfx.
% errflag: 0: uscita corretta, 1: uscita per errore.
%--------------------------------
% ESEMPIO.
%---------
% >>f=inline('x.^2-2');
% >>df=inline('2*x');
% >>[x,k,scarto,errflag]=newton(0.8,10^(-6),40,f,df)
% [ITER.]:   1[VALORE]: 0.80000 [ABS.SCARTO]: 8.50e-01
% [ITER.]:   2[VALORE]: 1.65000 [ABS.SCARTO]: 2.19e-01
% [ITER.]:   3[VALORE]: 1.43106 [ABS.SCARTO]: 1.67e-02
% [ITER.]:   4[VALORE]: 1.41431 [ABS.SCARTO]: 9.92e-05
% [ITER.]:   5[VALORE]: 1.41421 [ABS.SCARTO]: 3.48e-09
% x =
%
%     0.8000
%     1.6500
%     1.4311
%     1.4143
%     1.4142
%
% k =
%
%      5
%
% scarto =
%
%     0.8500
%    -0.2189
%    -0.0167
%    -0.0001
%    -0.0000
%
%  errflag =
%
```

```
%      0
%
%---------------------------------
% NOTA.
%---------
% IN ALCUNI MATLAB, SE QUANTO PROVATO DA' ERRORE
% ESEGUIRE
%
% [x,k,scarto,errflag]=newton(1,10^(-6),40,f,df)
%
%---------------------------------

%---------------------------------------
% PRIMA ITERAZIONE DEL METODO DI NEWTON.
%---------------------------------------
k=1;
x(1,1)=x0;

% VALUTAZIONE FUNZIONE.
fx=feval(funct,x(k,1));

if fx == 0 % x(1) E' SOLUZIONE.
    scarto(k,1)=0;
    errflag=0;
    return;
end

% VALUTAZIONE DERIVATA.
dfx=feval(dfunct,x(k,1));

if dfx == 0 % NEWTON NON PUO' PROSEGUIRE.
    scarto(k)=realmax;
    errflag=1;
    return;
else
    scarto(k,1)=-fx/dfx;
end

fprintf('\n \t [ITER.]: %3.0f',k);
fprintf('[VALORE]: %5.5f [ABS.SCARTO]: %2.2e',x(k,1),abs(scarto(k,1)));

%--------------------------------------------
% ITERAZIONI SUCCESSIVE DEL METODO DI NEWTON.
%--------------------------------------------
while (abs(scarto(k)) > tol) & (k < kmax)
    k=k+1;
    x(k,1)=x(k-1,1)+scarto(k-1,1);

    % VALUTAZIONE FUNZIONE.
    fx=feval(funct,x(k));

    if fx == 0 % x(1) E' SOLUZIONE.
        scarto(k,1)=0;
        errflag=0;
```

```
        return;
    end

    % VALUTAZIONE DERIVATA.
    dfx=feval(dfunct,x(k,1));

    if dfx == 0 % NEWTON NON PUO' PROSEGUIRE.
        scarto(k,1)=realmax;
        errflag=1;
        return;
    else
        scarto(k,1)=-fx/dfx;
    end

    fprintf('\n \t [ITER.]: %3.0f',k);
    fprintf('[VALORE]: %5.5f [ABS.SCARTO]: %2.2e',x(k),abs(scarto(k)));
end

% ANALISI DELL'USCITA DI NEWTON.
if (abs(scarto(k,1)) > tol) > 0
    errflag=1;
else
    errflag=0;
end
```

Quindi sul workspace di Matlab/Octave scriviamo

```
>> [x,k,scarto]=newton(1,10^(-6),20,'f','f1');

          [ITER.]:   1[VALORE]: 1.00000 [ABS.SCARTO]: 3.45e-01
          [ITER.]:   2[VALORE]: 0.65515 [ABS.SCARTO]: 2.22e-01
          [ITER.]:   3[VALORE]: 0.43359 [ABS.SCARTO]: 1.45e-01
          [ITER.]:   4[VALORE]: 0.28815 [ABS.SCARTO]: 9.63e-02
          [ITER.]:   5[VALORE]: 0.19183 [ABS.SCARTO]: 6.40e-02
          [ITER.]:   6[VALORE]: 0.12781 [ABS.SCARTO]: 4.26e-02
          [ITER.]:   7[VALORE]: 0.08518 [ABS.SCARTO]: 2.84e-02
          [ITER.]:   8[VALORE]: 0.05678 [ABS.SCARTO]: 1.89e-02
          [ITER.]:   9[VALORE]: 0.03785 [ABS.SCARTO]: 1.26e-02
          [ITER.]:  10[VALORE]: 0.02523 [ABS.SCARTO]: 8.41e-03
          [ITER.]:  11[VALORE]: 0.01682 [ABS.SCARTO]: 5.61e-03
          [ITER.]:  12[VALORE]: 0.01122 [ABS.SCARTO]: 3.74e-03
          [ITER.]:  13[VALORE]: 0.00748 [ABS.SCARTO]: 2.49e-03
          [ITER.]:  14[VALORE]: 0.00498 [ABS.SCARTO]: 1.66e-03
          [ITER.]:  15[VALORE]: 0.00332 [ABS.SCARTO]: 1.11e-03
          [ITER.]:  16[VALORE]: 0.00222 [ABS.SCARTO]: 7.38e-04
          [ITER.]:  17[VALORE]: 0.00148 [ABS.SCARTO]: 4.92e-04
          [ITER.]:  18[VALORE]: 0.00098 [ABS.SCARTO]: 3.28e-04
          [ITER.]:  19[VALORE]: 0.00066 [ABS.SCARTO]: 2.19e-04
          [ITER.]:  20[VALORE]: 0.00044 [ABS.SCARTO]: 1.46e-04
>>
```

che mostra una convergenza lenta, visto che la radice $x = 0$ è multipla (si osservi che $f^{(1)}(0) = \cos(0) - 1 = 0$)

NOTA 4.7. *Si osservi che si poteva fare a meno di scrivere le funzioni su file, lavorando direttamente da workspace di Matlab/Octave utilizzando il comando* `inline`

```
>> f=inline('sin(x)-x');
>> f1=inline('cos(x)-1');
>> [x,k,scarto]=newton(1,10^(-6),20,'f','f1');

          [ITER.]:   1[VALORE]: 1.00000 [ABS.SCARTO]: 3.45e-01
          [ITER.]:   2[VALORE]: 0.65515 [ABS.SCARTO]: 2.22e-01
          [ITER.]:   3[VALORE]: 0.43359 [ABS.SCARTO]: 1.45e-01
          [ITER.]:   4[VALORE]: 0.28815 [ABS.SCARTO]: 9.63e-02
          [ITER.]:   5[VALORE]: 0.19183 [ABS.SCARTO]: 6.40e-02
          [ITER.]:   6[VALORE]: 0.12781 [ABS.SCARTO]: 4.26e-02
          [ITER.]:   7[VALORE]: 0.08518 [ABS.SCARTO]: 2.84e-02
          [ITER.]:   8[VALORE]: 0.05678 [ABS.SCARTO]: 1.89e-02
          [ITER.]:   9[VALORE]: 0.03785 [ABS.SCARTO]: 1.26e-02
          [ITER.]:  10[VALORE]: 0.02523 [ABS.SCARTO]: 8.41e-03
          [ITER.]:  11[VALORE]: 0.01682 [ABS.SCARTO]: 5.61e-03
          [ITER.]:  12[VALORE]: 0.01122 [ABS.SCARTO]: 3.74e-03
          [ITER.]:  13[VALORE]: 0.00748 [ABS.SCARTO]: 2.49e-03
          [ITER.]:  14[VALORE]: 0.00498 [ABS.SCARTO]: 1.66e-03
          [ITER.]:  15[VALORE]: 0.00332 [ABS.SCARTO]: 1.11e-03
          [ITER.]:  16[VALORE]: 0.00222 [ABS.SCARTO]: 7.38e-04
          [ITER.]:  17[VALORE]: 0.00148 [ABS.SCARTO]: 4.92e-04
          [ITER.]:  18[VALORE]: 0.00098 [ABS.SCARTO]: 3.28e-04
          [ITER.]:  19[VALORE]: 0.00066 [ABS.SCARTO]: 2.19e-04
          [ITER.]:  20[VALORE]: 0.00044 [ABS.SCARTO]: 1.46e-04

