

Sulla propagazione degli errori

10 novembre 2007

1 Esercitazione sulla propagazione degli errori

1.1 Rappresentazione dei numeri al calcolatore

Risolvere un problema mediante calcolo numerico ed al calcolatore (il calcolo numerico era nato come calcolo *a mano*, ma oggi questo *modus operandi* è praticamente caduto in disuso, dato il bassissimo costo del calcolo automatico), significa utilizzare un'aritmetica a precisione finita, e quindi produrre soluzioni approssimate e non esatte. Soluzioni esatte possono essere prodotte se si riesce a risolvere il problema mediante *calcolo simbolico*.

Per quanto concerne la rappresentazione numerica, i calcolatori moderni rispettano in genere lo standard IEEE 754 che andiamo brevemente a descrivere.

Nello standard IEEE 754, i numeri in virgola mobile a precisione singola occupano una *parola* da 32 bit, mentre quelli a precisione doppia occupano due parole consecutive da 32 bit. Un numero *non nullo-normalizzato* si scrive come

$$x = (-1)^S * 2^{E-Bias} * (1.F) \quad (1)$$

dove $S = 0$ oppure $S = 1$ determina il *segno*, $E - Bias$ è l'*esponente* cui va sottratto il valore del BIAS ed F la *mantissa*. Alcuni esempi

Precisione	Ebits	Bias	FBits
Singola	8	127	23
Doppia	11	1023	52

Per capire meglio tale notazione consideriamo il link

<http://babbage.cs.qc.edu/IEEE-754/Decimal.html>

Il numero 12 corrisponde in singola precisione a

[0|10000010|100000000000000000000000]

Vediamo di comprenderne il significato:

1. 0 determina il segno. Poichè $(-1)^0 = 1$ il segno è +.

2. 10000010 determina il valore. Essendo il BIAS uguale a 127 e (si legga da destra a sinistra!) $E = 10000010 = 0 * 2^0 + 1 * 2^1 + 0 * 2^2 + 0 * 2^3 + 0 * 2^4 + 0 * 2^5 + 0 * 2^6 + 1 * 2^7 = 2 + 128 = 130$, l'esponente vale $130 - 127 = 3$. Quindi $2^{E-Bias} = 2^3 = 8$.
3. $F = 10000000000000000000000000000000$ è la componente relativa alla mantissa. La quantità, letta da sinistra a destra, riguarda la parte decimale F in (1). Quindi $F = 1 * 2^{-1} + 0 * 2^{-2} + \dots = 0.5$. Ora $(1.F)$ si legge come $1 + F = 1.5$.

In altre parole la rappresentazione di 12 è esatta ed il numero è rappresentato come $12 = 8 * 1.5$.

Non convinti, proviamo un altro esempio, ad esempio 126 che corrisponde in singola precisione a

[0|10000101|111110000000000000000000]

1. 0 determina il segno. Poichè $(-1)^0 = 1$ il segno è +.
2. 10000101 determina il valore. Essendo il BIAS uguale a 127 e (si legga da destra a sinistra!) $E = 10000101 = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 0 * 2^3 + 0 * 2^4 + 0 * 2^5 + 0 * 2^6 + 1 * 2^7 = 1 + 4 + 128 = 133$, l'esponente vale $133 - 127 = 6$. Quindi $2^{E-Bias} = 2^6 = 64$.
3. $F = 1111100000000000000000000000000$ è la componente relativa alla mantissa. La quantità, letta da sinistra a destra, riguarda la parte decimale F in (1). Quindi $F = 1 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4} + 1 * 2^{-5} + \dots = 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 = 0.96875$. Ora $(1.F)$ si legge come $1 + F = 1.96875$.

In altre parole la rappresentazione di 126 è esatta ed il numero è rappresentato come $126 = 64 * 1.96875$.

