SULLA PROPAGAZIONE DEGLI ERRORI *

A. SOMMARIVA † E M.VENTURIN ‡

0.1. Facoltativo: 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.

Si intende per *bit* una cifra nel sistema (usualmente binario) usato [2, p. 48], [9, p. 47].

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

$$x = (-1)^S \cdot 2^{E-Bias} \cdot (1.F) \tag{0.1}$$

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*. La scrittura $1.F$ significa $1 + F$ dove $F = \sum_{k=1}^{+\infty} a_k 2^{-k}$, con $a_k = 0$ oppure $a_k = 1$. 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

in cui viene convertito un numero decimale in forma binaria, secondo lo standard IEEE 754. Partiamo con un esempio semplice.
Il numero 12 corrisponde in singola precisione a

$$[0\,|\,10000010\,|\,10000000000000000000000]$$

Cerchiamo di comprenderne il significato:

1. 0 determina il segno. Poichè $(-1)^0 = 1$ il segno è $+$.
2. 10000010 determina il valore della potenza di 2. 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. $F = 10000000000000000000000$ è la componente relativa alla mantissa. La quantità , letta da sinistra a destra, riguarda la parte decimale F in (0.1). Quindi $F = 1 * 2^{-1} + 0 * 2^{-2} + \ldots = 0.5$. Ora $(1.F)$ si legge come $1 + F = 1.5$.

*Ultima revisione: 15 maggio 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)

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

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

$$[0\,|\,10000101\,|\,11111000000000000000000]$$

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 $2^{E-Bias} = 2^6 = 64$. Viene da chiedersi perchè introdurre il BIAS. La risposta è che rappresenta un alternativa all'introduzione di un bit di segno per l'esponente.
3. $F = 11111000000000000000000$ è la componente relativa alla mantissa. La quantità , letta da sinistra a destra, riguarda la parte decimale F in (0.1). Quindi $F = 1 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4} + 1 * 2^{-1} + \ldots = 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 = 0.96875$. 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 $126 = 64 \cdot 1.96875$.

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 :

$$2^{\text{-nbitsF}} \cdot 2^{\text{E-bias}}.$$

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 `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 `eps` alla fine bisogna porre myeps $= 2 *$ myeps (altrimenti myeps $<$ eps). E infatti myeps-eps $= 0$.

0.2. Facoltativo: Su `eps`, `realmax`, `realmin`, overflow e underflow. 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 pi distinto dallo zero, si parla di *underflow*, mentre quando si eccede il massimo numero rappresentabile, si parla di *overflow*. Il limite inferiore (che non sia *denormalizzato*!) è pari a $2^{1-\text{bias}}$ ed in Matlab è espresso dalla variabile `realmin`. Il limite superiore è pari a $2^{2^{\text{Ebits}}-2-bias} * (1 + (1 - 2^{-\text{Fbits}}))$ ed in Matlab è espresso dalla variabile `realmax`. Verifichiamolo.

1. Dalla tabella, in precisione doppia, si ha che *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 *bias*$= 1023$, *Ebits*$= 11$, ed *Fbits*$= 52$. Conseguentemente

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

   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.

Osserviamo che in realtà `realmin` non sembra essere il numero più piccolo in valore assoluto. Testiamolo sulla shell di Matlab.

```
>> help realmax

 REALMAX Largest positive floating point number.
    x = realmax is the largest floating point number representable
    on this computer.  Anything larger overflows.
```

```
    See also EPS, REALMIN.

 Overloaded methods
    help quantizer/realmax.m

>> a=realmax*2
a =
   Inf
>> help realmin

 REALMIN Smallest positive floating point number.
    x = realmin is the smallest positive normalized floating point
    number on this computer.  Anything smaller underflows or is
    an IEEE "denormal".

    See also EPS, REALMAX.

 Overloaded methods
    help quantizer/realmin.m

>> realmin/(2^51) == 0
ans =
     0
>> realmin/(2^52) == 0
ans =
     0
>> realmin/(2^53) == 0
ans =
     1
>>
```

Siccome in un costrutto logico il valore 1 corrisponde a `true`, si ricava che

$$\text{realmin}/(2^{52}) \approx 4.940656458412465\text{e}-324$$

è ancora un numero macchina, ma che fa parte dei cosidetti *numeri denormalizzati* (cf. [9], p.49) e cioè del tipo

$$x = (-1)^S \cdot 2^{E-Bias} \cdot (0.F). \tag{0.2}$$

Per curiosità vediamo il codice Matlab che implementa `realmin`:

```
function xmin = realmin
%REALMIN Smallest positive floating point number.
%   x = realmin is the smallest positive normalized floating point
%   number on this computer.  Anything smaller underflows or is
%   an IEEE "denormal".
%
%   See also EPS, REALMAX.

%   C. Moler, 7-26-91, 6-10-92.
%   Copyright 1984-2001 The MathWorks, Inc.
%   $Revision: 5.8 $  $Date: 2001/04/15 12:02:27 $
```

```
minexp = -1022;

% pow2(f,e) is f*2^e, computed by adding e to the exponent of f.

xmin = pow2(1,minexp);
```