>>
```

4.3. Esercizio sul calcolo delle radici quadrate. Il calcolo della radice quadrata di un numero reale $a \geq 0$ puó essere visto come il calcolo dell'unico zero positivo dell'equazione

$$x^2 - a = 0. \tag{4.6}$$

Posto $f(x) = x^2 - a$, essendo $f^{(1)}(x) = 2x$ da (4.1) otteniamo

$$x_{k+1} = x_k + h_k,\ h_k = -\frac{x_k^2 - a}{2\,x_k} = -\frac{x_k}{2} + \frac{a}{2\,x_k},\ k = 0, 1, \ldots \tag{4.7}$$

Scriviamo in files `rad2.m`, `drad2.m`, `newton.m` le seguenti funzioni:

```
function [fx]=rad2(x)
fx=x.^2-2;
```

```
function [fx]=drad2(x)
fx=2*x;
```

A questo punto scriviamo sul workspace di Matlab/Octave

```
[x,k,scarto]=newton(1.41,1e-8,50,'rad2','drad2');
```

Il metodo stampa sulla stessa workspace

```
[ITER.]:   1 [VALORE]: 1.41000 [ABS.SCARTO]: 4.22e-003
[ITER.]:   2 [VALORE]: 1.41422 [ABS.SCARTO]: 6.30e-006
[ITER.]:   3 [VALORE]: 1.41421 [ABS.SCARTO]: 1.40e-011
```

e quindi fornisce un'approssimazione della soluzione in sole 3 iterazioni se partiamo dal punto iniziale $x_0 = 1.41$.

Nota. Osserviamo che

1. posto $x^* := \sqrt{2}$;
2. scelti a e b cosicchè $0 < a < x^* := \sqrt{2}$, $b > x^*$;
3. $f'(x) = 2\,x \geq 0$, $f^{(2)}(x) = 2$ per $x \in [a, b]$.

Di conseguenza è possibile applicare il teorema 4.2, e si conseguenza se $x_0 = b$ il metodo di Newton converge a $x^* = \sqrt{(2)}$.

In altre parole, scelto arbitrariamente $x_0 > \sqrt{2}$ il metodo di Newton converge alla soluzione $\sqrt{2}$.
Da qui si vede con un po' di tecnica che la successione x_k converge a x_*.

NOTA 4.8. *Si osservi che il metodo di Newton permette di approssimare la radice quadrata di un numero nonnegativo, usando solamente le quattro operazioni* $+$, $-$, $*$ *e* $\backslash$.

4.4. Confronto tra il metodo di bisezione e di Newton. Finora abbiamo implementato mediante una function il metodo di bisezione, usando come criteri di arresto la semilunghezza dell'intervallo, l'approssimazione mediante rapporto incrementale del residuo pesato e il numero massimo di iterazioni.

Abbiamo successivamente eseguito degli esperimenti per esaminare gli zeri delle funzioni

1. $f(x) = x^2 - 2$ con $x_1 \in [1, 2]$;
2. $f(x) = \sin(x) - x = 0$

con particolari tolleranze richieste e numero massimo di iterazioni.

Plottiamo ora, in un unico grafico semi-logaritmico, l'andamento dell'errore relativo, e su workspace determiniamo la grandezza del residuo pesato relativo e della semi-lunghezza dell'intervallo relativa per il metodo di bisezione e dell'errore relativo e dello scarto relativo per il metodo di Newton applicati alla funzione

$$f(x) = x^2 - 2$$

(con punto iniziale $x_1 = 1$ per il metodo di Newton), con un numero di iterazioni pari a $k = 50$.

4.4.1. Commenti all'esercitazione.

1. Per far eseguire ai programmi esattamente 50 iterazioni, si può impostare il numero massimo di iterazioni a 50 e la tolleranza a 0.
2. Per disegnare il grafico degli errori, si esegua prima il codice relativo al metodo di Newton, avendo cura di usare nomi diversi per le due successioni generate. Conseguentemente, se `x` è il vettore contenente la successione prodotta dal metodo di bisezione e `y` quella prodotta dal metodo di Newton

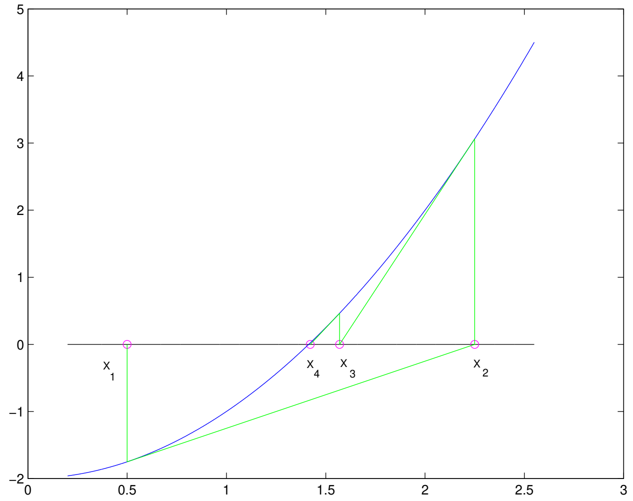

FIGURA 4.1. *Grafico che illustra geometricamente le iterazioni del metodo Newton per il calcolo della radice quadrata di 2.*

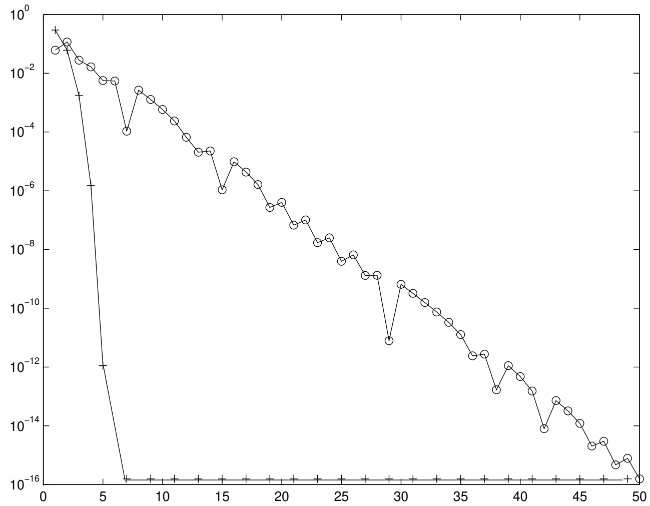

FIGURA 4.2. *Grafico che illustra l'errore relativo (rispetto la soluzione esatta $\sqrt{2}$) del metodo di bisezione e Newton, rappresentate rispettivamente da $\circ$ e $+$.*

```
>> [x,k,semilunghezza,residuopesato]=bisezione(1,2,0,0,50,'rad2');

 [IT]: 1 [c]:1.50000 [z1]:1.000 [AMP]:2.50e-001 [WRES]:1.00e-001
 [IT]: 2 [c]:1.25000 [z1]:1.250 [AMP]:1.25e-001 [WRES]:1.59e-001
 [IT]: 3 [c]:1.37500 [z1]:1.375 [AMP]:6.25e-002 [WRES]:3.80e-002
 [IT]: 4 [c]:1.43750 [z1]:1.375 [AMP]:3.13e-002 [WRES]:2.36e-002
 [IT]: 5 [c]:1.40625 [z1]:1.406 [AMP]:1.56e-002 [WRES]:7.90e-003
 [IT]: 6 [c]:1.42188 [z1]:1.406 [AMP]:7.81e-003 [WRES]:7.68e-003
  ....
 [IT]:49 [c]:1.41421 [z1]:1.414 [AMP]:8.88e-016 [WRES]:1.00e-015
```