Dalla tabella riguardante la precisione singola e i bit, si deduce che i numeri in virgola mobile non sono dunque un insieme denso, come sono invece i numeri reali, bensì un insieme discreto di valori sull'asse reale, in posizioni non equispaziate. Infatti, la distanza tra un numero in virgola mobile ed il successivo, è: $2^{-n_{bits}F} \cdot 2^{E-bias}$. La sua cardinalità è piuttosto elevata: in doppia precisione i numeri macchina sono

$$2^{64} \approx 1.844674407370955e + 019.$$

Si dice precisione di macchina, quel numero che rappresenta la distanza tra 1 ed il successivo numero in virgola mobile. In Matlab tale valore è rappresentato da `eps`. Vediamo di calcolarlo. Si scriva sulla shell di Linux

`pico myeps.m`

e nel file il codice Matlab

```
myeps=1;
while 1+myeps > 1;
    myeps=myeps/2;
end
myeps=2*myeps;
format long e
myeps-eps
```

Il codice Matlab parte con $\text{myeps}=1$ e continua a dividere per 2 finché trova un numero x tale che sommato a 1 dà (numericamente, non analiticamente) 1. Vista la definizione di myeps , $\text{eps} = 2 * x$. E infatti $\text{myeps}-\text{eps} = 0$.

I numeri nella rappresentazione floating-point stanno in un intervallo limitato, quindi l'*infinitamente piccolo* e l'*infinitamente grande* non sono rappresentabili come numeri in virgola mobile, nel calcolatore. Quando si raggiunge un valore talmente piccolo da non essere più distinto dallo zero, si parla di *underflow*, mentre quando si eccede il massimo numero rappresentabile, si parla di *overflow*. Il limite inferiore è pari a $2^{1-\text{bias}}$ ed in Matlab è espresso dalla variabile `realmin`. Il limite superiore è pari a $2^{2^{\text{Ebits}}-2-\text{bias}} * (1 + (1 - 2^{-\text{Fbits}}))$ ed in Matlab è espresso dalla variabile `realmax`. Verifichiamolo.

1. Dalla tabella, in precisione doppia, si ha che $\text{bias}=1023$ e quindi

$$2^{1-1023} = 2^{1022} = 2.225073858507201e - 308$$

Se digitiamo nella shell di Matlab il comando `realmin` otteniamo proprio questo valore.

2. Dalla tabella, in precisione doppia, si ha che $\text{bias}= 1023$, $\text{Ebits}= 11$, ed $\text{Fbits}= 52$. Conseguentemente

$$\begin{aligned} \text{realmax} &= 2^{2^{\text{Ebits}}-2-\text{bias}} * (1 + (1 - 2^{-\text{Fbits}})) \\ &= 2^{2^{11}-2-1023} * (1 + (1 - 2^{-52})) \\ &= 2^{2^{11}-2-1023} * (2 - 2^{-52}) = \\ &= 1.797693134862316e + 308 \end{aligned}$$

Per convincercene, digitiamo nella shell di Matlab/Octave

```
>> s=2^11-1025;
>> m=2-2^(-52)
m =
    2.0000
>> m=2-2^(-52);
>> t=(2^s)*m
t =
    1.7977e+308
>> realmax
ans =
    1.7977e+308
>> t-realmax
ans =
     0
>>
```

Quindi digitando `realmax` nella shell di Matlab otteniamo proprio questo valore.

Lo standard richiede che il risultato delle operazioni di addizione, sottrazione, moltiplicazione e divisione sia arrotondato esattamente, cioè il risultato deve essere calcolato esattamente e poi arrotondato al numero in virgola mobile più vicino.

Problemi in tal senso provengono dalla sottrazione. Per questo motivo, nei microprocessori moderni la sottrazione viene effettuata utilizzando la tecnica bit di guardia. Non tratteremo questo punto dettagliatamente. Per curiosità si confronti il sito

<http://babbage.cs.qc.edu/IEEE-754/References.xhtml>

1.2 Propagazione degli errori di arrotondamento

Per quanto detto, ogni calcolatore può rappresentare esattamente solo un numero finito di numeri reali, i cosiddetti numeri macchina. Per i rimanenti, seguendo uno standard quale IEEE 754, può fornirne solo un'approssimazione. Di conseguenza, le operazioni aritmetiche di base *possono* in generale essere soggette ad un errore nel risultato ed è dunque necessario conoscere l'entità di quest'ultimo e l'effetto che può avere in un algoritmo.

