Equazioni nonlineari

28 novembre 2007

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

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 velocitá con cui ció 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

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

cioè

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

Facili conti mostrano che per $p = 1$ 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-');
```

ottenendo il grafico in figura.

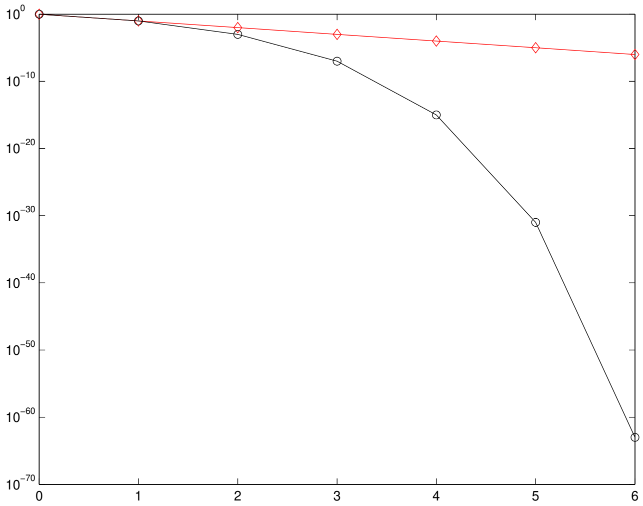

Figure 1: Grafico che illustra l'errore di un metodo con convergenza $p = 1$ (in rosso a rombi) e $p = 2$ (in nero a cerchietti), per $C = 1/10$ ed $e_0 = 1$.

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 semplicitá che sia uno solo, come nel caso in cui f sia crescente o decrescente.

Il metodo di bisezione puó 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

$$|b_k - a_k| \leq \epsilon_1$$

o quando
$$|f((a_k + b_k)/2)| \leq \epsilon_2 \tag{1}$$

o quando $k > nmax$, ove $nmax$ è un numero massimo di iterazioni. Successivamente descriveremo ed implementeremo una variante di (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` la 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

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

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(z1,z2,tolintv,tolres,maxit,f)

fz1=feval(f,z1);
fz2=feval(f,z2);

if fz1 == 0
    c=z1; k=0; semilunghezza=abs(z2-z1)/2; residuopesato=0;
    return;
end

if fz2 == 0
    c=z1; k=0; semilunghezza=abs(z2-z1)/2; residuopesato=0;
    return;
end

for index=1:maxit

    c(index)=(z1+z2)/2;
    fc(index)=feval(f,c(index));

    if sign(fc(index)) == sign(fz1)
        z1=c(index); fz1=fc(index);
    else
        z2=c(index); fz2=fc(index);
    end

    semilunghezza(index)=abs(z1-z2)/2;
    den=(fz2-fz1); den=(1-abs(sign(den)))*eps+den;
    residuopesato(index)=abs(fc(index)*2*(semilunghezza(index)/den));

    if (residuopesato(index) < tolres) | (semilunghezza(index) < tolintv) | (fc == 0)
```

```
        k=index;
        return;
    end

   fprintf('\n \t [IT]:%3.0f [c]: %5.5f [z1]:%3.3f',index,c(index),z1)
   fprintf('[AMP]: %2.2e [WRES]:%2.2e',semilunghezza(index),residuopesato(index));

end

k=maxit;
```

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 `semiamp` 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 `wres`, 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.

2. 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.

Esercizio. Salvare su un file **f.m** la function

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

e quindi digitare dalla shell 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 su shell 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 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

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

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.

Per quanto riguarda la velocità di convergenza si dimostra il seguente teorema di *convergenza locale* (cf. [1], p. 60)

Teorema 3.1 *Si assuma che $f(x)$, $f^{(1)}(x)$ ed $f^{(2)}(x)$ siano continue per ogni x in un certo intorno di x^* e che sia*

$$f(x^*) = 0,\ f^{(1)}(x^*) \neq 0.$$

Allora se x_0 è sufficientemente vicino a x^*, la successione $\{x_n\}$ converge a x^* ed è

$$\lim_{n\to\infty} \frac{x_{n+1} - x^*}{(x_{n+1} - x^*)^2} = -\frac{f^{(2)}(x^*)}{2f^{(1)}(x^*)} \tag{3}$$

provando che le iterate hanno ordine di convergenza 2.

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

Teorema 3.2 *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 $x_0 = b$.

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

Teorema 3.3 *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 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

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

3.1 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

$$x^2 - a = 0. \tag{4}$$

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

$$x_{k+1} = x_k + h_k,\ h_k = -\frac{x_k^2 - a}{2\,x_k} = -\frac{x_k}{2} + \frac{a}{2\,x_k},\ k = 0, 1, \ldots \tag{5}$$

Definiamo lo *scarto* (assoluto) di un metodo iterativo come

$$s_{n+1} = x_{n+1} - x_n. \tag{6}$$

Un test comunemente utilizzato è che sia

$$|s_{n+1}| \leq \tau \tag{7}$$

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 (6)-(7) 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.