```
 [IT]:50 [c]:1.41421 [z1]:1.414 [AMP]:4.44e-016 [WRES]:1.48e-016

>> [y,kn,scarto]=newton(1,0,50,'rad2','drad2');

 [ITER]:   1 [VALUE]: 1.00000 [STEP]: 5.00e-001
 [ITER]:   2 [VALUE]: 1.50000 [STEP]: 8.33e-002
 [ITER]:   3 [VALUE]: 1.41667 [STEP]: 2.45e-003
 [ITER]:   4 [VALUE]: 1.41422 [STEP]: 2.12e-006
 [ITER]:   5 [VALUE]: 1.41421 [STEP]: 1.59e-012
 [ITER]:   6 [VALUE]: 1.41421 [STEP]: 1.57e-016
  ....
 [ITER]:  50 [VALUE]: 1.41421 [STEP]: 1.57e-016

>> semilogy(abs(x-sqrt(2))/sqrt(2),'r-o'); hold on;
>> semilogy(abs(y-sqrt(2))/sqrt(2),'k-+'); hold on;
```

produce il grafico degli errori relativi dei due metodi. In rosso a tondini viene rappresentato l'errore assoluto della bisezione, in nero coi $+$ quello di Newton.

5. Esercizi per casa. Calcolare con i metodi di bisezione e Newton un'approssimazione dello zero α di

$$5x = \exp(-x) \tag{5.1}$$
$$x^4 - 2 = 0 \tag{5.2}$$
$$x \log(x) - 1 = 0 \tag{5.3}$$
$$(x-1)\log(x) = 0 \tag{5.4}$$
$$m = x - E\sin(x),\;\; m = 0.8,\; E = 2 \tag{5.5}$$

Verificare che per le approssimazioni $x^* \approx \alpha$ fornite sia effettivamente $f(x^*) \approx 0$. Posto come x^* il risultato finale, si disegni il grafico degli errori $|x^{(k)} - x^*|$ in scala semilogaritmica. Da questa dedurre, se possibile, l'ordine di convergenza del metodo e giustificare teoricamente la risposta.

6. Facoltativo: Metodo di punto fisso. Si supponga di dover calcolare α soluzione del problema $x = g(x)$ con $x \in [a,b]$ e ϕ sufficientemente derivabile in $[a,b]$. Si dimostra che

TEOREMA 6.1. *Si assuma che g sia derivabile con continuità in [a,b], che $g([a,b]) \subseteq [a,b]$ e che se $x \in [a,b]$ sia*

$$|g'(x)| \leq \lambda < 1.$$

Allora (cf. [1, p.80])

1. *$x = g(x)$ ha un'unica soluzione α in $[a,b]$;*
2. *per qualsiasi scelta di $x_0 \in [a,b]$, posto*

$$x_{n+1} = g(x_n)$$

la successione converge ad α;

3. *si ha*

$$|\alpha - x_n| \leq \frac{\lambda^n}{1-\lambda}|x_1 - x_0|.$$

Implementiamo la successione di punto fisso utilizzando il criterio di arresto dello *step* e approssimiamo con tale metodo la soluzione di

$$x = \frac{\exp(-x)}{5}$$

con una tolleranza di $10^{(-6)}$ partendo da $x_0 \in [0, 1]$ (a piacere).

```
function [x,k,scarto]=puntofisso(x0,tol,kmax,g)

% IL METODO CERCA DI APPROSSIMARE LA SOLUZIONE
% DEL PROBLEMA DI PUNTO FISSO "x=g(x)".

% PRIMA ITERAZIONE DEL METODO DI PUNTO FISSO x=g(x).
k=1;
x(1)=x0;
gx=feval(g,x(k));
scarto(k)=abs(gx-x(1));
fprintf('\n \t [ITER.]: %3.0f',k);
fprintf('[VALORE]: %5.5f [ABS.SCARTO]: %2.2e',x(k),abs(scarto(k)));

% ITERAZIONI SUCCESSIVE DEL METODO DI PUNTO FISSO.
while (abs(scarto(k)) > tol) & (k < kmax)
    k=k+1;
    x(k)=gx;
    gx=feval(g,x(k));
    scarto(k)=abs(gx-x(k));
    fprintf('\n \t [ITER.]: %3.0f',k);
    fprintf('[VALORE]: %5.5f [ABS.SCARTO]: %2.2e',x(k),abs(scarto(k)));
end

fprintf('\n');
```

Testiamo il nostro codice sull'esempio $x = \exp(-x)/5$

```
>> % DEFINISCO LA FUNZIONE COME INLINE.
>> h=inline('exp(-x)/5');
>>
>> % CALCOLO L'APPROSSIMAZIONE DELLA SOLUZIONE DI x=h(x).
>> [x,k,scarto]=puntofisso(0,10^(-6),100,h);

          [ITER.]:   1[VALORE]: 0.00000 [ABS.SCARTO]: 2.00e-01
          [ITER.]:   2[VALORE]: 0.20000 [ABS.SCARTO]: 3.63e-02
          [ITER.]:   3[VALORE]: 0.16375 [ABS.SCARTO]: 6.05e-03
          [ITER.]:   4[VALORE]: 0.16979 [ABS.SCARTO]: 1.02e-03
          [ITER.]:   5[VALORE]: 0.16877 [ABS.SCARTO]: 1.73e-04
          [ITER.]:   6[VALORE]: 0.16894 [ABS.SCARTO]: 2.92e-05
          [ITER.]:   7[VALORE]: 0.16891 [ABS.SCARTO]: 4.93e-06
          [ITER.]:   8[VALORE]: 0.16892 [ABS.SCARTO]: 8.33e-07
>>
>> % VERIFICO CHE x=h(x)
>> format long
>> x(k)
```

```
ans =

   0.168916686007286

>> feval(h,x(k))

ans =

   0.168915853145140

>> abs(feval(h,x(k))-x(k))

ans =

     8.328621461384245e-07

>>
```

6.0.2. Esercizio. Cosa si può dire relativamente alla sua convergenza? Per mostrare la convergenza del metodo di punto fisso, si può utilizzare il teorema 6.1 per $x_0 \in [0, 1]$?

6.0.3. Esercizio svolto. Risolvere col metodo di punto fisso l'equazione di Colebrook

$$\sqrt{x} = \frac{1}{-2\log_{10}\left(\frac{(\epsilon/D)}{3.7} + \frac{2.51}{\mathrm{Re}_D} \cdot \frac{1}{\sqrt{x}}\right)}$$

in cui il numero di Reynolds è $\mathrm{Re}_D = 10^5$ ed inoltre $\epsilon/D = 0.02$. La soluzione è $x \approx 0.0490$. Per prima cosa risolviamo l'equazione $t = C(t)$ dove

$$C(t) = \frac{1}{-2\log_{10}\left(\frac{(\epsilon/D)}{3.7} + \frac{2.51}{\mathrm{Re}_D} \cdot \frac{1}{t}\right)}$$

e una volta ottenuto t^* tale che $t^* = C(t^*)$ poniamo $x^* = (t^*)^2$.
Dopo aver effettuato

```
>> C=inline('-1./(2*log10(0.02/3.7+2.51./(10^5*x)))')

C =

     Inline function:
     C(x) = -1./(2*log10(0.02/3.7+2.51./(10^5*x)))

>> tt=0:0.01:10;
>> yy=feval(C,tt);
>> % GRAFICO DI "t-C(t)"
>> zz=tt-yy;
>> plot(tt,zz)
```

deduciamo che la soluzione t^* sta nell'intervallo $(0, 1)$ come si vede facilmente dal grafico.

Di conseguenza avremo