Questo problema non è da trascurarsi. Si possono trovare in internet vari siti in cui queste approssimazioni, dovute ad esempio al cambiamento di unità di misura, hanno portato a disastri quali la perdita di satelliti o la distruzione di stazioni petrolifere come indicato ad esempio in [3].

Passiamo quindi in dettaglio all'analisi del problema. Sia $x \neq 0$ un numero reale del tipo

$$x = (-1)^S \cdot 2^{(E-\text{bias})} \cdot (1.F + \delta)$$

in cui supponiamo $|\delta| < 2^{(-\text{nbitsF})}/2$. Allora, la sua rappresentazione in virgola mobile, sarà

$$\text{fl}(x) = (-1)^S \cdot 2^{(E-\text{bias})} \cdot 1.F$$

con un errore relativo (di arrotondamento)

$$\frac{|x - \text{fl}(x)|}{|x|} = \frac{|2^{(E-\text{bias})} \cdot \delta|}{|2^{(E-\text{bias})} \cdot (1.F + \delta)|} = \frac{|\delta|}{|1.F + \delta|}$$

Ricordiamo che fissato un vettore da approssimare

$$x^* = (x_1^*, \dots, x_n^*) \in \mathbb{R}^n,$$

si definisce

1. errore *assoluto* tra x e x^* la quantità

$$\|x - x^*\|$$

dove

$$\|y\| = \sqrt{\sum_{i=1}^n y_i^2}, \quad y = (y_1, \dots, y_n);$$

2. errore *relativo* tra x e $x^* \neq 0$ la quantità

$$\frac{\|x - x^*\|}{\|x^*\|}.$$

Per quanto riguarda le operazioni aritmetiche fondamentali si può dimostrare che la somma, la divisione e la moltiplicazione producono un errore relativo piccolo, mentre la sottrazione può anche produrre un errore relativo grande rispetto al risultato (ciò avviene quando i due operandi sono molto vicini tra di loro e si ha dunque una perdita di cifre significative nel risultato). Vediamo di seguito alcuni esempi in cui è fondamentale un adeguato trattamento dell'errore per giungere ad una buona approssimazione del risultato esatto.

1.3 Calcolo di una radice in una equazione di secondo grado

Vediamo un esempio concreto in cui l'introdurre una sottrazione potenzialmente pericolosa conduce effettivamente a problemi di instabilità della soluzione e come rimediare. Dato il polinomio di secondo grado $x^2 + 2px - q$, con $\sqrt{p^2 + q} \geq 0$ calcoliamo la radice

$$y = -p + \sqrt{p^2 + q}. \quad (2)$$

Questa classica formula è potenzialmente instabile a causa della sottrazione tra p e $\sqrt{p^2 + q}$. A tal proposito, dopo averla implementata in Matlab, verifichiamo la perdita di accuratezza per opportune scelte dei coefficienti p e q . Ripetiamo poi lo stesso tipo di indagine con una formula alternativa (e stabile) che si ottiene razionalizzando la formula (2). In altri termini

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

Seguendo [2] p. 10, il problema (e non l'algoritmo!) è *bencondizionato* per $q > 0$ e *malcondizionato* per $q \approx -p^2$.

Usando dei classici ragionamenti dell'analisi numerica si mostra che (cf. [6], p. 21, [2], p. 11)

1. il primo algoritmo non è *numericamente stabile* qualora $p \gg q$;
2. il secondo algoritmo è *numericamente stabile* qualora $p \gg q$.