Per quanto concerne pow2:

```
>> help pow2

 POW2   Base 2 power and scale floating point number.
    X = POW2(Y) for each element of Y is 2 raised to the power Y.

    X = POW2(F,E) for each element of the real array F and a integer
    array E computes X = F .* (2 .^ E).  The result is computed
    quickly by simply adding E to the floating point exponent of E.
    This corresponds to the ANSI C function ldexp() and the IEEE
    floating point standard function scalbn().

    See also LOG2, REALMAX, REALMIN.

>>
```

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 consulti il sito

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

Ogni calcolatore può rappresentare esattamente solo un numero finito di numeri reali, i cosidetti 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 [5].
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 $|\delta| < 2^{(-\text{nbitsF})}/2$. In precisione singola, secondo IEEE 754, Ebits $= 8$, bias $= 127$, Fbits $= 23$. La scrittura $1.F$ significa $1 + F$ dove $F = \sum_{k=1}^{+\infty} a_k 2^{-k}$, con $a_k = 0$ oppure $a_k = 1$.

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

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

si definisce

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

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

   dove

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

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

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

Si noti che se $\|x^*\| = 0$, cioè $x^* = 0$ per la proprietà delle norme [1, p.481], allora non ha senso la scrittura propria dell'errore relativo.
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).
Più precisamente se $\text{fl}(x)$ è il numero macchina che *corrisponde* a x, denotati con

$$x \oplus y = \text{fl}(\text{fl}(x) + \text{fl}(y)) \tag{0.3}$$
$$x \ominus y = \text{fl}(\text{fl}(x) - \text{fl}(y)) \tag{0.4}$$
$$x \otimes y = \text{fl}(\text{fl}(x) \cdot \text{fl}(y)) \tag{0.5}$$
$$x \oslash y = \text{fl}(\text{fl}(x) : \text{fl}(y)) \tag{0.6}$$

e per $\odot$ una delle operazioni precedentemente introdotte, cioè una tra $\oplus$, $\ominus$, $\otimes$, $\oslash$ corrispondenti a op cioè una tra $+$, $-$, $\cdot$, $:$, posto

$$\epsilon_x = \frac{|x - \text{fl}(x)|}{|x|} \tag{0.7}$$
$$\epsilon_{x,y}^{\odot} = \frac{|(x \,\text{op}\, y) - (x \odot y)|}{|x \,\text{op}\, y|} \tag{0.8}$$

si ha dopo qualche non banale conto (cf. [2, p.78])

$$\epsilon^{\oplus}_{x,y} \approx |\frac{x}{x+y}|\epsilon_x + |\frac{y}{x+y}|\epsilon_y \tag{0.9}$$

$$\epsilon^{\ominus}_{x,y} \approx |\frac{x}{x-y}|\epsilon_x + |\frac{y}{x-y}|\epsilon_y \tag{0.10}$$

$$\epsilon^{\otimes}_{x,y} \approx \epsilon_x + \epsilon_y \tag{0.11}$$

$$\epsilon^{\oslash}_{x,y} \approx |\epsilon_x - \epsilon_y| \tag{0.12}$$

il che mostra il pericolo di perdita di accuratezza nella somma e nella sottrazione qualora rispettivamente $x + y \approx 0$ e $x - y \approx 0$.

1. 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 $x^2 + 2\,px - q$, con $\sqrt{p^2+q} \geq 0$ calcoliamo la radice

$$y = -p + \sqrt{p^2+q}. \tag{1.1}$$

Osserviamo che essendo $\sqrt{p^2+q} \geq 0$ le radici

$$y = -p \pm \sqrt{p^2+q}. \tag{1.2}$$

dell'equazione sono reali. La soluzione descritta in (1.1) è la maggiore delle 2, ed è non negativa se e solo se $q \geq 0$. Si osserva subito che (1.1) è potenzialmente instabile per $p \gg q$ a causa della sottrazione tra p e $\sqrt{p^2+q}$. A tal proposito, dopo averla implementata in Matlab, verificheremo numericamente 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 (1.1). In altri termini

$$y = -p + \sqrt{p^2+q} = \frac{(-p+\sqrt{p^2+q})(p+\sqrt{p^2+q})}{(p+\sqrt{p^2+q})} = \frac{q}{(p+\sqrt{p^2+q})} \tag{1.3}$$

Ricordiamo ora che un problema si dice *bencondizionato* (o *malcondizionato*) a seconda che nel particolare contesto le perturbazioni sui dati non influenzino (o influenzino) eccessivamente i risultati. Nel caso di un algoritmo, per indicare un simile comportamento rispetto alla propagazione degli errori dovute alle perturbazioni sui dati, si parla di *algoritmo bencondizionato* (o *algoritmo malcondizionato*) anche se è più usuale il termine di stabilità [2, p. 66].

Seguendo [2, p. 10], [2, p. 78], 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. [10], p. 21, [3], p. 11)

1. il primo algoritmo (1.2) non è *numericamente stabile* qualora $p \gg q > 0$;
2. il secondo algoritmo (1.3) è *numericamente stabile* qualora $p \gg q > 0$.

In [10, p.22], si suggerisce un test interessante per

$$p = 1000,\ q = 0.018000000081$$

la cui soluzione esatta è $0.9 \cdot 10^{-5}$. Secondo [3], p. 11 è notevole l'esperimento in cui

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

avente soluzione esatta 10^{-7}. Si osservi che in entrambi i casi effettivamente $p \gg q$.
Passiamo ora all'implementazione e verifica numerica di questi due tests, osservando che i problemi relativi sono comunque ben condizionati.

