NUMERI MACCHINA *

A. SOMMARIVA†

Conoscenze richieste. Equazioni di secondo grado, successioni, formula di Taylor in una e due variabili, integrali di Riemann.

Conoscenze ottenute. In questa sezione analizziamo la rappresentazione floating point di un numero. Di seguito introduciamo il concetto di errore assoluto e relativo, condizionamento, stabilità e complessità computazionale con esempi in Matlab.

1. Numeri macchina e loro proprietà. Fissato un numero naturale $\beta > 1$ è possibile vedere che ogni numero reale $x \neq 0$ ha una unica rappresentazione del tipo

$$x = \text{sign}(x)\beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k}, \quad 0 < d_1,\, 0 \leq d_k \leq \beta - 1,\, e \in \mathbb{Z}$$

dove

$$\text{sign}(y) = \begin{cases} 1, & y > 0 \\ -1, & y < 0 \\ 0, & y = 0. \end{cases}$$

è la funzione segno. La particolarità della sommatoria è ovviamente tale che x potrebbe essere non rappresentabile esattamente dal calcolatore se esistono infiniti $d_k > 0$, in quanto il calcolatore è capace di immagazzinare solo un numero finito di tali d_k. A tal proposito, definiamo il sottoinsieme $F(\beta, t, L, U)$ di $\mathbb{R}$, costituito da quei numeri y che sono 0 oppure sono rappresentati con il *sistema a virgola mobile normalizzata o floating point normalizzato* da

$$y = \text{sign}(y) \cdot (0.d_1 \ldots d_t)\beta^e = \text{sign}(y) \cdot \beta^e \sum_{k=1}^{t} d_k \beta^{-k}, \quad d_1 > 0,\, 0 \leq d_k \leq \beta - 1$$

con $\beta > 1$ numero naturale detto *base*, t numero prefissato di cifre di *mantissa*, e l'esponente intero $e \in \mathbb{Z}$ tale che $L \leq e \leq U$. Ogni elemento di $F(\beta, t, L, U)$ è detto *numero macchina*. La parola *normalizzato* sottolinea che $d_1 > 0$. Si può provare che $\sum_{k=1}^{t} d_k \beta^{-k} < 1$ e quindi $e = \texttt{floor}(1 + \log_\beta(x))$.

Vediamo un esempio. Supponiamo di voler vedere se 100 appartiene a $F(10, 6, -6, 8)$. Essendo l'esponente corretto $e = \texttt{floor}(1 + \log_{10}(100)) = 3$, si vede subito che $100 = 10^3 \cdot 0.1 = 10^3 \cdot (0.100000)\beta^e$.

La base utilizzata dalla maggior parte dei computer attuali è $\beta = 2$, ma fino a non molti anni fa alcuni calcolatori usavano ancora la base $\beta = 10$ come l'HP-15C ($\approx$ 1980) o $\beta = 16$ nel caso dell'IBM 3033 ($\approx$ 1977) .

Di ogni numero macchina si immagazzinano una cifra per il segno, le cifre della mantissa e le cifre dell'esponente. Ognuna di queste cifre è detta *bit*. Ogni numero è registrato in vettore detto *parola* di un certo numero di M *bits*. Fissato M per la precisione *singola*, nel caso di precisione *doppia* questo è usualmente composto da 2 parole di M bits. Classici

*Ultima revisione: 20 ottobre 2011

†Dipartimento di Matematica Pura ed Applicata, Università degli Studi di Padova, stanza 419, via Trieste 63, 35121 Padova, Italia (alvise@euler.math.unipd.it). Telefono: +39-049-8271350.

esempi sono per la precisione singola $F(2, 24, -126, 128)$ in cui ogni numero macchina occupa 32 bit (o 4 byte essendo un byte pari a 8 bit) e per la doppia $F(2, 53, -1022, 1024)$ in cui ogni numero macchina occupa 64 bit. Per capirlo, vediamo cosa serva immagazzinare in precisione doppia. Essendo un numero non nullo di segno $+1$ o -1, basta un solo bit per il segno. Delle 53 cifre della mantissa, solo 52 devono essere ricordate (la prima per forza è 1 in virtù della normalizzazione). Infine, per l'esponente, vista una trattazione particolare che tiene conto anche di possibili problemi, servono 11 bit.