Seguendo [6], p. 22, si ha un test interessante per

$$p = 1000, q = 0.018000000081$$

la cui soluzione esatta è $0.9 \cdot 10^{-5}$. Secondo [2], p. 11 è notevole l'esperimento in cui

$$p = 4.999999999995 \cdot 10^{+4}, q = 10^{-2}$$

avente soluzione esatta è 10^{-7} . Si osservi che in entrambi i casi effettivamente $p \gg q$.

1.3.1 Implementazione in Matlab

Scriviamo un programma `radicesecgrado.m` in Matlab che illustri i due algoritmi.

```
% p=4.999999999995*10^(+4); q=10^(-2); sol=10^(-7);
p=1000; q=0.018000000081; sol=0.9*10^(-5);

% ALGORITMO 1
s=p^2;
t=s+q;
if t >=0
    u=sqrt(t);
else
    fprintf('\n \t [RADICI COMPLESSE]');
end
s1=-p+u;

% ALGORITMO 2
s=p^2;
t=s+q;
if t >=0
    u=sqrt(t);
else
    fprintf('\n \t [RADICI COMPLESSE]');
end
v=p+u;
t1=q/v;

fprintf('\n \t [ALG.1] [1]: %10.19f',s1);
fprintf('\n \t [ALG.2] [1]: %10.19f',t1);
if length(sol) > 0 & (sol ~= 0)
    relerrs1=abs(s1-sol)/abs(sol);
    relerrt1=abs(t1-sol)/abs(sol);
    fprintf('\n \t [REL.ERR.] [ALG.1]: %2.2e',relerrs1);
    fprintf('\n \t [REL.ERR.] [ALG.2]: %2.2e',relerrt1);
end
```

Digitiamo quindi da shell Matlab/Octave il comando `radicesecgrado` e otteniamo

```
>> radicesecgrado

[ALG.1] [1]: 0.0000089999999772772
[ALG.2] [1]: 0.0000090000000000000
[REL.ERR.] [ALG.1]: 2.52e-009
[REL.ERR.] [ALG.2]: 0.00e+000
```

Come previsto, il secondo algoritmo si comporta notevolmente meglio del primo.

Esercizio. Testare il codice per il secondo esempio in cui

```
p=4.999999999995*10^(+4); q=10^(-2); sol=10^(-7);
```

1.4 Successioni convergenti a π

Si implementino le successioni $\{u_n\}$, $\{z_n\}$, 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

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

che *teoricamente* convergono a π . Si implementi poi la successione, diciamo $\{y_n\}$, che si ottiene *razionalizzando*, cioè moltiplicando numeratore e denominatore per

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

e si calcolino u_m , z_m e y_m per $m = 2, 3, \dots, 40$.

Si disegni in un unico grafico l'andamento dell'errore relativo di u_n , z_n e y_n rispetto a π . A tal proposito ci si aiuti con l'help di Matlab relativo al comando `semilogy`. I grafici devono avere colori o patterns diversi. Facoltativo: in un riquadro mettere un legame tra colore (e/o pattern) e successione, usando il comando Matlab `legend` (aiutarsi con l'help). Ricordiamo che tale comando non esiste in GNU Octave.

1.4.1 Risoluzione

In seguito scriviamo un implementazione di quanto richiesto commentando i risultati. Si salvi in un file `pi_greco.m` il codice

```
% SEQUENZE CONVERGENTI "PI GRECO".

% METODO 1.

s(1)=1; u(1)=1;
s(2)=1.25; u(2)=s(2);
for n=2:40
    s(n+1)=s(n)+(n+1)^(-2);
    u(n+1)=sqrt(6*s(n+1));
    fprintf('\n \t [SEQ.1][INDEX]: %3.0f', n);
    fprintf(' [REL.ERR]: %2.2e', abs(u(n+1)-pi)/pi);
end
rel_err_u=abs(u-pi)/pi;
```

```

fprintf('\n');

% 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 );
    fprintf('\n \t [SEQ.2] [N]: %3.0f', n);
    fprintf(' [REL.ERR]: %2.2e', abs(z(n+1)-pi)/pi);
end
rel_err_z=abs(z-pi)/pi;

fprintf('\n');

% 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;
    fprintf('\n \t [SEQ.3] [N]: %3.0f',n);
    fprintf(' [REL.ERR]: %2.2e', abs(z(n+1)-pi)/pi);
end
rel_err_y=abs(y-pi)/pi;

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