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, che compie un errore relativo dell'ordine di circa 10^{-9}.

Esercizio veloce. Testare il codice `radicesecgrado.m` per l'esempio in cui

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

2. Approssimazione numerica di π. Storicamente sono state scoperte diverse successioni convergenti sempre più rapidamente a π (cf. [18]). In questa sezione ci interesseremo a 3 di queste mostrando sia come le velocità di convergenza possano essere diverse, sia come possano insorgere questioni di instabilità [4, p. 16].

2.1. Successioni convergenti a π: tests numerici. 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 $m = 2, 3, \ldots, 40$ (che teoricamente dovrebbero approssimare π). Non è difficile osservare che

$$\begin{cases} y_1 = 1,\ y_2 = 2 \\ y_{n+1} = \frac{2^{\frac{1}{2}} |y_n|}{\sqrt{1 + \sqrt{1 - 4^{1-n} \cdot y_n^2}}} \end{cases}$$

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 vecchie versioni di GNU Octave. Un esempio del suo utilizzo in Octave è (cf. [7]

```
legend ("sin (x)");
```

In seguito scriviamo un'implementazione di quanto richiesto commentando i risultati. Si salvi in un file `pigreco.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(y(n+1)-pi)/pi);
end
rel_err_y=abs(y-pi)/pi;


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

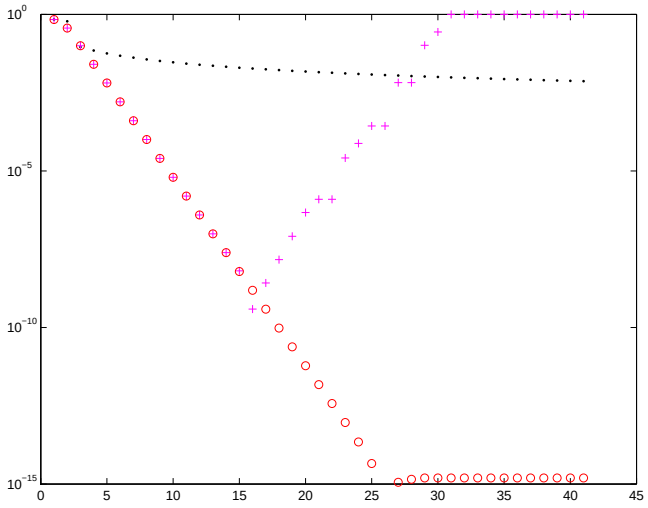

FIGURA 2.1. *Grafico che illustra le 3 successioni, rappresentate rispettivamente da* `.`, `+` *e* `o`.

1. Non tutti i programmi sono functions. Alcuni sono semplicemente un codice che viene interpretato da Matlab (il cosidetto *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. Piú assegnazioni possono essere scritte in una stessa riga. Per convincersene si osservi la prima riga del file `pigreco.m` dopo i commenti.
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 `pi` 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