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;
```

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

% PRIMA ITERAZIONE DEL METODO DI NEWTON.
k=1;
x(1)=x0;
fx=feval(funct,x(k));
dfx=feval(dfunct,x(k));
scarto(k)=-fx/dfx;
fprintf('\n \t [ITER.]: %3.0f',k);
fprintf('[VALORE]: %5.5f',x(k));
fprintf('[ABS.SCARTO]: %2.2e',abs(scarto(k)));
```

```
% ITERAZIONI SUCCESSIVE DEL METODO DI NEWTON.
while (abs(scarto(k)) > tau) & (k < kmax)
    k=k+1;
    x(k)=x(k-1)+scarto(k-1);
    fx=feval(funct,x(k));
    dfx=feval(dfunct,x(k));
    scarto(k)=-fx/dfx;
    fprintf('\n \t [ITER.]: %3.0f',k);
    fprintf('[VALORE]: %5.5f',x(k));
    fprintf('[ABS.SCARTO]: %2.2e',abs(scarto(k)));
end
```

A questo punto scriviamo sulla shell di Matlab/Octave

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

Il metodo stampa sulla stessa shell

```
 [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. $f'(x) = 2x \geq 0$, $f^{(2)}(x) = 2$ per $x \in [a, b]$.

Di conseguenza è possibile applicare il teorema 3.2, 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_*.

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

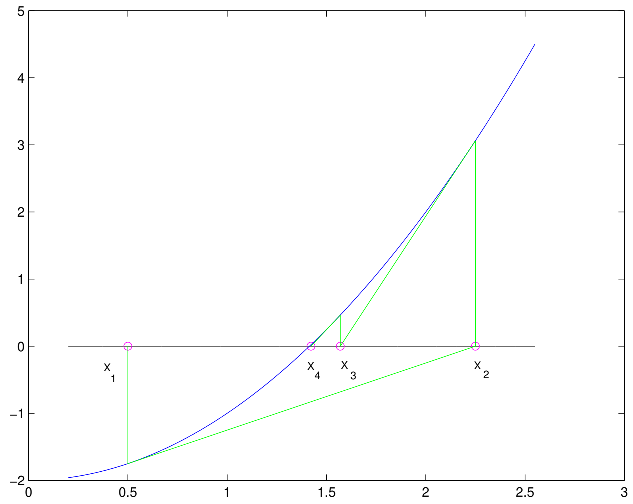

Figure 2: Grafico che illustra geometricamente le iterazioni del metodo Newton per il calcolo della radice quadrata di 2.

con particolari tolleranze richieste e numero massimo di iterazioni.

Plottiamo ora, in un unico grafico semi-logaritmico, l'andamento dell'errore relativo, 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 $k = 50$.

3.2.1 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

```
>> [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
```

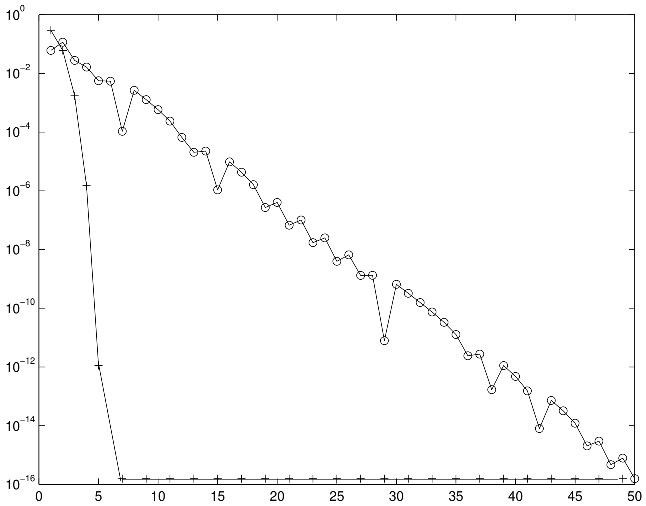

Figure 3: Grafico che illustra la convergenza del metodo di bisezione e Newton, rappresentate rispettivamente da o e +.

```
[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.

3.3 Alcuni tests numerici.

In questa sezione intendiamo eseguire alcuni test numerici con il metodo di bisezione e di Newton. A tal proposito salviamo in 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 le corrispondenti derivate nel file `dg.m`

```
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
```

Esercizio (Per casa). Si risolva numericamente il problema di calcolare la radice cubica di un numero, col metodo di Newton.

Esercizio (Facoltativo). 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?

Nota. 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) per calcolare la radice $2.09455148154233\ldots$ di $x^3 - 2x - 5$. 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.