```

1.4.2 Commenti alla Risoluzione

1. Non tutti i programmi sono functions. Alcuni sono semplicemente un codice che viene interpretato da Matlab (il cosiddetto *programma principale*). Usiamo funzioni solo quando vogliamo introdurre porzioni di codice che possono tornare utili a più programmi principali, o che semplificano la loro lettura.
2. Più assegnazioni possono essere scritte in una stessa riga. Per convincersene si osservi la prima riga del file `pigreco.m` dopo i commenti.

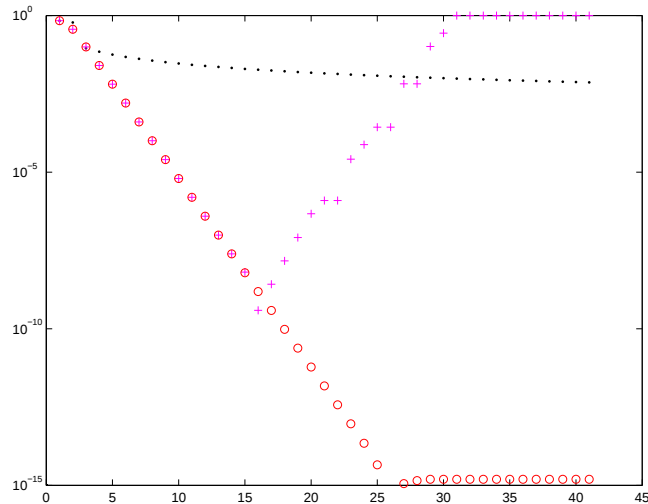


Figure 1: Grafico che illustra le 3 successioni, rappresentate rispettivamente da ., + e o.

3. L'istruzione descritta dopo `for n=2:40` non richiede l'incremento della variabile n , in quanto ciò è automatico.
4. Non serve il ; dopo l'end che chiude il ciclo `for`.
5. Con un po' di tecnica il ciclo `for` è sostituibile col un ciclo `while`. In questo caso però bisogna incrementare la variabile n .
6. Nel denominatore riguardante l'errore relativo scriviamo π e non `abs(pi)` in quanto $\pi = |\pi|$.
7. Nel comando `fprintf`, utilizziamo `\n` che manda a capo e `\t` che esegue un *tab* (uno spazietto in avanti). Se non si scrive `\n`, Matlab scriverà di seguito l'output sulla stessa riga, rendendo difficile la lettura dell'errore relativo.
8. Nel comando `fprintf` alcune parti vengono scritte come testo altre come contenuto di una variabile. Consideriamo ad esempio gli `fprintf` nella prima successione:

```
fprintf('\n \t [SEQ.1] [INDEX]: %3.0f',n+1);
```

```
fprintf('\n \t [REL.ERR.]: %2.2e \n', relerru(n+1) );
```

Dopo essere andati a capo e spaziato a destra, Matlab scrive sul monitor

```
[SEQ.1] [INDEX] :
```

e quindi valuta $n+1$ che viene stampato con *tre cifre decimali prima della virgola* in notazione decimale. Scrive poi sul monitor [REL. ERR.] : , e accede alla *cella di memoria* della variabile *relerru* di cui considera la componente $n+1$ -sima. Di seguito stampa su monitor il suo valore con *due cifre decimali prima della virgola, due cifre decimali dopo la virgola* in notazione esponenziale.