   ```
   {\tt {[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 $i = 1, \ldots,$ dim(`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`. Osserviamo che in Octave, i grafici vengono rappresentati differentemente in quanto invece di un cerchietto viene talvolta disegnato un rombo.
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).

3. Successione ricorrente. In questa sezione mostriamo come alcune formule di ricorrenza in avanti possano essere instabili, mentre d'altro canto le relative versioni all'indietro possono essere stabili [8, Esercizio 1.9, p.35]. Problemi simili con una precisa discussione della propagazione dell'errore sono trattati pure in [4, p. 23, problema 11]

Sia

$$I_n = e^{-1} \int_0^1 x^n \, e^x \, dx \tag{3.1}$$

Per $n = 0$ si ha

$$I_0 = e^{-1} \int_0^1 e^x \, dx = e^{-1}(e^1 - 1).$$

Per $n = 1$, è facile verificare che, essendo

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

dal secondo teorema del calcolo integrale (cf. [14]) si ha che $I_1 = e^{-1} \approx 0.3679$ e più in generale integrando per parti, cioè osservando che

$$\int_a^b f'(x)\, g(x) dx = f(x)\, g(x) \mid_a^b - \int_a^b f(x)\, g'(x) dx,$$

ricaviamo per $f(x) = \exp(x)$ e $g(x) = x^{n+1}$ e dalla definizione di I_n

$$\begin{aligned} I_{n+1} &= \exp(-1) \int_0^1 x^{n+1} \exp(x)\, dx \\ &= \exp(-1) \left(x^{n+1} \exp(x) \mid_0^1 - (n+1) \int_0^1 x^n \exp(x)\, dx \right) \\ &= \exp(-1) \left(1^{n+1} \cdot \exp(+1) - 0 \cdot \exp(0) - (n+1) \exp(+1) \cdot \exp(-1) \int_0^1 x^n \exp(x)\, dx \right) \\ &= 1 - (n+1) \exp(-1) \int_0^1 x^n \exp(x)\, dx \\ &= 1 - (n+1)\, I_n. \end{aligned} \tag{3.2}$$

Da (3.1) essendo l'integranda $x^n \exp x > 0$ per $x \in (0, 1]$ si ha

$$I_n > 0.$$

Inoltre $x \leq 1$ implica $x^2 \leq x$ e più in generale $x^{n+1} \leq x^n$ da cui $x^{n+1} \exp(x) \leq x^n \exp(x)$ e quindi

$$I_{n+1} \leq I_n.$$

La successione I_n è positiva e non crescente e quindi ammette limite L finito. Da (3.2), calcolando il limite L per $n \to \infty$ ad ambo i membri si ottiene portando 1 a primo membro

$$L - 1 = -\lim_n (n+1)\, I_n.$$

L'unica possibilità affinchè $\lim_n (n+1)\, I_n$ esista e sia finito è che sia

$$L = \lim_n I_n = 0.$$

Infatti se il limite L fosse non nullo, si avrebbe

$$L - 1 = \lim_n (n+1)\, I_n = \lim_n (n+1) \lim_n I_n = +\infty,$$

il che è assurdo essendo $L < +\infty$ e quindi $L - 1 < +\infty$.

1. Si calcoli I_n per $n = 1, \ldots, 99$ 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* $\{t_n\}_{n=1,\ldots,100}$ 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

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

Nota: la successione all'indietro deriva dall'osservare che per n sufficientemente grande $I_n \approx 0$. Quindi, posto per n sufficientemente grande $I_n = 0$ riscrivendo la successione in avanti come successione all'indietro (cioè I_n in funzione di I_{n+1}), abbiamo

$$I_n = \frac{1 - I_{n+1}}{n+1}.$$

3. Si disegni in un unico grafico semi-logaritmico (usare `semilogy` e `subplot`) l'andamento di $|s_n|$ e $|t_n|$ per $n = 1, \ldots, 100$.
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. (Esercizio non banale). Si calcolino 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 $\{s_n\}$.
   Suggerimento: si scriva $e_1^{(s)} = \delta$ (o equivalentemente $s_1 = e^{-1} + \delta$) ed in seguito si esprima $e_m^{(s)} := I_m - s_m$ in funzione di δ. Da questi conti si vede il motivo del cattivo comportamento della successione in avanti: un piccolo errore sul dato iniziale, ha effetti catastrofici sul risultato al crescere di m.

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" (IN AVANTI).
s(1)=exp(-1);
for n=1:99
    s(n+1)=1-(n+1)*s(n);
end

% SUCCESSIONE "t_n" (ALL'INDIETRO).
m=500; M=2*m; % PER CALCOLARE I VALORI TRA
              % 500 E 1, PONGO t_1000=0.
t=zeros(M,1); % INIZIALIZZAZIONE "t".
for n=M:-1:2
    j=n-1;
    t(j)=(1-t(n))/n;
end

%-----------------------------------------
% ESEGUE UNA MATRICE 2 x 1 DI GRAFICI.
% IL PRIMO GRAFICO LO METTE IN (1,1).
%-----------------------------------------
```

```
subplot(2,1,1);

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


%----------------------------------------
% ERRORI ASSOLUTI.
% ANALISI DI t(1) POSTO t(m)=0; m=2,4,...,20
%----------------------------------------
t_1_exact=exp(-1);
for m=1:10

    M(m)=2*m;
    tt=zeros(M(m),1); % INIZIALIZZAZIONE "t".

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

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

end

%--------------------------------------
% SECONDO GRAFICO: ERRORI ASSOLUTI.
%--------------------------------------
subplot(2,1,2)
semilogy(M,abs(val_t_1),'k-');
```

Alcune osservazioni sul codice

- Nel primo grafico sono stati calcolati i valori assoluti di s_n perchè la successione in avanti assume valori negativi, non permettendo così l'uso di `semilogy` nel plot successivo;
- 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 poiché 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

$$s_i \to 0 \Leftrightarrow |s_i| \to 0$$

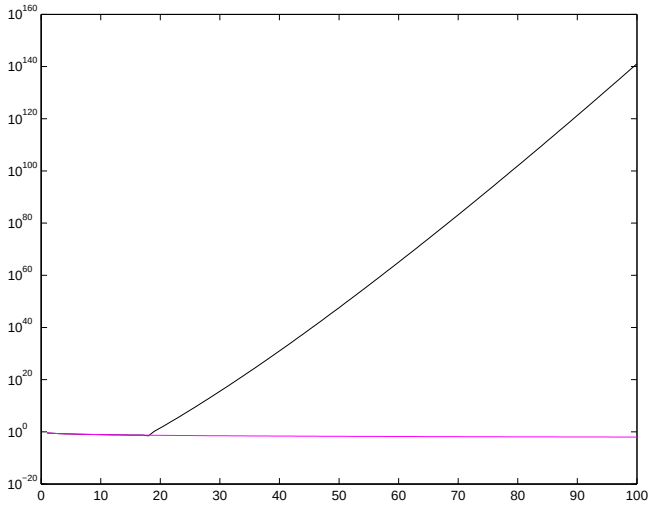

FIGURA 3.1. *Grafico che illustra i valori assoluti assunti dalla successione in avanti (in nero) e all'indietro (in rosa magenta). Sono stati calcolati i valori assoluti perchè la successione in avanti assume valori negativi, non permettendo così l'uso di* `semilogy`

$$t_i \to 0 \Leftrightarrow |t_i| \to 0.$$

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

- Il comando `subplot` permette di plottare più grafici nella stessa finestra di Matlab/Octave, senza sovrapporli. Per ulteriori informazioni, dall'help di Matlab:

```
SUBPLOT Create axes in tiled positions.
   H = SUBPLOT(m,n,p), or SUBPLOT(mnp), breaks the Figure window
   into an m-by-n matrix of small axes, selects the p-th axes for
   for the current plot, and returns the axis handle.  The axes
   are counted along the top row of the Figure window, then the
   second row, etc.  For example,

       SUBPLOT(2,1,1), PLOT(income)
       SUBPLOT(2,1,2), PLOT(outgo)

   plots income on the top half of the window and outgo on the
   bottom half.
```

