# Alcuni esperimenti in Matlab relativi alla teoria degli errori

Alvise Sommariva

Università degli Studi di Padova
Dipartimento di Matematica Pura e Applicata

12 aprile 2022

**Problema.**

*Dato $x^2 + 2px - q$, con $\sqrt{p^2+q} \geq 0$ implementare alcuni algoritmi in Matlab che valutino la radice positiva mediante la formula*

$$y = -p + \sqrt{p^2+q}. \tag{1}$$

Osserviamo che

- Algoritmo 1: si valuta

$$y = -p + \sqrt{p^2+q} \tag{2}$$

  (potenzialmente instabile causa cancellazione qualora $p \gg q$).

- Algoritmo 2: ottenuto via razionalizzazione di (1):

$$y = \frac{q}{(p + \sqrt{p^2+q})} \tag{3}$$

  (non ha problemi di cancellazione qualora $p \gg q$).

Algoritmo 1. Salviamo il seguente codice in `radicesecgrado.m`.

```
p=1000; q=0.018000000081; sol=0.9*10^(-5);

% ALGORITMO 1: s1=-p+sqrt(p^2+q)
s=p^2;
t=s+q;
if t >=0
   u=sqrt(t); % u=sqrt(p^2+q)
else
   fprintf('\n \t [RADICI COMPLESSE]');
end
s1=-p+u; % valutazione radice richiesta col primo algoritmo
```

Algoritmo 2. Di seguito, sullo stesso file scriviamo il secondo algoritmo ottenuto mediante razionalizzazione:

```
% ALGORITMO 2:  t1=q/ (p+sqrt(p^2+q))
s=p^2;
t=s+q;
if t >=0
   u=sqrt(t);  % u=sqrt(p^2+q)
else
   fprintf('\n \t [RADICI COMPLESSE]');
end
v=p+u;   % v=p+sqrt(p^2+q)
t1=q/v; % valutazione radice richiesta col secondo algoritmo
```

Infine, stampiamo risultati ed errori relativi.

```
% Soluzione fornita dal primo algoritmo.
fprintf('\n \t [ALG.1]: %10.19f',s1);

% Soluzione fornita dal secondo algoritmo.
fprintf('\n \t [ALG.2]: %10.19f',t1);

if length(sol) > 0 & (sol ~= 0)
    % Errore relativo del primo algoritmo.
    rerr1 =abs(s1-sol)/abs(sol);
    % Errore relativo del secondo algoritmo.
    rerr2=abs(t1-sol)/abs(sol);
    % Stampa risultati.
    fprintf('\n \t [REL.ERR.ALG.1]: %2.2e',rerr1);
    fprintf('\n \t [REL.ERR.ALG.2]: %2.2e',rerr2);
end

fprintf('\n \n');
```

Come previsto, il secondo algoritmo si comporta notevolmente meglio del primo, che compie un errore relativo dell'ordine di circa $10^{-9}$.

Infatti:

```
>> radicesecgrado
   [ALG.1]: 0.0000001000007614493
   [ALG.2]: 0.0000001000000000000
   [REL.ERR.ALG.1]: 7.61e-06
   [REL.ERR.ALG.2]: 1.32e-16
>>
```

**Nota.**

*Seppure l'errore relativo sembri piccolo, è significativo e non è dovuto al problema ma esclusivamente all'algoritmo utilizzato.*

Eseguiamo un codice Matlab che valuti le successioni $\{u_n\}$, $\{z_n\}$, teoricamente convergono a $\pi$, definite rispettivamente come

$$\begin{cases} s_1 = 1, \ s_2 = 1 + \frac{1}{4} \\ u_1 = 1, \ u_2 = 1 + \frac{1}{4} \\ s_{n+1} = s_n + \frac{1}{(n+1)^2} \\ u_{n+1} = \sqrt{6\, s_{n+1}} \end{cases}$$

e

$$\begin{cases} z_1 = 1, \ z_2 = 2 \\ z_{n+1} = 2^{n-\frac{1}{2}} \sqrt{1 - \sqrt{1 - 4^{1-n} \cdot z_n^2}} \end{cases} \qquad (4)$$