```
>> format long
```

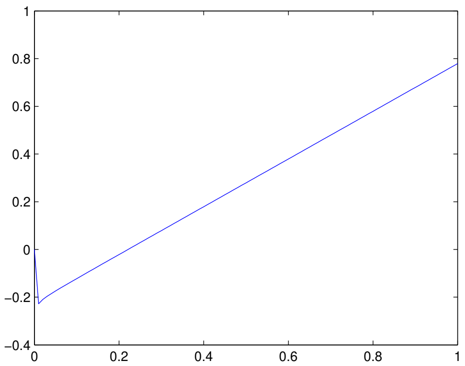

FIGURA 6.1. *Grafico della funzione $t - C(t)$ che mostra che $t - C(t) = 0$ per $t \approx 0.2$.*

```
>> [t,k,scarto]=puntofisso(0.2,10^(-13),100,C);

         [ITER.]:    1[VALORE]: 0.20000 [ABS.SCARTO]: 2.15e-02
         [ITER.]:    2[VALORE]: 0.22151 [ABS.SCARTO]: 9.40e-05
         [ITER.]:    3[VALORE]: 0.22142 [ABS.SCARTO]: 3.71e-07
         [ITER.]:    4[VALORE]: 0.22142 [ABS.SCARTO]: 1.47e-09
         [ITER.]:    5[VALORE]: 0.22142 [ABS.SCARTO]: 5.79e-12
         [ITER.]:    6[VALORE]: 0.22142 [ABS.SCARTO]: 2.29e-14

>> xsol=(t(k))^2

xsol =

   0.049026548280573

>>
```

7. Facoltativo: Metodo delle secanti. Uno dei metodi più comunemente utilizzati per la risoluzione di equazioni nonlineari è quello delle secanti, che nel caso di sistemi di equazioni nonlineari porta (non banalmente) al molto noto metodo di Broyden.

Il metodo delle secanti è definito dalla successione

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{7.1}$$

ottenuta dal metodo di Newton sostituendo $f'(x_n)$ col rapporto incrementale

$$\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Si nota subito che a differenza del metodo di Newton, quello delle secanti richiede due punti iniziali x_0, x_1 e non necessita ovviamente del calcolo della derivata f'.

Si dimostra che

TEOREMA 7.1. *Se $f \in C^2([a,b])$ con la radice $\alpha \in [a,b]$ e $f'(\alpha)$ allora se x_0, x_1 sono sufficientemente vicine ad α le iterate del metodo delle secanti convergono ad α, con ordine* *di convergenza*

$$p = \frac{(1+\sqrt{5})}{2} \approx 1.62.$$

Esercizio. Aiutandosi con quanto descritto nella routine `newton.m`, si implementi il metodo delle secanti e lo si usi per calcolare la radice quadrata di 2 con una tolleranza di $10^{(-15)}$. Quale criterio di arresto si usi quello step

$$|x_{n+1} - x_n| \leq \text{toll}.$$

Se eseguito correttamente si ottiene qualcosa del tipo:

```
>> [x,k,scarto]=secants(1,2,10^(-15),100,'rad2');

         [ITER.]:   1 [VALORE]: 2.00000 [ABS.SCARTO]: 1.00e+000
         [ITER.]:   2 [VALORE]: 1.33333 [ABS.SCARTO]: 6.67e-001
         [ITER.]:   3 [VALORE]: 1.40000 [ABS.SCARTO]: 6.67e-002
         [ITER.]:   4 [VALORE]: 1.41463 [ABS.SCARTO]: 1.46e-002
         [ITER.]:   5 [VALORE]: 1.41421 [ABS.SCARTO]: 4.23e-004
         [ITER.]:   6 [VALORE]: 1.41421 [ABS.SCARTO]: 2.12e-006
         [ITER.]:   7 [VALORE]: 1.41421 [ABS.SCARTO]: 3.16e-010
         [ITER.]:   8 [VALORE]: 1.41421 [ABS.SCARTO]: 3.14e-016
```

supposto che il valore alla prima iterazione sia quello di $x(2)$ che nell'esempio è 2. Calcoliamo una stima della velocità di convergenza

```
>> abs_scarto=abs(scarto);
>> L=length(abs_scarto);
>> ratios=abs_scarto(2:L)./abs_scarto(1:L-1);
>> L1=length(ratios);
>> p_approx=log(ratios(2:L1))./log(ratios(1:L1-1));
>> format long
>> p_approx
p_approx =
   5.67887358726758
   0.65854134728041
   2.33747974959565
   1.49349105655602
   1.66495845984246
   1.56815727154018
```

e quindi la velocità di convergenza stimata è prossima a quella teorica $(1+\sqrt{5})/2 \approx 1.62$.

Di seguito si esegua il calcolo della radice cubica del proprio numero di matricola.

8. Facoltativo: Un'equazione polinomiale. Si supponga di dover calcolare la più grande radice dell'equazione

$$x^6 - x + 1 = 0 \tag{8.1}$$

A tal proposito, aiutandosi con pico o l'editor preferito, poniamo `eqtype=1` in `g.m` e `dg.m`.

Eseguiamo un plot per avere un'idea della disposizione degli zeri. Da una regola dovuta a Cartesio, se il polinomio p di cui calcolare gli zeri z_i è

$$p(x) := \sum_{j=0}^{n} a_j x^j \tag{8.2}$$

allora

$$|z_i| \leq 1 + \max_{0 \leq i \leq n-1} \frac{|a_i|}{|a_n|} \tag{8.3}$$

Nel nostro caso, essendo

$$a_0 = 1,\ a_1 = -1,\ a_6 = 1,\ a_2 = \ldots = a_5 = 0$$

si ha quindi $|z_i| \leq 2$.
Eseguiamo dal workspace di Matlab/Octave il comando

```
>> x=-2:0.001:2; y=g(x); plot(x,y,'r-'); hold on;
>> z=zeros(size(y)); plot(x,z,'g-'); hold on;
```

Come risultato abbiamo il grafico in figura. Da questo si vede che la radice (positiva) cercata risulta essere un numero nell'intervallo [1.0,1.5]. Zoomando più volte si vede che appartiene all'intervallo [1.11,1.16]. Lanciamo quindi il metodo di bisezione (se ne osservi l'applicabilità !) ottenendo

```
>> [c,k,semilunghezza,residuopesato]=bisezione(1.12,1.16,10^(-8),10^(-8),50,'g');