9. Il comando `semilogy` ha come primo argomento l'ascissa (che in questo caso sono gli indici di iterazione 1 : 41) e quale secondo argomento l'ordinata `relerru`. Nel grafico (in scala logaritmica nelle ordinate y), vengono disegnate l' i -sima componente dell'ascissa e l' i -sima componente delle ordinate per $i = 1, \dots, \dim(\text{relerru})$. Il grafico viene *tenuto* grazie al comando di `hold on` e di seguito si ripete il procedimento per `relerrz` e `relerry`. Si osservi che i vettori *ascissa* e *ordinata* devono essere della stessa dimensione e dello stesso tipo (cioè entrambi vettori riga o entrambi vettori colonna).
10. Dall'help di `semilogy` si evince che se la variabile *ascissa* non viene scritta allora Matlab indicizza automaticamente col vettore di naturali da 1 alla dimensione del vettore *ordinata*.

Per il risultato del plot si consideri la prima figura. Abbiamo indicato la prima successione con `.`, la seconda con `+` e la terza successione con `o`.

Dal punto di vista dell'analisi numerica si vede che

1. La prima successione converge molto lentamente a π , la seconda diverge mentre la terza converge velocemente a π .
2. Per alcuni valori $\{z_n\}$ e $\{y_n\}$ coincidono per alcune iterazioni per poi rispettivamente divergere e convergere a π . Tutto ciò è naturale poichè le due sequenze sono analiticamente (ma non numericamente) equivalenti.
3. 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} (probabilmente per questioni di arrotondamento).

1.5 Successione ricorrente

Sia

$$I_n = e^{-1} \int_0^1 x^n e^x dx \quad (4)$$

E' facile verificare che, essendo

$$\int x \exp(x) dx = (x-1) \exp(x) + c$$

si ha dal secondo teorema del calcolo integrale (cf. [8]) $I_1 = e^{-1}$ e più in generale integrando per parti che

$$I_{n+1} = e^{-1} \left(x^{n+1} e^x \Big|_0^1 - (n+1) \int_0^1 x^n e^x dx \right) = 1 - (n+1) I_n. \quad (5)$$

Da (4) essendo l'integranda $x^n \exp x > 0$ per $x \in (0, 1]$ si ha

$$I_n > 0$$

mentre da (5), calcolando il limite per $n \rightarrow \infty$ ad ambo i membri si ottiene

$$\lim_n I_n = 0.$$

1. Si calcoli I_n per $n = 1, \dots, 99$ mediante la successione *in avanti*

$$\begin{cases} s_1 = e^{-1} \\ s_{n+1} = 1 - (n+1) s_n \end{cases}$$

2. Fissato $m = 500$, si calcoli la successione *all'indietro* $\{t_n\}_{n=1, \dots, 100}$ definita come

$$\begin{cases} t_{2m} = 0 \\ t_{n-1} = \frac{1-t_n}{n} \end{cases}$$

Si osservi che per raggiungere tale obiettivo bisogna calcolare i termini

$$t_m, t_{m-1}, \dots, t_{100}, t_{99}, \dots, t_2, t_1.$$

3. Si disegni in un unico grafico semi-logaritmico (usare `semilogy` e `subplot`) l'andamento di $|s_n|$ e $|t_n|$ per $n = 1, \dots, 100$.
4. Si calcoli il valore di t_1 per diversi valori di m , da $m = 1$ a $m = 10$ e lo si confronti con I_1 .
5. Si calcolino infine a mano $e_m^{(s)} := I_m - s_m$ in funzione di $e_1^{(s)}$ e $e_1^{(t)} := I_1 - t_1$ in funzione di $e_m^{(t)}$. Infine si spieghi l'andamento oscillante della successione $\{s_n\}$.