Osserviamo che per $e \geq t$ con t numero di cifre della mantissa, essendo $\beta > 1$ un numero naturale, allora

$$\pm\beta^e \sum_{k=1}^{t} d_k \beta^{-k} = \pm \sum_{k=1}^{t} d_k \beta^{e-k}$$

è un numero intero, essendo $e - k > 0$. Di conseguenza i numeri macchina più grandi in valore assoluto sono interi. Si vede facilmente che tutti gli interi in valore assoluto inferiori o uguali a $\beta^U \cdot (1 - \beta^{-t})$ sono in particolare nell'insieme $F(\beta, t, L, U)$.

L'insieme dei numeri macchina $F(\beta, t, L, U)$ ha un numero finito di elementi e si può dimostrare che la sua cardinalità è

$$2(\beta - 1)\beta^{t-1}(U - L + 1) + 1.$$

Di conseguenza in singola precisione i numeri sono circa $4.2789 \cdot 10^9$ mentre in precisione doppia sono circa $1.8438 \cdot 10^{19}$. Se x è il numero reale

$$x = \text{sign}(x)(0.d_1, \ldots d_t, \ldots)\beta^e = \text{sign}(x)\beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k}, \quad d_1 > 0$$

allora con $\text{fl}(x)$ denotiamo

$$fl(x) = \text{sign}(x)(0.d_1, \ldots, d_{t-1}, \overline{d_t})\beta^e = \text{sign}(x)\beta^e (\sum_{k=1}^{t-1} d_k \beta^{-k} + \overline{d_t}\beta^{-t})$$

in cui se si effettua il *troncamento*

$$\overline{d_t} = d_t, \;\; d_1 > 0$$

mentre se viene effettuato l'*arrotondamento*.

$$\overline{d_t} = \begin{cases} d_t, & d_{t+1} < \beta/2 \\ d_t + 1, & d_{t+1} \geq \beta/2 \end{cases}$$

La scelta tra troncamento o arrotondamento è eseguita a priori dal sistema numerico utilizzato dal calcolatore. Attualmente la scelta di arrotondare è la più diffusa.

Se l'esponente e del numero x è minore di L si commette un'errore di *underflow*, mentre se è maggiore di U si commette un'errore di *overflow*. Se invece $e \in [L, U]$, $x = \pm\beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k}$ e per qualche $t^* > t$ si ha $d_{t^*} > 0$ allora si commette un errore di *troncamento* o *arrotondamento* a seconda si effettui l'uno o l'altro.

Essendo un numero macchina del tipo

$$x = \pm\beta^e \sum_{k=1}^{t} d_k \beta^{-k}$$

con $d_1 > 0$ e $0 \leq d_k \leq \beta - 1$ si vede subito che

$$
\begin{aligned}
|x| &= \beta^e \sum_{k=1}^{t} d_k \beta^{-k} \leq \beta^e \sum_{k=1}^{t} (\beta-1)\beta^{-k} \\
&= \beta^e(\beta-1) \sum_{k=1}^{t} \beta^{-k} = \beta^e(\beta-1)\left(\frac{1-\beta^{-(t+1)}}{1-\beta^{-1}} - 1\right) \\
&= \beta^e(\beta-1)\left(\frac{\beta(1-\beta^{-(t+1)})}{\beta-1} - 1\right) \\
&= \beta^e(\beta-1)\left(\frac{\beta(1-\beta^{-(t+1)}) - (\beta-1)}{\beta-1}\right) \\
&= \beta^e(1-\beta^{-t}) < \beta^e \qquad (1.1)
\end{aligned}
$$

In altri termini è minore di 1 la somma relativa alla mantissa, cioè $\sum_{k=1}^{t} d_k \beta^{-k} < 1$.

Poichè , dai calcoli eseguiti si vede che $\beta^e(1-\beta^{-t})$ è un numero macchina, si deduce che il massimo numero macchina rappresentabile è quindi

$$\beta^U(1-\beta^{-t}).$$

Si vede facilmente che il più piccolo numero macchina (e quindi normalizzato!) positivo rappresentabile è β^{L-1}.