Implementiamo poi una terza successione, diciamo $\{y_n\}$, che si ottiene razionalizzando (4), cioè moltiplicando numeratore e denominatore di

$$z_{n+1} = 2^{n-\frac{1}{2}} \sqrt{1 - \sqrt{1 - 4^{1-n} \cdot z_n^2}}$$

per

$$\sqrt{1 + \sqrt{1 - 4^{1-n} \cdot z_n^2}}$$

e calcoliamo $u_m$, $z_m$ e $y_m$ per $m = 2, 3, \ldots, 40$ (che teoricamente dovrebbero approssimare $\pi$).

Infine disegniamo in un unico grafico l'andamento dell'errore relativo di $u_n$, $z_n$ e $y_n$ rispetto a $\pi$ aiutandoci con l'help di Matlab relativo al comando `semilogy`.

Di seguito scriviamo un'implementazione di quanto richiesto.

Scriviamo dapprima sul file `pigreco.m` il codice relativo alla valutazione dei primi 41 termini della prima successione $\{u_n\}_{n=1,\dots,41}$ definita da

$$\begin{cases} s_1 = 1, \ u_1 = 1 \\ s_{n+1} = s_n + \frac{1}{(n+1)^2} \\ u_{n+1} = \sqrt{6\, s_{n+1}} \end{cases}$$

```
% SEQUENZE CONVERGENTI "PI GRECO".

% METODO 1.
s(1)=1; u(1)=1;
for n=1:40
    s(n+1)=s(n)+(n+1)^(-2);
    u(n+1)=sqrt(6*s(n+1));
end
rel_err_u=abs(u-pi)/pi;

fprintf('\n');
```

Sempre sullo stesso file, scriviamo il codice relativo alla valutazione dei primi 41 termini della seconda successione $\{z_n\}_{n=1,\dots,41}$ definita da

$$\begin{cases} z_1 = 1, \ z_2 = 2 \\ z_{n+1} = 2^{n-\frac{1}{2}} \sqrt{1 - \sqrt{1 - 4^{1-n} \cdot z_n^2}} \end{cases} \tag{5}$$

```
% METODO 2.
format long
z(1)=1;
z(2)=2;
for n=2:40
    c=(4^(1-n)) * (z(n))^2; inner_sqrt=sqrt(1-c);
    z(n+1)=(2^(n-0.5))*sqrt( 1-inner_sqrt );
end
rel_err_z=abs(z-pi)/pi;

fprintf('\n');
```

Di seguito, scriviamo il codice relativo alla valutazione dei primi 41 termini della terza successione $\{y_n\}_{n=1,\ldots,41}$ ottenuta *razionalizzando la seconda*

```
% METODO 3.
y(1)=1;
y(2)=2;
for n=2:40
    num=(2^(1/2)) * abs(y(n));
    c=(4^(1-n)) * (z(n))^2;
    inner_sqrt=sqrt(1-c);
    den=sqrt( 1+inner_sqrt );
    y(n+1)=num/den;
end
rel_err_y=abs(y-pi)/pi;
```

Infine facciamo il grafico delle tre successioni all'interno della stessa finestra:

```
% SEMILOGY PLOT.
semilogy(1:length(u),rel_err_u,'k.',...
 1:length(z),rel_err_z,'m+',...
 1:length(y),rel_err_y,'ro');
```

Si osservi che i `tre` puntini alla fine delle terzultima e penultima riga permettono di andare a capo senza che il codice generi errore di sintassi.

Per concludere si digiti nella command window `pigreco`. Nella figura descriviamo i risultati.