1.5.1 Risoluzione

Di seguito vediamo un'implementazione di quanto richiesto. L'ultimo punto del problema lo lasciamo al lettore. Scriviamo il codice in un file `succricorrente.m`:

```
% SUCCESSIONE RICORRENTE.

% SUCCESSIONE "s_n".
s(1)=exp(-1);
for n=1:99
    s(n+1)=1-(n+1)*s(n);
end

% SUCCESSIONE "t_n".
m=500; M=2*m;
t=zeros(M,1); % INIZIALIZZAZIONE "t".
```

```

for n=M:-1:2
    j=n-1;
    t(j)=(1-t(n))/n;
end

% PLOT SEMI-LOGARITMICO.
semilogy(1:length(s),abs(s),'k-'); hold on;
semilogy(1:length(s),abs(t(1:length(s))), 'm-');
pause(5); hold off;

% ANALISI DI t(1) PER VALORI DIFFERENTI DI "m".
t_1_exact=exp(-1);
for m=1:10
    M=2*m;
    t=zeros(M,1); % INIZIALIZZAZIONE "t".

    for n=M:-1:2 % SI OSSERVI CHE IL CICLO VA DA "M" A 2.
        t(n-1)=(1-t(n))/n;
    end

    val_t_1(m)=t_1_exact-t(1);
    fprintf('\n \t [M]: %2.0f [VAL.]: %10.15f',M,val_t_1(m));

end

semilogy(1:length(val_t_1),abs(val_t_1),'k-');

```

1.5.2 Commenti alla Risoluzione

Alcune osservazioni sul codice

- abbiamo usato un comando del tipo `for n=M:-1:2` cosicchè il ciclo `for` parte da M e arriva a 2, sottraendo di volta in volta 1;
- abbiamo inizializzato con un comando del tipo `zeros(M,1)` il vettore colonna t , che altrimenti non è in memoria (e quindi le componenti più piccole di $t(n-1)$ sarebbero inaccessibili poichè indefinite);
- abbiamo applicato un grafico semilogaritmico poichè quello usuale non dice granchè (sperimentarlo).
- in figura abbiamo plottato le successioni $\{|s_i|\}$, $\{|t_i|\}$ e non $\{s_i\}$, $\{t_i\}$. Questo non è un problema in quanto

$$s_i \rightarrow 0 \Leftrightarrow |s_i| \rightarrow 0$$

$$t_i \rightarrow 0 \Leftrightarrow |t_i| \rightarrow 0.$$

D'altro canto la prima successione diverge assumendo anche valori negativi, rendendo difficile la comprensione del grafico.

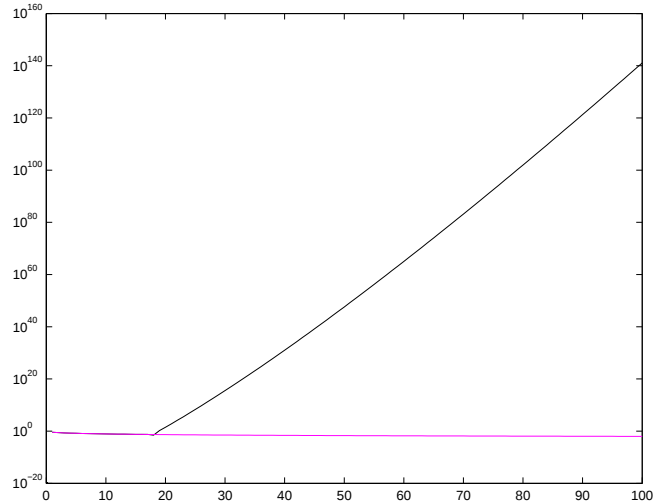


Figure 2: Grafico che illustra i valori assoluti assunti dalla successione in avanti (in nero) e all'indietro (in rosa magenta)

Numericamente, dopo aver digitato sulla shell di Matlab/Octave `succricorrente` otteniamo con un po' di attesa due grafici e i seguenti risultati. Il primo grafico (si veda la figura 2) mostra un confronto tra i risultati della successione in avanti (quella plottata in nero) e quelli della successione all'indietro (in rosa magenta). Dai ragionamenti qualitativi, è evidente che la prima non fornisce la soluzione, mentre la seconda sembra convergere a 0.