Si dice *precisione di macchina*, quel numero ξ_m che rappresenta la distanza tra 1 ed il successivo numero in virgola mobile. Calcoliamo tale valore. Necessariamente il numero successivo a 1 ha per mantissa $d_1 = 0, \ldots, d_{t-1} = 0, d_t = 1$ e esponente $e = 1$ (perchè deriva da (1.1)?) e quindi è $x = \beta^1 \cdot \beta^{-t} = \beta^{1-t}$. Si sottolinea che la precisione di macchina non coincide in generale con il più piccolo numero rappresentabile dal calcolatore. Più in generale la distanza tra $x = \text{sign}(x)\beta^e \sum_{k=1}^{t} d_k \beta^{-k}$ e il floating point successivo è uguale a $\text{sign}(x)\beta^{e-t}$ e quindi i numeri floating point non sono equispaziati.

In Matlab ad esempio, il valore minimo in modulo è

$$x_{min} = 2.225073858507201 \cdot 10^{?308}$$

e il massimo

$$x_{max} = 1.7976931348623158 \cdot 10^{+308}$$

. Come anticipato, un numero più piccolo di x_{min} produce un underflow. Di solito questa situazione è trattata in modo particolare. Un numero più grande di x_{max} invece fornisce un messaggio di overflow e di solito il numero è immagazzinato come `Inf`. Il numero macchina più prossimo a x_{min} è

$$2.225073858507202 \cdot 10^{?308}$$

$$1.797693134862315710 + 308$$

e di conseguenza i numeri macchina non sono equispaziati.

1.1. Errori relativi e assoluti. 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. Inoltre se il vettore ha un solo componente, allora la norma $\|\cdot\|$ coincide con l'usuale valore assoluto $|\cdot|$.

Sia

$$x = \text{sign}(x)\beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k} \neq 0, \quad d_1 > 0$$

un numero reale e

$$fl(x) = \text{sign}(x)(0.d_1, \ldots, d_{t-1}, \overline{d_t})\beta^e = \text{sign}(x)\beta^e (\sum_{k=1}^{t-1} d_k \beta^{-k} + \overline{d_t}\beta^{-t}), \quad d_1 > 0$$

la sua approssimazione in virgola mobile.

L'errore relativo compiuto nell'approssimare x con $fl(x)$, visto che $|x| \geq |d_1 \beta^{e-1}| \geq |\beta^{e-1}|$ poichè $d_1 > 0$, verifica

$$\begin{aligned} \frac{|x - fl(x)|}{|x|} &\leq \frac{|\text{sign}(x)\beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k} - (\text{sign}(x)\beta^e (\sum_{k=1}^{t-1} d_k \beta^{-k} + \overline{d_t}\beta^{-t}))|}{|\beta^{e-1}|} \\ &= \beta \cdot \left| \sum_{k=t}^{+\infty} d_k \beta^{-k} - \overline{d_t}\beta^{-t} \right| \end{aligned} \tag{1.2}$$

Nel caso si effettui il troncamento abbiamo $d_t = \overline{d_t}$ e quindi, essendo $\xi_M = \beta^{1-t}$ la precisione di macchina

$$\beta \cdot \left| \sum_{k=t}^{+\infty} d_k \beta^{-k} - \overline{d_t}\beta^{-t} \right| = \beta \cdot \left| \sum_{k=t+1}^{+\infty} d_k \beta^{-k} \right| < \beta \cdot (\beta \cdot \beta^{-t-1}) = \beta^{1-t} = \xi_M$$

mentre se si effettua l'arrotondamento abbiamo dopo alcuni calcoli che

$$\frac{|x - fl(x)|}{|x|} < \frac{\beta^{1-t}}{2} = \frac{\xi_M}{2}.$$

Il minor errore relativo compiuto nell'approssimare x con $fl(x)$ via arrotondamento, suggerisce perchè quest'ultimo sistema sia più utilizzato. Da questo 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.