 [IT]:   1 [c]: 1.14000 [z1]: 1.120 [AMP]: 1.00e-002 [WRES]: 5.47e-003
 [IT]:   2 [c]: 1.13000 [z1]: 1.130 [AMP]: 5.00e-003 [WRES]: 4.66e-003
 [IT]:   3 [c]: 1.13500 [z1]: 1.130 [AMP]: 2.50e-003 [WRES]: 2.79e-004
 [IT]:   4 [c]: 1.13250 [z1]: 1.133 [AMP]: 1.25e-003 [WRES]: 2.22e-003
 [IT]:   5 [c]: 1.13375 [z1]: 1.134 [AMP]: 6.25e-004 [WRES]: 9.73e-004
 [IT]:   6 [c]: 1.13438 [z1]: 1.134 [AMP]: 3.12e-004 [WRES]: 3.49e-004
 [IT]:   7 [c]: 1.13469 [z1]: 1.135 [AMP]: 1.56e-004 [WRES]: 3.66e-005
 [IT]:   8 [c]: 1.13484 [z1]: 1.135 [AMP]: 7.81e-005 [WRES]: 1.20e-004
 [IT]:   9 [c]: 1.13477 [z1]: 1.135 [AMP]: 3.91e-005 [WRES]: 4.15e-005
 [IT]:  10 [c]: 1.13473 [z1]: 1.135 [AMP]: 1.95e-005 [WRES]: 2.42e-006
 [IT]:  11 [c]: 1.13471 [z1]: 1.135 [AMP]: 9.77e-006 [WRES]: 1.71e-005
 [IT]:  12 [c]: 1.13472 [z1]: 1.135 [AMP]: 4.88e-006 [WRES]: 7.34e-006
 [IT]:  13 [c]: 1.13472 [z1]: 1.135 [AMP]: 2.44e-006 [WRES]: 2.46e-006
 [IT]:  14 [c]: 1.13472 [z1]: 1.135 [AMP]: 1.22e-006 [WRES]: 1.73e-008
 [IT]:  15 [c]: 1.13473 [z1]: 1.135 [AMP]: 6.10e-007 [WRES]: 1.20e-006
 [IT]:  16 [c]: 1.13472 [z1]: 1.135 [AMP]: 3.05e-007 [WRES]: 5.93e-007
 [IT]:  17 [c]: 1.13472 [z1]: 1.135 [AMP]: 1.53e-007 [WRES]: 2.88e-007
 [IT]:  18 [c]: 1.13472 [z1]: 1.135 [AMP]: 7.63e-008 [WRES]: 1.35e-007
 [IT]:  19 [c]: 1.13472 [z1]: 1.135 [AMP]: 3.81e-008 [WRES]: 5.90e-008
 [IT]:  20 [c]: 1.13472 [z1]: 1.135 [AMP]: 1.91e-008 [WRES]: 2.08e-008

>>
```

Desiderando avere almeno 8 cifre decimali come richiesto dalla tolleranza, dopo aver ricordato che c è un vettore con la *storia* dell'approssimazione dello zero cercato, come si evince da

```
>> c

c =
  Columns 1 through 7
    1.1400    1.1300    1.1350    1.1325    1.1338    1.1344    1.1347
```

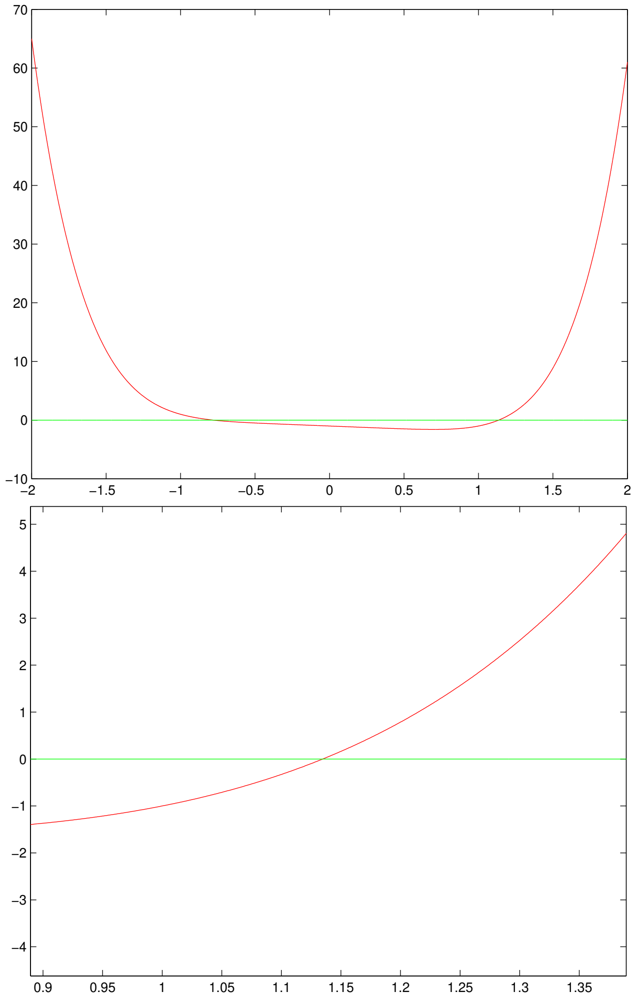

FIGURA 8.1. *Grafico del polinomio* $x^6 - x - 1$.

```
  Columns 8 through 14
    1.1348    1.1348    1.1347    1.1347    1.1347    1.1347    1.1347
  Columns 15 through 21
    1.1347    1.1347    1.1347    1.1347    1.1347    1.1347    1.1347

>>
```

digitiamo

```
>> format long; M=length(c); risultato=c(M)

risultato =
   1.13472414016724

>>
```

Se intendiamo usare il metodo di Newton, bisogna calcolare la derivata di $g(x) := x^6 - x - 1$ che è uguale a $Dg(x) := 6\,x^5 - 1$. Questa è implementata nel file `dg.m` e basta settare il parametro `eqtype=1;` per poterla valutare.
Dopo aver fatto questa modifica, testiamo il metodo di Newton con punto iniziale $x_0 = 1.12$, tolleranza 10^{-8} e numero massimo di iterazioni uguale a 50, ottenendo:

```
>> [x,k,scarto,errflag]=newton(1.12,1e-8,50,'g','dg');

 [ITER]:   1 [VALUE]: 1.12000 [STEP]: 1.53e-002
 [ITER]:   2 [VALUE]: 1.13527 [STEP]: 5.43e-004
 [ITER]:   3 [VALUE]: 1.13472 [STEP]: 7.14e-007
 [ITER]:   4 [VALUE]: 1.13472 [STEP]: 1.23e-012

>> format long;
>> x

x =
   1.12000000000000   1.13526807498409   1.13472485264634   1.13472413840275

>> M=length(x); risultato=x(M)

risultato =
   1.13472413840275

>>
```

8.1. Facoltativo: Un esempio sul calcolo dello zero di una funzione. Consideriamo la funzione

$$f_2(x) := x - 6.28 - \sin(x)$$

e supponiamo di doverne calcolare uno zero x^*. Notiamo che la soluzione non è 2π. Infatti

$$f_2(2\pi) := 2\pi - 6.28 - \sin(2\pi) \approx 0.00318530717959$$

come si vede dal workspace di Matlab/Octave

```
>> format long
>> 2*pi-6.28-sin(2*pi)
ans =
   0.00318530717959
>>
```

Essendo $\sin(x) \in [-1, 1]$ necessariamente $6.28 + \sin(x)$ sta nell'intervallo $[5.28, 7.28]$ e visto che

$$x - 6.28 - \sin(x) = 0 \Leftrightarrow x = 6.28 + \sin(x)$$

deduciamo che $x^* \in [5.28, 7.28]$. Eseguiamo quindi un plot in tale intervallo. Dopo aver settato `eqtype=2` tanto in `g` quanto in `dg`, digitiamo nel workspace di Matlab/Octave

```
>> x=5.28:0.001:7.28; y=g(x); plot(x,y); hold on;
>> x=5.28:0.001:7.28; y=zeros(size(x)); plot(x,y,'g-'); hold on;
```

(si confronti con la ralativa figura). Dopo qualche zoom intorno allo zero della funzione, si evince che lo zero cercato è nell'intervallo $[6, 6.04]$.
Dal grafico nell'intervallo più ampio, si vede che il metodo delle tangenti (vederlo geometricamente!) converge partendo da $x_0 = 4$. Otteniamo quale risultato

```
>> [x,k,scarto,errflag]=newton(4,1e-8,50,'g','dg');

 [ITER]:   1 [VALUE]: 4.00000 [STEP]: 9.21e-001
 [ITER]:   2 [VALUE]: 4.92112 [STEP]: 4.80e-001
 [ITER]:   3 [VALUE]: 5.40118 [STEP]: 2.93e-001
 [ITER]:   4 [VALUE]: 5.69428 [STEP]: 1.80e-001
 [ITER]:   5 [VALUE]: 5.87397 [STEP]: 9.86e-002
 [ITER]:   6 [VALUE]: 5.97256 [STEP]: 3.73e-002
 [ITER]:   7 [VALUE]: 6.00988 [STEP]: 5.51e-003
 [ITER]:   8 [VALUE]: 6.01539 [STEP]: 1.14e-004
 [ITER]:   9 [VALUE]: 6.01550 [STEP]: 4.85e-008
 [ITER]:  10 [VALUE]: 6.01550 [STEP]: 7.79e-015