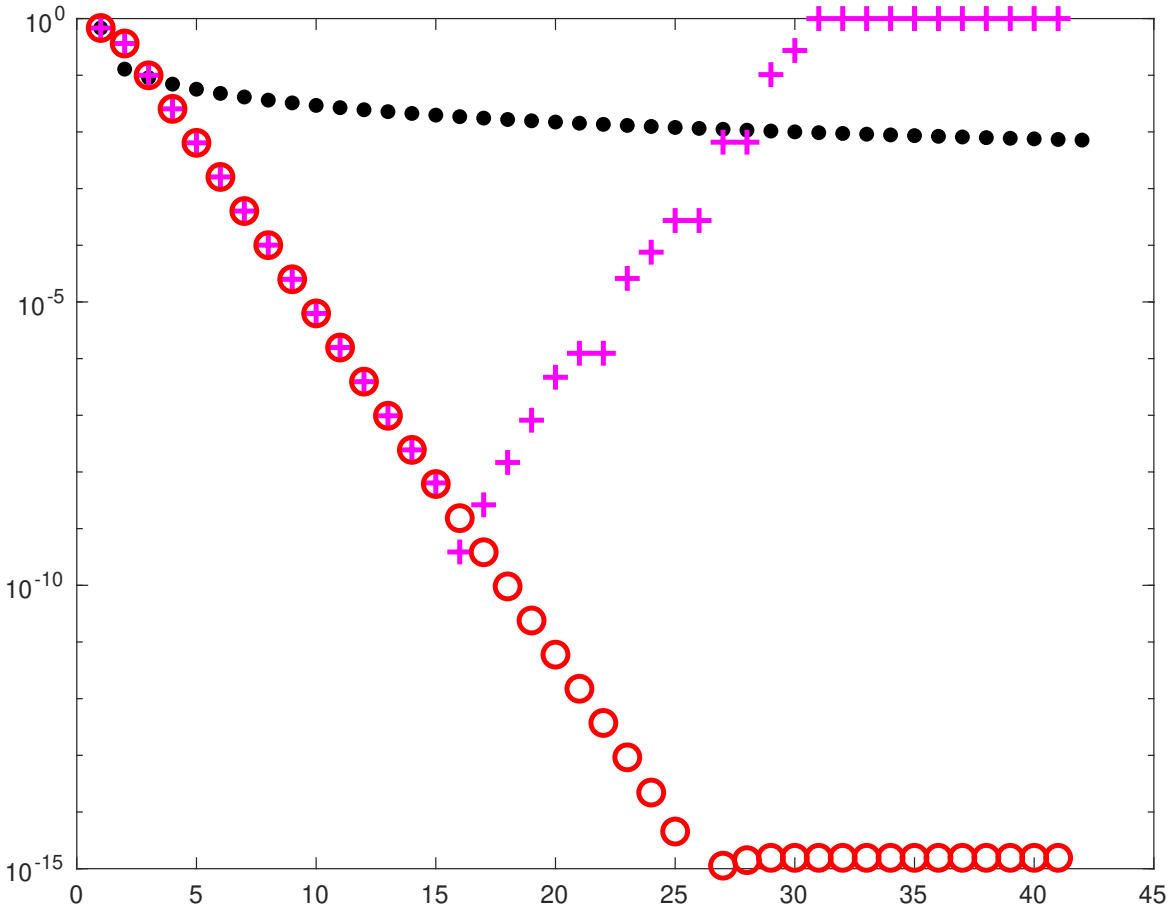


Figura: Grafico che illustra le 3 successioni, rappresentate rispettivamente la prima da `.`, la seconda da + e la terza da o.

## Discussione risultati.

- La prima successione converge molto lentamente a $\pi$, la seconda diverge mentre la terza converge velocemente a $\pi$.
- Per alcuni valori $\{z_n\}$ e $\{y_n\}$ coincidono per alcune iterazioni per poi rispettivamente divergere e convergere a $\pi$. Tutto ciò è naturale poichè le due sequenze sono analiticamente (ma non numericamente) equivalenti.
- Dal grafico dell'errore relativo, la terza successione, dopo aver raggiunto errori relativi prossimi alla precisione di macchina, si assesta ad un errore relativo di circa $10^{-15}$ (dovuti alla precisione di macchina).