3.0.5. Analisi dei risultati ottenuti. 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 4) 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 grafico 3.0.5 mostra per $m = 2, \ldots, 20$ il comportamento del metodo all'indietro nell'approssimare il valore iniziale $I_1 = \exp -1$. Evidentemente, al crescere di m, l'approssima-

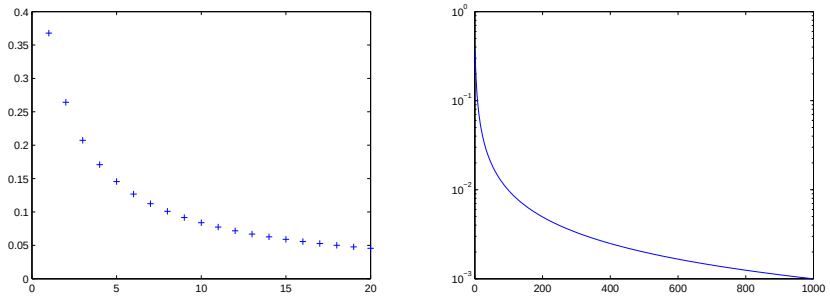

FIGURA 3.2. *Grafico che illustra la successione in avanti: i primi 20 e i primi 1000 termini. Si ricordi che* $\exp(-1) \approx 0.3679$.

zione diventa sempre più accurata. In particolare l'errore assoluto compiuto è stampato nel workspace dal programma `succricorrente.m` come segue:

```
>> succricorrente

          [M]:  2 [VAL.]: 0.132120558828558
          [M]:  4 [VAL.]: 0.007120558828558
          [M]:  6 [VAL.]: 0.000176114384113
          [M]:  8 [VAL.]: 0.000002503273002
          [M]: 10 [VAL.]: 0.000000023114272
          [M]: 12 [VAL.]: 0.000000000149839
          [M]: 14 [VAL.]: 0.000000000000720
          [M]: 16 [VAL.]: 0.000000000000003
          [M]: 18 [VAL.]: 0.000000000000000
          [M]: 20 [VAL.]: 0.000000000000000

>>
```

4. Facoltativo: Un esempio sulla soluzione delle equazioni di secondo grado. Consideriamo l'equazione di secondo grado

$$x^2 - 2\sqrt{3}x + 3 = 0 \tag{4.1}$$

Si nota subito che

$$x^2 - 2\sqrt{3}x + 3 = (x - \sqrt{3})^2 \tag{4.2}$$

e quindi (4.2) ha un'unica soluzione $\sqrt{3}$ con molteplicità 2, poichè la derivata di $x^2 - 2\sqrt{3}x + 3$ è la funzione $2x - 2\sqrt{3}x$ si annulla in $\sqrt{3}$ (cf. [2, p.420]).

Per quanto riguarda il condizionamento di una radice α (cf. [2, p.420]), si prova che se $\alpha(\epsilon)$ è la radice di

$$P_\epsilon(z) = P(z) + \epsilon g(z), \;\; \epsilon > 0 \tag{4.3}$$

allora se α è semplice

$$\alpha(\epsilon) \approx \alpha + \epsilon \left(\frac{-g(\alpha)}{P^1(\alpha)} \right) \tag{4.4}$$

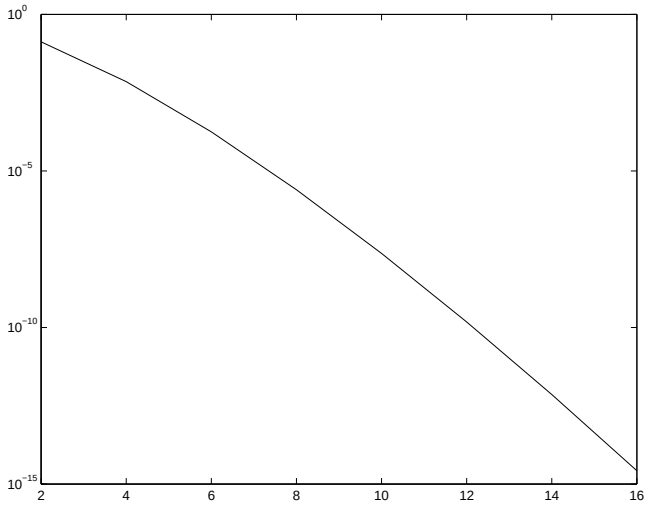

FIGURA 3.3. *Grafico che illustra l'errore assoluto compiuto dalla successione all'indietro nell'approssimare* $\exp(-1)$ *(partendo da* $t_2 = 0, \ldots, t_20 = 0$*).*

mentre se α è multipla con moltiplicità m, cioè

$$P(\alpha) = \ldots = P^{(m-1)}(\alpha) = 0 \;\; P^{(m)}(\alpha) \neq 0$$

allora esiste una radice $\alpha(\epsilon)$ di P_ϵ tale che

$$\alpha(\epsilon) \approx \alpha + \epsilon^{1/m} \left(\frac{-m! g(\alpha)}{P^{(m)}(\alpha)} \right) \tag{4.5}$$