>>
```

il che mostra la convergenza piuttosto lenta del metodo. Similmente, il metodo converge pure partendo da $x_0 = 8$, ma nuovamente i risultati non sono entusiasmanti

```
>> [x,k,scarto,errflag]=newton(8,1e-8,50,'g','dg');

 [ITER]:   1 [VALUE]: 8.00000 [STEP]: 6.38e-001
 [ITER]:   2 [VALUE]: 7.36216 [STEP]: 3.80e-001
 [ITER]:   3 [VALUE]: 6.98191 [STEP]: 2.50e-001
 [ITER]:   4 [VALUE]: 6.73155 [STEP]: 1.83e-001
 [ITER]:   5 [VALUE]: 6.54886 [STEP]: 1.80e-001
 [ITER]:   6 [VALUE]: 6.36930 [STEP]: 8.88e-001
 [ITER]:   7 [VALUE]: 5.48106 [STEP]: 2.63e-001
 [ITER]:   8 [VALUE]: 5.74386 [STEP]: 1.59e-001
 [ITER]:   9 [VALUE]: 5.90295 [STEP]: 8.28e-002
 [ITER]:  10 [VALUE]: 5.98571 [STEP]: 2.69e-002
 [ITER]:  11 [VALUE]: 6.01264 [STEP]: 2.84e-003
 [ITER]:  12 [VALUE]: 6.01547 [STEP]: 3.01e-005
 [ITER]:  13 [VALUE]: 6.01550 [STEP]: 3.35e-009

>>
```

Dalle ultime iterazioni si capisce che la convergenza è comunque superlineare.

8.2. Facoltativo. Zeri multipli di una funzione e metodo di Newton: un esempio. Abbiamo visto che il metodo di Newton non ha convergenza quadratica nel caso del problema $f(x) = 0$ in cui x^* sia uno zero multiplo di molteplicità $p > 1$ cioè tale che

$$f(x^*) = f^{(1)}(x^*) = \ldots = f^{(p-1)}(x^*) = 0.$$

Per esempio, si calcolino con il metodo di Newton gli zeri multipli della funzione

$$f_3(x) := \exp(x^2) - 1$$

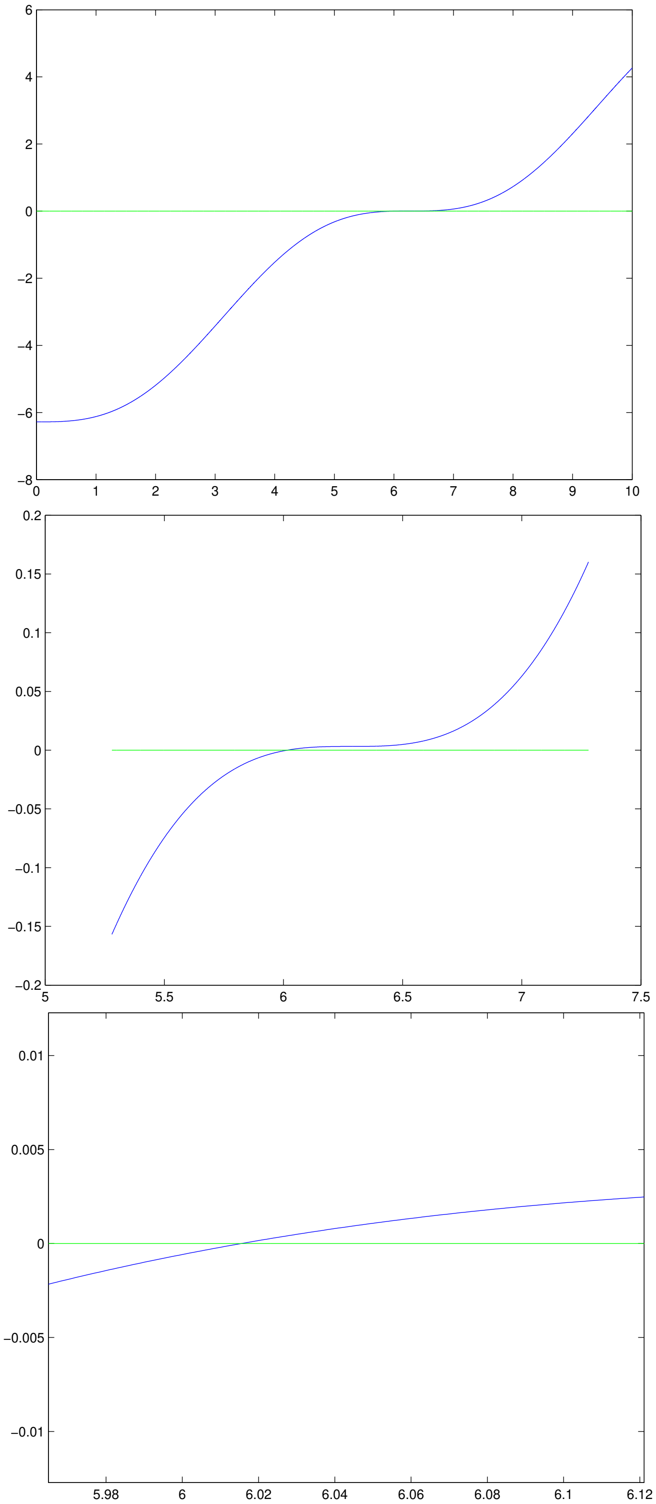

FIGURA 8.2. *Grafico del polinomio* $x - 6.28 - \sin(x)$ *su varie scale, vicino alla radice* $x^* \approx 6.01$.

implementata in `g.m` con `eqtype=3` (ricordarsi di settare pure `dg.m`!). La funzione ha uno zero multiplo in $x^* = 0$, essendo

$$Df_3(x) := 2x\exp(x^2), \ f_3(x^*) = Df_3(x^*) = 0.$$

D'altra parte

$$f_3(x) := \exp(x^2) - 1 = 0 \Leftrightarrow \exp(x^2) = 1 \Leftrightarrow x^2 = \log(\exp(x^2)) = \log(1) = 0$$

il he nuovamente mostra che la radice $x = 0$ è multipla (calcolare la derivata prima di f nella radice $x = 0$).

Dal plot dell'errore del metodo in scala semi-logaritmica, valutare l'ordine di convergenza del metodo.

Risoluzione. Settando `eqtype=3` in `g.m` e `dg.m`, lanciamo il metodo di Newton per diversi valori iniziali. Il metodo non è globalmente convergente. Ad esempio:

```
>> [x,k,scarto,errflag]=newton(8,1e-8,50,'g','dg');

 [ITER]:   1 [VALUE]: 8.00000 [STEP]: 5.44e+027
 [ITER]:   2 [VALUE]: -5443167958782763500000000000.00000 [STEP]: Inf
 [ITER]:   3 [VALUE]:  -Inf [STEP]: NaN

>>
```

Una radice è naturalmente $x^* = 0$, ed essendo $\exp(x^2) > 1$ se $x \neq 0$, essa è pure l'unica. Come visto è doppia (ha cioè molteplicità 2), e dalla teoria si può provare che di conseguenza la convergenza di Newton è lineare. Vediamolo direttamente partendo da $x_0 = 1$.