## Una successione ricorrente.

Consideriamo la successione $\{I_n\}$ definita da

$$I_n = e^{-1} \int_0^1 x^n e^x \, dx \tag{6}$$

Si vede che

- $I_0 = 1 - e^{-1}$;
- $I_{n+1} = 1 - (n+1) I_n$ per $n = 0, 1, \ldots$;
- $I_n > 0$, decrescente e si prova che $I_n \to 0$ come $1/n$.
- se é noto $I_n$ allora $I_{n-1} = (1 - I_n)/n$.

Calcoliamo $I_n$ per $n = 1, \ldots, 99$:

- mediante la successione in avanti

$$\begin{cases} s_1 = e^{-1} = I_1 \\ s_{n+1} = 1 - (n+1) s_n \approx I_{n+1} \end{cases} \tag{7}$$

- mediante la successione all'indietro

$$\begin{cases} t_{1000} = 0 \\ t_{n-1} = (1 - t_n)/n \approx I_{n-1}. \end{cases} \tag{8}$$

con $n = 1000, 999, \ldots, 2$.

Scriviamo il codice in un file `succricorrente.m`.

```
% cancelliamo variabili e funzioni precedentemente definite.
clear all;
% successione "s_n".
s(1)=exp(-1); % valore che approssima $I_1$
for n=1:99
    % valore che approssima $I_{n+1}$
    s(n+1)=1-(n+1)*s(n);
end
% successione "t_n".
M=1000;
t=zeros(1,M); % inizializzazione "t" come vettore riga.
for n=M:-1:2
    % valore che approssima $I_{n-1}$
    t(n-1)=(1-t(n))/n;
end
% plot semilogaritmico
clf;
semilogy(1:length(s),abs(s),'k-',...
    1:length(s),abs(t(1:length(s))),'m-');
legend('s','t')
```

Quindi digitiamo sulla shell di Matlab/Octave `succricorrente`. Otteniamo il grafico in figura, che mostra come la prima successione non converge a 0 per una cattiva propagazione degli errori, mentre la seconda, quella all'indietro fornisce buoni risultati.