Approssimiamo in (4.2) il valore $\sqrt{3}$ con 1.7321 fornito da Matlab usando il cosidetto `format short`. In questo caso, posto $g(z) = 2z$,

$$\epsilon = -(\sqrt{3} - 1.7321) \approx 4.9192 \cdot 10^{-5} \approx 5 \cdot 10^{-5}$$

abbiamo che

$$\alpha(\epsilon) \approx \alpha + \epsilon^{1/m} \left(\frac{-m! \alpha}{P^{(2)}(\alpha)} \right) \tag{4.6}$$

ed essendo $P^{(2)}(z) = 2$, $m = 2$ (molteplicità della radice $\alpha = \sqrt{3}$), ricaviamo

$$\alpha(\epsilon) \approx \alpha + (5 \cdot 10^{-5})^{1/2} \left(\frac{-2!2\sqrt{3}}{2} \right) \tag{4.7}$$

in cui

$$|(5 \cdot 10^{-5})^{1/2} \left(\frac{-2!2\sqrt{3}}{2} \right)^{1/2}| \approx 0.02449489742783$$

Quindi ci si aspetta che una delle radici di

$$P_\epsilon(z) = x^2 - 2 \cdot 1.7321 \cdot x + 3$$

disti circa 0.02449489742783 da $\sqrt{3}$.

In effetti, utilizzando semplici modifiche del programma `radicisecondogrado`, si hanno due radici,

$$x_1 \approx 1.7451541181241765000, \;\; x_2 \approx 1.7190458818758234000.$$

ed è , come si vede facilmente dalla shell di Matlab/Octave:

```
>> 1.7451541181241765000-sqrt(3)
ans =
    0.0131
>> 1.7190458818758234000-sqrt(3)
ans =
   -0.0130
>>
```

5. Esercizio. Una funzione f risulta difficile da valutare al calcolatore nel punto $x \neq 0$ in cui $f(x) \neq 0$ qualora a piccoli errori (relativi) sui dati

$$|x - x_c|/|x|$$

corrispondano grandi errori (relativi) sui risultati

$$|f(x) - f(x_c)|/|f(x)|.$$

Di conseguenza è importante valutare la quantità

$$\mathcal{K}(f, x, x_c) = \frac{|f(x) - f(x_c)|/|f(x)}{|x - x_c|/|x|}.$$

Se f è derivabile con continuità nel più piccolo intervallo $\mathcal{I}$ aperto contenente x ed x_c, per il teorema della media, essendo

$$f(x) - f(x_c) = f'(\xi) \cdot (x - x_c), \;\; \xi \in \mathcal{I},$$

ricaviamo facilmente che

$$\mathcal{K}(f, x, x_c) \approx \mathcal{K}(f, x) := \frac{|x \cdot f'(x)|}{|f(x)|}.$$

Date le funzioni

$$f_1(x) = 1 - \sqrt{1 - x^2}$$

$$f_2(x) = 1 - x$$

si calcoli analiticamente il *condizionamento*

$$\mathcal{K}(f_1, x) = \frac{|x \cdot f_1'(x)|}{|f_1(x)|}$$

$$\mathcal{K}(f_2, x) = \frac{|x \cdot f_2'(x)|}{|f_2(x)|}$$

Utilizzando Matlab/Octave, si plottino in scala semilogaritmica i grafici di $\mathcal{K}(f_1, \cdot)$ e $\mathcal{K}(f_2, \cdot)$ nel set di punti $x = -1 + 10^{(-3)}, -1 + 2 \cdot 10^{(-3)}, \ldots, 1 - 2 \cdot 10^{(-3)}, 1 - 10^{(-3)},$. Si può dire da questi dove la funzione è malcondizionata (cioè assume valori di condizionamento *alti*)?

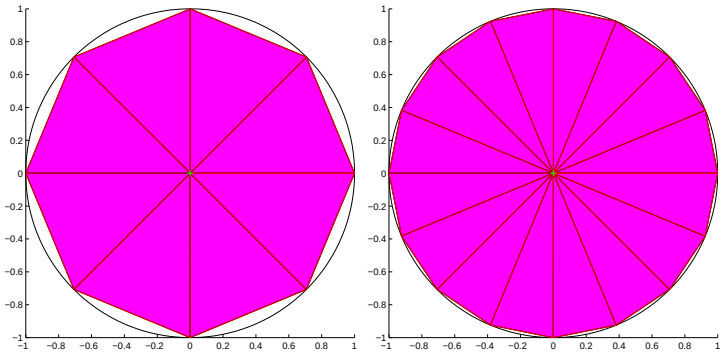

FIGURA 6.1. *Grafico che illustra le suddivisioni considerate dall'algoritmo di Archimede, per $p = 3$, $p = 4$.*

6. Alcuni esercizi.

1. **Esercizio facile**. Si calcoli l'errore relativo tra 1 e il valore che si ottiene valutando in Matlab/Octave
$$\frac{(1+\eta)-1}{\eta}$$
con $\eta = 10^{-1}, 10^{-2}, \ldots, 10^{-15}$. Si consideri poi $\eta = 8.8817841970012523E-16$ e si calcoli la medesima quantitá, giustificando i risultati ottenuti [4, p. 5, problema 3].
Suggerimento: quanto vale `eps`?