Il secondo grafico (si veda la figura 1.1) per vari $M = 2 * m$ mostra il comportamento del metodo all'indietro nell'approssimare il valore iniziale $\exp - 1$. Evidentemente, al crescere di m , l'approssimazione diventa sempre più precisa. In particolare l'errore assoluto compiuto è stampato nel display dal programma `ricorrente.m` come segue:

```
>> ricorrente
```

```
[M]: 2 [VAL.]: 0.132120558828558
[M]: 4 [VAL.]: 0.007120558828558
[M]: 6 [VAL.]: 0.000176114384113
[M]: 8 [VAL.]: 0.000002503273002
[M]: 10 [VAL.]: 0.00000023114272
[M]: 12 [VAL.]: 0.00000000149839
[M]: 14 [VAL.]: 0.000000000000720
[M]: 16 [VAL.]: 0.000000000000003
[M]: 18 [VAL.]: 0.000000000000000
[M]: 20 [VAL.]: 0.000000000000000
```

```
>>
```

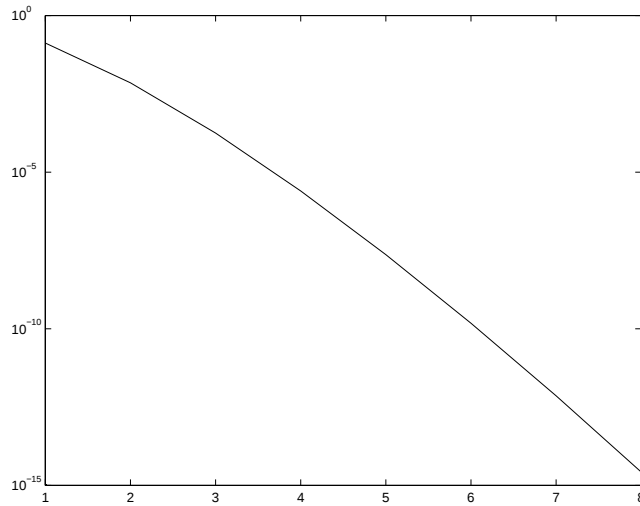


Figure 3: Grafico che illustra il valore assoluto della successione all'indietro nell'approssimare $\exp(-1)$.

2 Alcuni esercizi

2.1 Esercizio 1

Si calcoli l'errore relativo tra 1 e il valore che si ottiene valutando in Matlab/Octave

$$\frac{(1 + \eta) - 1}{\eta}$$

con $\eta = 10^{-1}, 10^{-2}, \dots, 10^{-15}$. Si consideri poi $\eta = 8.8817841970012523E - 16$ e si calcoli la medesima quantità, giustificando i risultati ottenuti.

Suggerimento: quanto vale `eps`?

2.2 Esercizio 2

Siano $f := \tan$ e $g := \arctan$. Si consideri la funzione composta

$$f \cdot g(x) := f(g(x)) = \tan(\arctan(x)) = x, \quad x \in (-\infty, +\infty).$$

Si calcolino

$$x = 10^k, \text{ per } k = -20 + 40h, \text{ e } h = 0, 0.01, 0.02, \dots, 1$$

e si valuti l'errore relativo compiuto. Si può dire che anche *numericamente* $\tan(\arctan(x)) = x$?

References

- [1] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.
- [2] V. Comincioli, *Problemi di analisi numerica*, Mc Graw-Hill, 1991.
- [3] T. Huckle , <http://www.zenger.informatik.tu-muenchen.de/persons/huckle/bugse.html>.
- [4] J.H. Mathews e K.D. Fink, *Numerical Methods using Matlab*, Prentice Hall, 1999.
- [5] A. Quarteroni e F. Saleri, *Introduzione al calcolo scientifico*, Springer Verlag, 2006.
- [6] J. Stoer, *Introduzione all'analisi numerica*, Zanichelli, 1984.
- [7] The MathWorks Inc., *Numerical Computing with Matlab*, <http://www.mathworks.com/moler>.
- [8] Wikipedia, http://it.wikipedia.org/wiki/Teorema_fondamentale_del_calcolo_integrale