2. Facoltativo: Rappresentazione IEEE 754 dei numeri al calcolatore. La rappresentazione numerica nei moderni calcolatori rispetta 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 sono memorizzati in un vettore detto *parola* di 32 bits, mentre quelli a precisione doppia occupano due parole consecutive da 32 bits. Un numero *non nullo-normalizzato* si scrive come

$$x = (-1)^S \cdot 2^{E-Bias} \cdot (1 + F) \tag{2.1}$$

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* con $F = \sum_{k=1}^{+\infty} a_k 2^{-k}$, con $a_k = 0$ oppure $a_k = 1$. Evidentemente, in un calcolatore non si possono immagazzinare un numero infinito di a_k, e ci si limita ai cosidetti *numeri macchina*

$$x = (-1)^S \cdot 2^{E-Bias} \cdot (1 + F) \tag{2.2}$$

con E intero tale che $|E| < 2^{N_E}$ per un certo prefissato N_E e $F = \sum_{k=1}^{N_F} a_k 2^{-k}$,

Alcuni esempi

Precisione	N_E	N_F	Bias
Singola	8	23	127
Doppia	11	52	1023

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!) $E = 10000010 = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 0 \cdot 2^5 + 0 \cdot 2^6 + 1 \cdot 2^7 = 2 + 128 = 130$, 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 (2.2). Quindi $F = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + \ldots = 0.5$. 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 $12 = 8 \cdot 1.5$.

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!) $E = 10000101 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 0 \cdot 2^5 + 0 \cdot 2^6 + 1 \cdot 2^7 = 1 + 4 + 128 = 133$, 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 (2.2). Quindi $F = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-1} + \ldots = 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 = 0.96875$. 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 $126 = 64 \cdot 1.96875$.

La distanza tra un numero in virgola mobile x^- ed il successivo x^+, è

$$2^{-N_F} \cdot 2^{E-bias}.$$

Per convincersene, supponiamo che siano

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

Allora, essendo $F^+ - F- = 2^{-N_F}$ (pensarci su!) facilmente

$$|x^+ - x^-| = 2^{E-Bias} \cdot (F^+ - F-) = 2^{-N_F} \cdot 2^{E-bias}.$$

La cardinalità dei numeri macchina in IEEE 754 è piuttosto elevata. Ad esempio, in doppia precisione sono

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

La *precisione di macchina*, cioè quel numero che rappresenta la distanza tra 1 ed il successivo numero in virgola mobile. Per avere un'idea del suo ordine di grandezza, tale valore `eps` in Matlab 7.6.0 vale

```
>> format long
>> eps
ans =
     2.220446049250313e-16
>>
```

2.1. Facoltativo: Su `eps`, `realmax`, `realmin`, overflow e underflow in IEEE754. 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-bias} ed in Matlab è espresso dalla variabile `realmin`. Il limite superiore è pari a $2^{2^{N}_{E}-2-bias} \cdot (1+(1-2^{\text{-Fbits}}))$ 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*$= 1023$, $N_E = 11$, ed $N_F = 52$. Conseguentemente

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

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 \cdot 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

$$x = (-1)^S \cdot 2^{E-Bias} \cdot (0.F). \tag{2.3}$$

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 \cdot 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 . \cdot  (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.