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

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

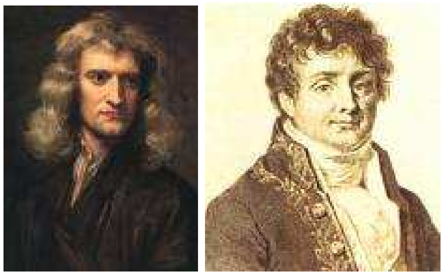

Figure 4: Ritratti di Newton (1642-1727) e Fourier (1768-1830).

Colson). Il metodo descritto era molto differente da quello attuale, e veniva sostanzialmente applicato a polinomi [2, p. 413], [11].

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.

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

3.3.1 Facoltativo: Un'equazione polinomiale.

Si supponga di dover calcolare la più grande radice dell'equazione

$$x^6 - x + 1 = 0 \tag{8}$$

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

$$p(x) := \sum_{j=0}^{n} a_j x^j \tag{9}$$

allora

$$|z_i| \leq 1 + \max_{0 \leq i \leq n-1} \frac{|a_i|}{|a_n|} \tag{10}$$

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 dalla shell 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
  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)
```

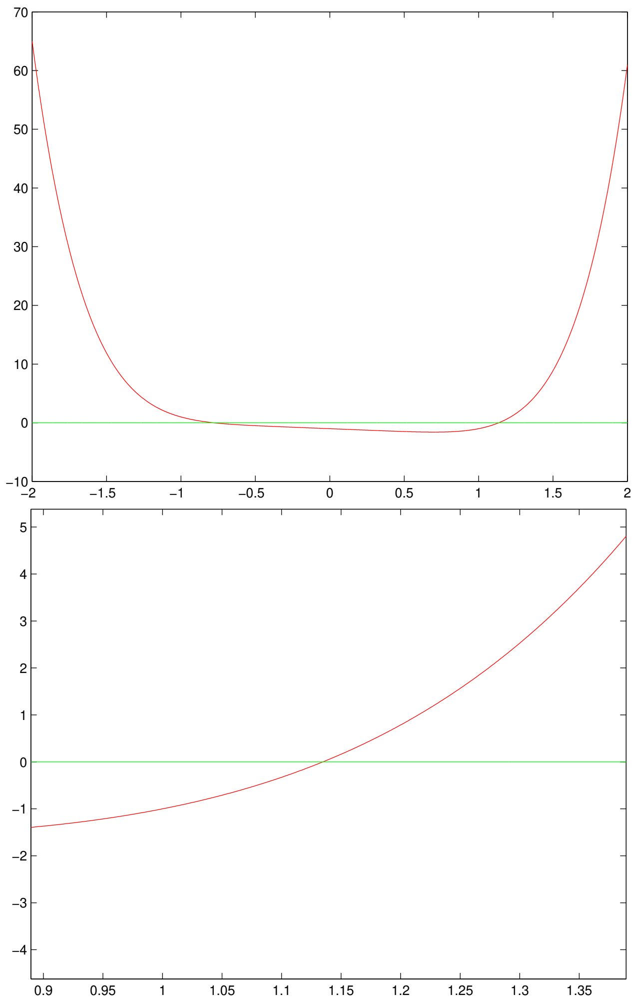

Figure 5: Grafico del polinomio $x^6 - x - 1$.

```
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 $Dg(x) := 6x^5 - 1$. 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]=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

>>
```

3.4 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^*. 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 nella shell 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]=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 lenta del metodo. Similmente, il metodo converge pure partendo da $x_0 = 8$, ma nuovamente i risultati non sono entusiasmanti:

```
>> [x,k,scarto]=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

>>
```

3.5 Esercizio facoltativo: Sugli zeri multipli di una funzione.

Si calcolino con il metodo di Newton gli zeri multipli della funzione

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

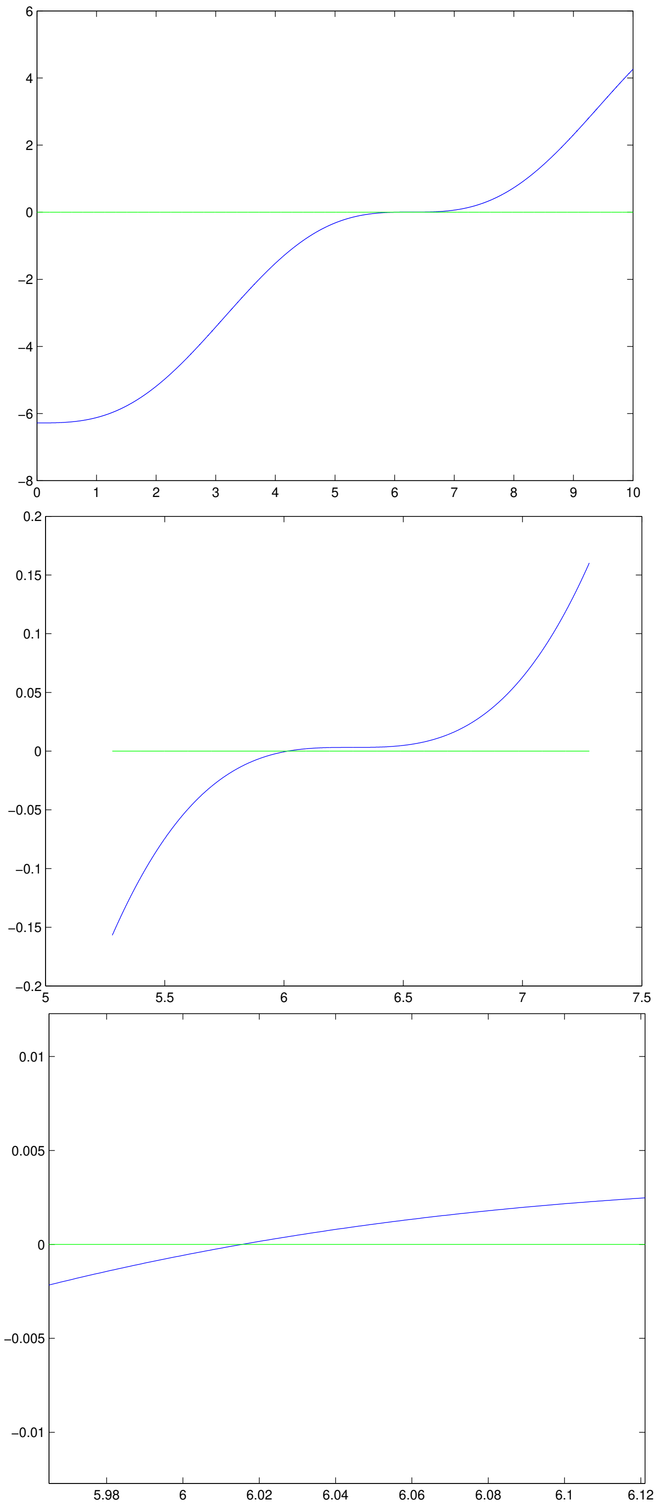

Figure 6: Grafico del polinomio $x - 6.28 - \sin(x)$.

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

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

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]=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]=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

$$e_{k+1} \approx Ce_k^p$$

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

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

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

Dalla shell 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.

3.6 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
```

3.7 Facoltativo: Alcuni esercizi

Esercizio (facoltativo). Nota la soluzione del problema $x^2 - a = 0$ (si ottiene con il comando `sqrt(a)`) si stimi la velocita' di convergenza del metodo di Newton (5) 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.

3.8 Online

Per approfondimenti, si considerino le pagine web

http://it.wikipedia.org/wiki/Calcolo_dello_zero_di_una_funzione
http://it.wikipedia.org/wiki/Metodo_della_bisezione
http://it.wikipedia.org/wiki/Metodo_delle_tangenti
http://en.wikipedia.org/wiki/Newton's_method
http://en.wikipedia.org/wiki/Joseph_Raphson
http://www-history.mcs.st-andrews.ac.uk/Biographies/Fourier.html
http://www-history.mcs.st-andrews.ac.uk/Biographies/Newton.html
http://www-history.mcs.st-andrews.ac.uk/Biographies/Raphson.html

References

[1] K. Atkinson, *An Introduction to Numerical Analysis*, Wiley, (1989).

[2] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, McGraw-Hill, (1990).

[3] A. Quarteroni, F. Saleri *Introduzione al Calcolo Scientifico*, Springer, (2002).

[4] Mac Tutor (Fourier), http://www-history.mcs.st-andrews.ac.uk/Biographies/Fourier.html.

[5] Mac Tutor (Newton), http://www-history.mcs.st-andrews.ac.uk/Biographies/Newton.html.

[6] Mac Tutor (Raphson), http://www-history.mcs.st-andrews.ac.uk/Biographies/Raphson.html.

[7] Wikipedia (Newton), http://en.wikiquote.org/wiki/Newton.

[8] Wikipedia (Calcolo dello zero di una funzione), http://it.wikipedia.org/wiki/Calcolo_dello_zero_di_una_funzione.

[9] Wikipedia (Metodo della bisezione),
http://it.wikipedia.org/wiki/Metodo_della_bisezione.

[10] Wikipedia (Metodo delle tangenti),
http://it.wikipedia.org/wiki/Metodo_delle_tangenti.

[11] Wikipedia (Newton's method),
http://en.wikipedia.org/wiki/Newton's_method.

[12] Wikipedia (Joseph Raphson),
http://en.wikipedia.org/wiki/Joseph_Raphson.