2. **Esercizio facile**. Siano $f := \tan$ e $g := \arctan$. Si consideri la funzione composta
$$f \cdot g(x) := f(g(x)) = \tan(\arctan(x)) = x, \ x \in (-\infty, +\infty).$$
Si calcolino
$$x = 10^k, \text{ per } k = -20 + 40h, \text{ e } h = 0, 0.01, 0.02, \ldots, 1$$
e si valuti l'errore relativo compiuto. Si puó dire che anche *numericamente* $\tan(\arctan(x)) = x$?

3. **Esercizio facoltativo di media difficoltà** . Esistono più successioni che approssimano π e sono attribuite ad Archimede. Ad esempio, osservato che questi non è altro che l'area del cerchio avente raggio 1, si inscrivono nel cerchio al variare di $p = 2, 3, \ldots$ dei poligoni regolari aventi 2^p lati. Osserviamo che per $p = 2$, si deve determinare l'area del quadrato inscritto, che con facili conti è uguale a 2.
Nel caso generale, si osserva che il poligono consiste di 2^p triangoli in cui un lato corrisponde ad un lato del poligono e un vertice con il centro del cerchio (si confronti con le figure).
Se indichiamo con θ l'angolo al centro di ogni triangolo, si verifica che la sua area è $\frac{\sin(\theta)}{2}$. In particolare per un poligono di 2^p lati, si ha $\theta_p = \frac{2\pi}{2^p}$ e l'area del poligono inscritto, costituito da 2^p triangoli uguali, è
$$I_p = 2^p \frac{\sin(\theta_p)}{2} = 2^p \frac{\sin(\frac{2\pi}{2^p})}{2}.$$
In realtà Archimede usò questa ed altre idee per cercare di approssimare π con stime dall'alto e dal basso, via anche poligoni circoscritti. Come citato in [13]:

FIGURA 6.2. *Un francobollo raffigurante Archimede (287ac-212ac).*

Nel breve lavoro La misura del cerchio *viene dimostrato anzitutto che un cerchio equivalente a un triangolo con base eguale alla circonferenza e altezza eguale al raggio. Tale risultato è ottenuto approssimando arbitrariamente il cerchio, dall'interno e dall'esterno, con poligoni regolari inscritti e circoscritti. Con lo stesso procedimento Archimede espone un metodo con il quale può approssimare arbitrariamente il rapporto tra circonferenza e diametro di un cerchio dato, rapporto che oggi si indica con* π*. Le stime esplicitamente ottenute limitano questo valore fra 22/7 e 3 + 10/71.* Secondo [19], Archimede usò un altro algoritmo per approssimare π. Visto che 2π non è altro che la lunghezza della circonferenza del cerchio avente raggio uguale a 1 e questa è maggiore del perimetro a_k di un qualsiasi poligono regolare di $n = 6 \cdot 2^k$ lati inscritto e minore del perimetro di un qualsiasi poligono regolare di $n = 6 \cdot 2^k$ lati circoscritto b_k, misurando a_k e b_k si può affermare che

$$a_k \leq 2\pi \leq b_k.$$

Si può inoltre provare che

$$a_k = 2n \, \sin\left(\frac{\pi}{n}\right),$$

$$b_k = 2n \, \tan\left(\frac{\pi}{n}\right).$$

Stimare π come

$$a_k \leq 2\,\pi \leq b_k.$$

Per quale n si riesce a determinare π con 10 cifre decimali esatte? Pensare se sia giusto dire $b_k - a_k < 10^{-10}$ allora π è approssimato da a_k con 10 cifre decimali esatte.

A. Si approssimi π usando il primo algoritmo di Archimede (basato sull'area di poligoni regolari inscritti) per $p = 2 : 20$. Si stampi l'errore assoluto e relativo compiuto.

B. Stimare π come

$$a_k \leq \pi \leq b_k$$

mediante l'algoritmo di Archimede basato sulla valutazione dei perimetri a_k, b_k di alcuni poligoni regolari. Per quale n si riesce a determinare π con 10 cifre decimali

esatte? Per $k = 0, \ldots, 4$ si ha

$$3.00000 \leq \pi \leq 3.46410 \tag{6.1}$$
$$3.10583 \leq \pi \leq 3.21539 \tag{6.2}$$
$$3.13263 \leq \pi \leq 3.15966 \tag{6.3}$$
$$3.13935 \leq \pi \leq 3.14609 \tag{6.4}$$
$$3.14103 \leq \pi \leq 3.14271 \tag{6.5}$$

Osservazione: il fatto che ci sia un π nei secondi membri di alcune espressioni non è *un gatto che si morde la coda.* Infatti è solo una trascrizione in radianti di un angolo che ha un significato geometrico indipendente dalla quantità π. In questi esercizi, per semplificare la questione si usa però π per calcolare π, cosa che dimostra una volta in più l'abilità di Archimede, che calcolò tali quantità con metodi elementari e geometrici.
Si cita il fatto che per $k = 4$ Archimede riusci' ad affermare che

$$\frac{223}{71} \leq \pi \leq \frac{22}{7}.$$

4. **Esercizio, facile facoltativo**. Posto $h = 10^{-1}, 10^{-2}, \ldots, 10^{-15}$, si approssimi la derivata di $\exp(1)$ con il rapporto incrementale

$$\frac{\exp(1+h) - \exp(1)}{h}$$

e quindi si valuti l'errore assoluto compiuto (rispetto alla soluzione $\exp(1)$). L'approssimazione migliora al diminuire di h?