```
>> [x,k,scarto,errflag]=newton(1,1e-8,50,'g','dg');

 [ITER]:   1 [VALUE]: 1.00000 [STEP]: 3.16e-001
 [ITER]:   2 [VALUE]: 0.68394 [STEP]: 2.73e-001
 [ITER]:   3 [VALUE]: 0.41081 [STEP]: 1.89e-001
 [ITER]:   4 [VALUE]: 0.22180 [STEP]: 1.08e-001
 [ITER]:   5 [VALUE]: 0.11359 [STEP]: 5.64e-002
 [ITER]:   6 [VALUE]: 0.05716 [STEP]: 2.85e-002
 [ITER]:   7 [VALUE]: 0.02863 [STEP]: 1.43e-002
 [ITER]:   8 [VALUE]: 0.01432 [STEP]: 7.16e-003
 [ITER]:   9 [VALUE]: 0.00716 [STEP]: 3.58e-003
 [ITER]:  10 [VALUE]: 0.00358 [STEP]: 1.79e-003
 [ITER]:  11 [VALUE]: 0.00179 [STEP]: 8.95e-004
 [ITER]:  12 [VALUE]: 0.00090 [STEP]: 4.48e-004
 [ITER]:  13 [VALUE]: 0.00045 [STEP]: 2.24e-004
 [ITER]:  14 [VALUE]: 0.00022 [STEP]: 1.12e-004
 [ITER]:  15 [VALUE]: 0.00011 [STEP]: 5.59e-005
 [ITER]:  16 [VALUE]: 0.00006 [STEP]: 2.80e-005
 [ITER]:  17 [VALUE]: 0.00003 [STEP]: 1.40e-005
 [ITER]:  18 [VALUE]: 0.00001 [STEP]: 6.99e-006
 [ITER]:  19 [VALUE]: 0.00001 [STEP]: 3.50e-006
 [ITER]:  20 [VALUE]: 0.00000 [STEP]: 1.75e-006
 [ITER]:  21 [VALUE]: 0.00000 [STEP]: 8.74e-007
 [ITER]:  22 [VALUE]: 0.00000 [STEP]: 4.37e-007
 [ITER]:  23 [VALUE]: 0.00000 [STEP]: 2.18e-007
 [ITER]:  24 [VALUE]: 0.00000 [STEP]: 1.09e-007
 [ITER]:  25 [VALUE]: 0.00000 [STEP]: 5.48e-008
```

```
 [ITER]:  26 [VALUE]: 0.00000 [STEP]: 2.64e-008
 [ITER]:  27 [VALUE]: 0.00000 [STEP]: 1.57e-008
 [ITER]:  28 [VALUE]: 0.00000 [STEP]: 8.85e-009

>>
```

Vediamo direttamente l'ordine di convergenza. Visto che la soluzione è $x^* = 0$ necessariamente

$$e_k = |x^{(k)} - x^*| = |x^{(k)}|.$$

Se il metodo ha ordine p, per k sufficientemente grande, esiste C indipendente da k tale che

$$e_{k+1} \approx C e_k^p$$

$$e_{k+2} \approx C e_{k+1}^p$$

e quindi dividendo membro a membro abbiamo

$$\frac{e_{k+2}}{e_{k+1}} \approx \left(\frac{e_{k+1}}{e_k}\right)^p.$$

Calcolando il logaritmo di ambo i membri

$$\log\left(\frac{e_{k+2}}{e_{k+1}}\right) \approx \log\left(\frac{e_{k+1}}{e_k}\right)^p = p\log\left(\frac{e_{k+1}}{e_k}\right)$$

da cui

$$p \approx \frac{\log\left(\frac{e_{k+2}}{e_{k+1}}\right)}{\log\left(\frac{e_{k+1}}{e_k}\right)}.$$

Nel nostro caso, essendo $e_k = |x^{(k)}|$ abbiamo

$$p \approx \frac{\log\left(\frac{|x^{(k+2)}|}{|x^{(k+1)}|}\right)}{\log\left(\frac{|x^{(k+1)}|}{|x^{(k)}|}\right)}.$$

Dal workspace di Matlab/Octave

```
>> absx=abs(x);
>> L=length(absx);
>> ratios=absx(2:L)./absx(1:L-1);
>> L1=length(ratios);
>> p_approx=log(ratios(2:L1))./log(ratios(1:L1-1));
>> format long
>> p_approx'

ans =

   1.34180444561870
   1.20915162745740
```

```
   1.08581578729537
   1.02616296799749
   1.00694914160602
   1.00176551299514
   1.00044319047015
   1.00011091165740
   1.00002773504230
   1.00000693421959
   1.00000173374970
   1.00000043279671
   1.00000011084991
   1.00000003726416
   0.99999996879110
   1.00000015214145
   0.99999983966276
   1.00000320369014
   0.99999323164192
   1.00004521264807
   0.99976874978756
   0.99962083100483
   0.99886365484392
   1.00329108837037
   0.95044659133176
   1.23127847870172

>>
```

da cui si evince che effettivamente la convergenza è lineare.

8.3. Facoltativo: Function `fsolve`. Si testi la funzione `fsolve` di Matlab o GNU Octave sui casi precedenti, aiutandosi con gli esempi forniti dal comando `help -i fsolve`. In GNU-Octave (con cygwin, su Windows XP) versione 2.1.73 abbiamo

```
octave:1> [x,info,msg]=fsolve('rad2',1);
warning: in /usr/lib/octave/2.1.73/oct/i686-pc-cygwin/fsolve.oct
near line 14, column 13:
warning: time stamp for `/cygdrive/d/CORSI_MATLAB/MIEI/SM_EQNON
LINEARI_2007/MFILES/rad2.m' is in the future
octave:2> x
x = 1.4142
```

In Matlab 6.1.0.450

```
>> fun = inline('x.^2-2');
>> x = fsolve(fun,[1 2],optimset('Display','off'));
>> format long;
>> x
x =
   1.41421356237470   1.41421356237469
```

9. Facoltativo. Nota storica. Il metodo di Newton è anche noto come metodo di Newton-Raphson, in quanto sviluppato da Newton nell'opera *Analysis Aequationes Numero Terminorum Infinitas* (1666, scritto nel 1671 e quindi pubblicato nel 1711 da William Jones)

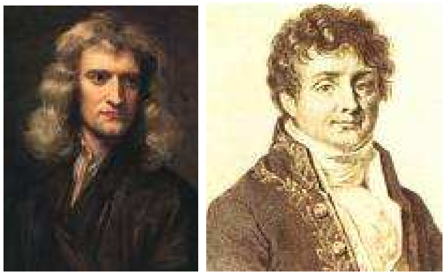

FIGURA 9.1. *Ritratti di Newton (1642-1727) e Fourier (1768-1830).*

per calcolare la radice $2.09455148154233\ldots$ di $x^3 - 2\,x - 5$. Il metodo forse derivava da alcune idee di Francois Viete che a sua volta si rifaceva al lavoro del matematico persiano Sharaf al-Din al-Tusi. Erone di Alexandria a sua volta utilizzava idee in qualche modo simili.

La questione venne ripresa nel *De metodis fluxionum et serierum infinitarum* (scritto nel 1671, tradotto e pubblicato come *Method of Fluxions* nel 1736 da John Colson). Il metodo descritto era molto differente da quello attuale, e veniva sostanzialmente applicato a polinomi [2, p. 413], [16].

Il metodo di Newton venne pure pubblicato nel *A Treatise of Algebra both Historical and Practical* (1685) di John Wallis.

Nel 1690, Joseph Raphson pubblicò nel *Analysis aequationum universalis* una versione maggiormente algoritmica, iterativa e meno complicata di quella di Newton. Quindi Simpson, nel 1740, generalizzò queste idee per equazioni nonlineari e sistemi nonlineari di piccole dimensioni.

Nel 1768, Jean-Raymond Mourraille nel *Traité de la ré solution des é quation numé riques* cominciò a discutere la convergenza del metodo di Newton, sottolineando il legame con l'aspetto geometrico delle tangenti.

Pure Fourier diede impulso a questi studi nel *Question d'analyse algebrique* (1818).

10. Esercizio facoltativo: il metodo di Halley. Il metodo di Halley [15] approssima uno zero x^* dell'equazione $f(x) = 0$ con la sequenza di iterazioni

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)}$$

a partire da un punto iniziale x_0. Ovviamente, si richiede l'esistenza della derivata seconda.

Se f è differenziabile tre volte con continuità e x^* non è uno zero delle sue derivate, allora si può dimostrare (non immediato!) che in un certo intorno di x^* le iterate x_n soddisfano

$$|x_{n+1} - x^*| \leq K \cdot |x_n - x^*|^3, \text{ per qualche } K > 0.$$

cioè con convergenza cubica.
Quale esercizio si

1. approssimi col metodo di Halley lo zero di

   $$f_2(x) := x - 6.28 - \sin(x),$$

   partendo da $x_0 = 4$;

2. approssimi col metodo di Halley la radice quadrata di 2 partendo da $x_0 = 1$;
3. approssimi col metodo di Halley la radice cubica di 2 partendo da $x_0 = 1$.

interrompendo il processo quando lo step $|x_{n+1} - x_n|$ è inferiore di 10^{-14}. Si paragoni quindi il metodo di Halley con quello di Newton, sugli stessi problemi: quale dei due sembra convergere più velocemente?

11. Alcuni esercizi facoltativi.

1. Scritto un file `g.m`