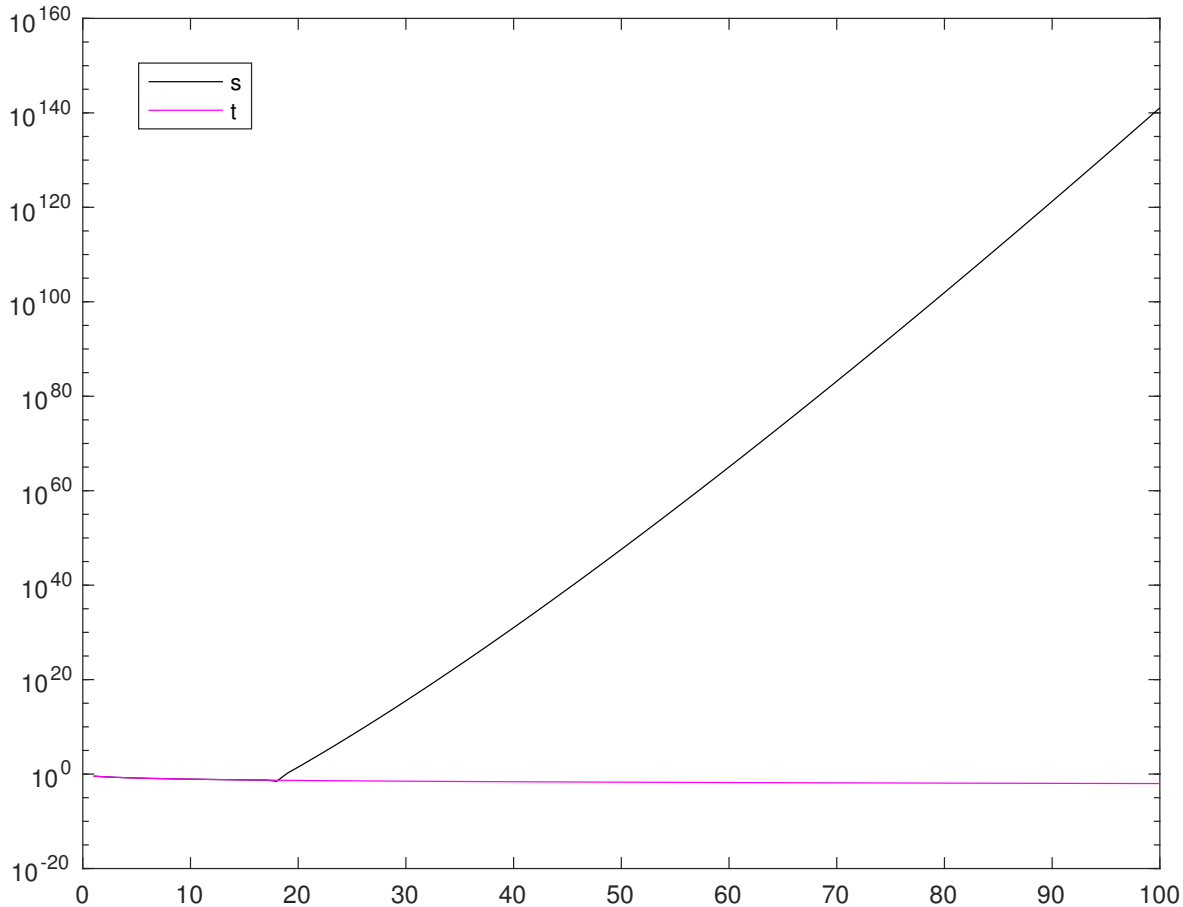


Figura: Grafico che illustra i valori assoluti assunti dalla successione in avanti (in nero) e all'indietro (in rosa magenta).

Osserviamo per prima cosa che $\exp(-1)$ non è un numero macchina e quindi verrà approssimato, compiendo un certo errore assoluto
$\epsilon \leq \exp(-1) \cdot eps \approx 8.2e-17$ (ricordare la definizione di precisione di macchina).
La successione in avanti amplifica gli errori. Infatti, nel caso migliore, se

$$\bar{I}_1 = I_1 + \epsilon$$

$$\bar{I}_{n+1} = 1 - (n+1)\bar{I}_n$$

allora

$$\begin{aligned}
\bar{I}_2 &= 1 - 2\bar{I}_1 = 1 - 2(I_1 + \epsilon) = I_2 - 2 \cdot 1 \cdot \epsilon \\
\bar{I}_3 &= 1 - 3\bar{I}_2 = 1 - 3(I_2 + 2\epsilon) = I_3 - 3 \cdot 2 \cdot 1 \cdot \epsilon \\
\bar{I}_4 &= 1 - 4\bar{I}_3 = 1 - 4(I_3 + 3\epsilon) = I_4 - 4 \cdot 3 \cdot 2 \cdot 1 \cdot \epsilon
\end{aligned}$$

e in generale

$$\bar{I}_n = I_n + (-1)^n n! \cdot \epsilon$$

ovvero

$$|\bar{I}_n - I_n| = n! \cdot \epsilon$$

con il termine $n! \cdot \epsilon$ che tende velocemente a $+\infty$ al crescere di $n$.

La successione all'indietro invece smorza gli errori. Infatti, se

$$\bar{I}_m = I_m + \epsilon, \quad \bar{I}_{n-1} = (1 - \bar{I}_n)/n$$

allora si vede con qualche conto che

$$\begin{aligned} \bar{I}_{m-1} &= I_{m-1} - \epsilon/m \\ \bar{I}_{m-2} &= I_{m-2} - \epsilon/((m-1)\cdot m) \\ &\dots \\ \bar{I}_{m-k} &= I_{m-k} - \epsilon/\prod_{s=0}^{k}(m-s) \end{aligned}$$

ovvero si compie un errore assoluto

$$|\bar{I}_{m-k} - I_{m-k}| = \epsilon/\prod_{s=0}^{k}(m-s)$$

con il termine $\epsilon/\prod_{s=0}^{k}(m-s)$ che tende velocemente a 0 al crescere di $k$.