5. **Esercizio, facile facoltativo**. Si esegua una tabella in cui si studino a due a due le somme, prodotti, divisioni di NaN, Inf, 0, 1 e confrontarle con i noti risultati di indeterminazione della teoria dei limiti.

6. **Esercizio, facile facoltativo**. Fissato $\theta = 0 : (2\pi/1001) : 2\pi$, si valuti $\sin^2(\theta) + \cos^2(\theta)$ e si calcoli l'errore assoluto rispetto il valore 1.

7. Alcune frasi celebri. In [12], oltre a varie curiosità su π si cita un curioso aneddoto riguardante il noto matematico J. H. Conway:

"*Un giorno decisi di imparare a memoria le prime mille cifre del pi greco* - ricorda Conway - *stimolato da mia moglie Larissa, una matematica di origine russa, che aveva bisogno del valore di pi e non ricordava che 3,14. Le insegnai le prime cento cifre che ricordavo già a memoria. Ma questo a lei non bastava e, visto che anch'io non sapevo andare oltre, decidemmo insieme di programmare lo studio di cento nuove cifre ogni giorno, per arrivare almeno a mille, da imparare nei momenti in cui eravamo insieme, al di fuori del nostro lavoro*".

"*E' stato divertente* - continua Conway - *perchè ogni domenica facevamo una passeggiata fino a Grantchester, una graziosa, piccola cittadina vicino a Cambridge e lungo il percorso recitavamo a turno i gruppi successivi di 20 cifre del pi, come fossero piccole poesie. Venti cifre io e venti cifre mia moglie e così di seguito, alternandoci nella recita: in questo modo siamo arrivati a memorizzare le mille cifre del pi greco*".

Pure la morte di Archimede è coperta dalla leggenda [13]:

Ad un tratto entrò nella stanza un soldato e gli ordinò di andare con lui da Marcello. Archimede rispose che sarebbe andato dopo aver risolto il problema e messa in ordine la dimostrazione. Il soldato si adirò , sguainò la spada e lo uccise.

Frasi celebri attribuite ad Archimede

- *Noli turbare circulos meos* cioè *Non turbare i miei cerchi.*
- le sue ultime parole pare furono *Soldato, stia lontano dal mio disegno.*

8. Online. Alcuni siti utili per la comprensione della lezione:

1. http://it.wikipedia.org/wiki/Archimede
2. http://utenti.quipo.it/base5/numeri/pigreco.htm
3. http://it.wikipedia.org/wiki/Pi_greco
4. http://it.wikipedia.org/wiki/Teorema_fondamentale_del_calcolo_integrale
5. http://it.wikipedia.org/wiki/Floating_point
6. http://it.wikipedia.org/wiki/IEEE_754r
7. http://it.wikipedia.org/wiki/IEEE_754
8. http://mathworld.wolfram.com/ArchimedesAlgorithm.html

RIFERIMENTI BIBLIOGRAFICI

[1] K. Atkinson, *An Introduction to Numerical Analysis*, Wiley, (1989).
[2] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.
[3] V. Comincioli, *Problemi di analisi numerica*, Mc Graw-Hill, 1991.
[4] M. Frontini e E. Sormani, *Fondamenti di Calcolo Numerico, problemi in laboratorio*, Apogeo, 2005.
[5] T. Huckle , http://wwwzenger.informatik.tu-muenchen.de/persons/huckle/bugse.html.
[6] J.H. Mathews e K.D. Fink, *Numerical Methods using Matlab*, Prentice Hall, 1999.
[7] Network Theory, *GNU Octave Manual Version*, http://www.network-theory.co.uk/docs/octave3/octave_160.html.
[8] A. Quarteroni e F. Saleri, *Introduzione al calcolo scientifico*, Springer Verlag, 2006.
[9] A. Quarteroni, R. Sacco e F. Saleri, *Matematica Numerica*, Springer Verlag, 1998.
[10] J. Stoer, *Introduzione all'analisi numerica*, Zanichelli, 1984.
[11] The MathWorks Inc., *Numerical Computing with Matlab*,http://www.mathworks.com/moler.
[12] Web Page , http://utenti.quipo.it/base5/numeri/pigreco.htm.
[13] Wikipedia (Archimede), http://it.wikipedia.org/wiki/Archimede
[14] Wikipedia (Teorema Fondamentale del Calcolo Integrale), http://it.wikipedia.org/wiki/Teorema_fondamentale_del_calcolo_integrale
[15] Wikipedia (Floating Point), http://it.wikipedia.org/wiki/Floating_point
[16] Wikipedia (IEEE 754), http://it.wikipedia.org/wiki/IEEE_754
[17] Wikipedia (IEEE 754r), http://it.wikipedia.org/wiki/IEEE_754r
[18] Wikipedia (Pi Greco), http://it.wikipedia.org/wiki/Pi_greco
[19] Wolfram (Archimedes' Algorithm), http://mathworld.wolfram.com/ArchimedesAlgorithm.html