```
function y=g(x)

eqtype=1;

switch eqtype
case 1
    y=x.^6-x-1;
case 2
    y=x-6.28-sin(x);
case 3
    y=exp(x.^2)-1;
end
```

   e `dg.m` relativo alle corrispondenti derivate

```
function y=dg(x)

eqtype=1;

switch eqtype
case 1
    y=6*(x.^5)-1;
case 2
    y=1-cos(x);
case 3
    y=2*x.*exp(x.^2);
end
```

   si studi l'equazione $g(x) = 0$ al variare di `eqtype` tra 1 e 3.

2. Si risolva numericamente il problema di calcolare la radice cubica di un numero, col metodo di Newton.
3. Si risolva numericamente il problema di calcolare lo zero di $f(x) = \arctan(x)$, col metodo di Newton partendo prima da $x_0 = 1.2$ ed una seconda volta da $x_0 = 1.4$. Convergono in entrambi casi?
4. Si calcoli il reciproco di un numero c, risolvendo l'equazione $\frac{1}{x} - c = 0$ col metodo di Newton. Osservare che questo era utile per vecchi computers che non avevano un chip che faceva la divisione, bensì uno che faceva la moltiplicazione (cf. [4, p.35-39]).
5. Nota la soluzione del problema $x^2 - a = 0$ (si ottiene con il comando `sqrt(a)`) si stimi l'ordine di convergenza del metodo di Newton (4.7) per il calcolo della

radice quadrata di un numero. A questo proposito conoscendo $|e_{k+1}|$, $|e_k|$, $|e_{k-1}|$ e ritenendo costante la costante asintotica di errore C calcolare un'approssimazione di p, via il calcolo di un logaritmo. Si ricordi che

$$p \approx \frac{\log\left(\frac{|e^{(k+2)}|}{|e^{(k+1)}|}\right)}{\log\left(\frac{|e^{(k+1)}|}{|e^{(k)}|}\right)}.$$

11.1. Online. Per approfondimenti, si considerino le pagine web

1. http://it.wikipedia.org/wiki/Calcolo_dello_zero_di_una_funzione
2. http://it.wikipedia.org/wiki/Metodo_della_bisezione
3. http://it.wikipedia.org/wiki/Metodo_delle_tangenti
4. http://en.wikipedia.org/wiki/Newton's_method
5. http://en.wikipedia.org/wiki/Joseph_Raphson
6. http://www-history.mcs.st-andrews.ac.uk/Biographies/Fourier.html
7. http://www-history.mcs.st-andrews.ac.uk/Biographies/Newton.html
8. http://www-history.mcs.st-andrews.ac.uk/Biographies/Raphson.html

12. Frasi celebri.

1. Mathematical Analysis is as extensive as nature herself. (Fourier)
2. Laplace seems quite young; his voice is quiet but clear, and he speaks precisely, though not very fluently; his appearance is pleasant, and he dresses very simply; he is of medium height. His teaching of mathematics is in no way remarkable and he covers the material very rapidly. (Fourier)
3. Monge has a loud voice and he is energetic, ingenious and very learned. It is well known that his talent is particularly for geometry, physics and chemistry. The subject that he teaches is a fascinating one, and he describes it with the greatest possible clarity. He is even considered to be too clear, or, rather to deal with his material too slowly. He gives individual practical lessons to his students. He speaks colloquially, and for the most part precisely. He is not only to be considered for his great knowledge but is also greatly respected in public and in private. His appearance is very ordinary. (Fourier)
4. Lagrange, the foremost scholar of Europe, appears to be between 50 and 60 years old, though he is in fact younger; he has a strong Italian accent and pronounces an 's' as if it were a 'z'; he dresses very quietly, in black or brown; he speaks colloquially and with some difficulty, with the hesitant simplicity of a child. Everyone knows that he is an extraordinary man, but one needs to have seen him to recognise him as a great one. He speaks only in discussion, and some of what he says excites ridicule. The other day he said There are a lot of important things to be said on this subject, but I shall not say them. The students are incapable of appreciating his genius, but the teachers make up for that. (Fourier)
5. In the absence of any other proof, the thumb alone would convince me of God's existence. (Newton)

6. Truth is ever to be found in the simplicity, and not in the multiplicity and confusion of things. (Newton)
7. Tact is the art of making a point without making an enemy. (Newton)
8. We build too many walls and not enough bridges. (Newton)
9. Oh Diamond! Diamond! Thou little knowest the mischief done! (Said to a pet dog who knocked over a candle and set fire to his papers). (Newton)
10. Gravitation is not responsible for people falling in love. (Newton?)

RIFERIMENTI BIBLIOGRAFICI

[1] K. Atkinson, *An Introduction to Numerical Analysis*, Wiley, (1989).
[2] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, McGraw-Hill, (1990).
[3] J.E. Dennis, Jr. e R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, (1996).
[4] G. Gambolati, *Elementi di Calcolo Numerico*, Edizioni Libreria Cortina, (1988).
[5] C.T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM Frontiers in Applied Mathematics. SIAM, Philadelphia, (1995).
[6] A. Quarteroni, F. Saleri *Introduzione al Calcolo Scientifico*, Springer, (2002).
[7] Mac Tutor (Fourier), http://www-history.mcs.st-andrews.ac.uk/Biographies/Fourier.html.
[8] Mac Tutor (Newton), http://www-history.mcs.st-andrews.ac.uk/Biographies/Newton.html.
[9] Mac Tutor (Raphson), http://www-history.mcs.st-andrews.ac.uk/Biographies/Raphson.html.
[10] G. Rodriguez, *Algoritmi Numerici*, Pitagora Editrice, (2008).
[11] Wikipedia (Newton), http://en.wikiquote.org/wiki/Newton.
[12] Wikipedia (Calcolo dello zero di una funzione), http://it.wikipedia.org/wiki/Calcolo_dello_zero_di_una_funzione.
[13] Wikipedia (Metodo della bisezione), http://it.wikipedia.org/wiki/Metodo_della_bisezione.
[14] Wikipedia (Metodo delle tangenti), http://it.wikipedia.org/wiki/Metodo_delle_tangenti.
[15] Wikipedia (Halley's method), http://en.wikipedia.org/wiki/Halley's_method.
[16] Wikipedia (Newton's method), http://en.wikipedia.org/wiki/Newton's_method.
[17] Wikipedia (Joseph Raphson), http://en.wikipedia.org/wiki/Joseph_Raphson.
[18] E. Zeidler, *Nonlinear Functional Analysis and its Applications: Part 1: Fixed-Point Theorems*, Springer, (1998).