Si tenga conto che relativamente all'esperimento

$$I_{1000} \approx 9.980049850517696e-04$$

e quindi nell'approssimarlo con $t_{1000} = 0$ si fa un'errore assoluto di circa $9e-04$.

Di conseguenza

$$\begin{aligned}
|\bar{I}_{999} - I_{999}| &\approx 9e-04/1000 = 9e-07,\\
|\bar{I}_{998} - I_{998}| &\approx \epsilon/(999\cdot 1000) \approx 9e-10,\\
|\bar{I}_{997} - I_{997}| &\approx \epsilon/(998\cdot 999\cdot 1000) \approx 9e-13,\\
|\bar{I}_{996} - I_{996}| &\approx \epsilon/(997\cdot 998\cdot 999\cdot 1000) \approx 9e-16,
\end{aligned}$$

e quindi già $I_{996}$ è calcolato con estrema accuratezza.

Ci poniamo il problema di valutare il polinomio

$$p(x) = a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n \tag{9}$$

in un punto $x$.

Osserviamo che

$$p(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + \ldots + x \cdot (a_{n-1} + x \cdot a_n))) \tag{10}$$

Supponiamo sia $a = (a_0, \ldots, a_n)$ il vettore di dimensione $n+1$ delle componenti del polinomio.

Possiamo valutare il polinomio tramite i seguenti due algoritmi,

- il primo che valuta direttamente il polinomio secondo quanto descritto in (9),
- il secondo che effettua la stessa operazione come descritto in (10) calcolando dapprima $s_1 = a_{n-1} + x \cdot a_n$, poi $s_2 = a_{n-2} + x \cdot s_1$ e così via.

Di seguito salviamo nella function `algoritmo_horner` il codice

```
clear all;
% il polinomio 1+2x+3x^2+4x^3 e' codificato con [1 2 3 4].
a=[1 2 3 4];
x=pi;
y1=algoritmo1(a,x);
y2=algoritmo2(a,x);
fprintf('\n \t algoritmo 1: %1.15e',y1);
fprintf('\n \t algoritmo 2: %1.15e',y2);
fprintf('\n \n');
```

In calce a `algoritmo_horner` scriviamo le function `algoritmo1`

```
function s=algoritmo1(a,x)
xk=1; s=a(1);
for i=2:length(a)
   xk=xk*x;
   s=s+a(i)*xk;
end
```

e la function `algoritmo2`

```
function s=algoritmo2(a,x)
L=length(a);
s=a(L); % COMPONENTE a_n IMMAGAZZINATA IN a(n+1).
for i=L-1:-1:1
      s=a(i)+x*s;
end
```

**Nota.**

- *Matlab permette di scrivere all'interno di una function, nel nostro caso* `algoritmo_horner` *altre functions utilizzato dallo stesso, ovvero* `algoritmo1`, `algoritmo2`.
- *Si osservi che questo non vale per uno script Matlab che non sia una function.*

Quindi lanciamo il codice `algoritmo_horner` per la valutazione di $p(x) = 1 + 2 \cdot x + 3 \cdot x^2 + 4 \cdot x^3$ in $x = \pi$ e ricaviamo

```
>> algoritmo_horner
   algoritmo 1: 1.609171052316469e+02
   algoritmo 2: 1.609171052316469e+02
>>
```

La differenza sta nella complessità computazionale e non nel risultato numerico.

- Il primo algoritmo richiede $2n$ moltiplicazioni e $n$ somme.
- Il secondo algoritmo richiede $n$ moltiplicazioni e $n$ somme.