>>
```

3. Operazioni con i numeri macchina. 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 $\mathrm{fl}(x)$ è il numero macchina che *corrisponde* a x, denotiamo con

$$x \oplus y = \mathrm{fl}(\mathrm{fl}(x) + \mathrm{fl}(y)) \tag{3.1}$$
$$x \ominus y = \mathrm{fl}(\mathrm{fl}(x) - \mathrm{fl}(y)) \tag{3.2}$$
$$x \otimes y = \mathrm{fl}(\mathrm{fl}(x) \cdot \mathrm{fl}(y)) \tag{3.3}$$
$$x \oslash y = \mathrm{fl}(\mathrm{fl}(x) : \mathrm{fl}(y)) \tag{3.4}$$

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

$$\epsilon_x = \frac{|x - \mathrm{fl}(x)|}{|x|} \tag{3.5}$$
$$\epsilon_{x,y}^{\odot} = \frac{|(x \,\mathrm{op}\, y) - (x \odot y)|}{|x \,\mathrm{op}\, y|} \tag{3.6}$$

Per prima cosa osserviamo che alcune proprietà caratteristiche di $+$, $-$, $\cdot$ e $:$ non valgono per i corrispettivi $\oplus$, $\ominus$, $\otimes$, $\oslash$. Infatti per la somma $\oplus$ e il prodotto $\otimes$ vale la proprietà commutativa ma non vale quella associativa, e nemmeno quella distributiva.

Vediamo alcuni esempi. Mostriamo come non valga la proprietà associativa della somma. Consideriamo $F(10, 4, -10, +10)$ con *fl* basato sul troncamento. Sia $a = 2000$, $b = 2.5$, $c = 7.8$. Allora:

$$(a \oplus b) \oplus c = 0.2002 \cdot 10^4 \oplus 0.7800 \cdot 10^1 = 0.2009 \cdot 10^4$$
$$a \oplus (b \oplus c) = 0.2000 \cdot 10^4 \oplus 0.1030 \cdot 10^2 = 0.2010 \cdot 10^4$$

e quindi $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$. Per quanto riguarda la proprietà distributiva, in $F(10, 4, -10, +10)$, posti $a = 0.800000 \cdot 10^9$, $b = 0.500009 \cdot 10^4$, $c = 0.500008 \cdot 10^4$

$$\begin{aligned} a \otimes (b \ominus c) &= 0.800000 \cdot 10^9 \otimes 0.1 \cdot 10^{-2} = 0.800000 \cdot 10^9 \\ (a \otimes b) \ominus (a \otimes c) &= (0.800000 \cdot 10^9 \otimes b = 0.500009 \cdot 10^4) \ominus \\ &\quad \ominus (0.800000 \cdot 10^9 \otimes b = 0.500008 \cdot 10^4) \to \text{overflow} \end{aligned} \tag{3.7}$$

(l'overflow è dovuto alla sequenza di operazioni da compiere).

3.0.1. Errori nelle operazioni e loro propagazione. Consideriamo ora come si propagano gli errori in $\oplus$, $\ominus$, $\otimes$, $\oslash$. Dopo qualche tedioso conto (cf. [2, p.78])

$$\epsilon_{x,y}^{\oplus} \leq \left|\frac{x}{x+y}\right|\epsilon_x + \left|\frac{y}{x+y}\right|\epsilon_y \tag{3.8}$$

$$\epsilon_{x,y}^{\ominus} \leq \left|\frac{x}{x-y}\right|\epsilon_x + \left|\frac{y}{x-y}\right|\epsilon_y \tag{3.9}$$

$$\epsilon_{x,y}^{\otimes} \lessapprox \epsilon_x + \epsilon_y \tag{3.10}$$

$$\epsilon_{x,y}^{\oslash} \leq |\epsilon_x - \epsilon_y| \tag{3.11}$$

il che mostra il pericolo di perdita di accuratezza nella somma e nella sottrazione qualora rispettivamente $x+y \approx 0$ e $x-y \approx 0$ (fenomeno di *cancellazione*). Vediamone un esempio. Consideriamo il sistema $F(10, 5, -5, +5)$ (con arrotondamento) e i due numeri

$$a = 0.73415776, \qquad b = 0.73402350.$$

Naturalmente,

$$\text{fl}(a) = 0.73416, \qquad \text{fl}(b) = 0.73402$$

e quindi

$$a - b = 0.00013426, \qquad a \ominus b = 0.00014 = 10^{-3} \cdot 0.14000$$

con un errore relativo pari a

$$|(a-b) - (a \ominus b|)/|a-b| \approx 4.3/100$$

cioè molto grande (oltre il 4 per cento!).

Proviamo di seguito il caso della somma e della moltiplicazione, lasciando al lettore quelli di sottrazione e divisione. Con $a \lessapprox b$ si intende a *inferiore* o uguale circa a b.

Per quanto riguardo la somma, supposti $x+y \neq 0$, $x \neq 0$, $y \neq 0$, e $\text{fl}(\text{fl}(x)+\text{fl}(y)) = \text{fl}(x)+\text{fl}(y)$ (cioè $\text{fl}(x)+\text{fl}(y)$ numero macchina, richiesta non eccessiva), abbiamo

$$\begin{aligned}
\epsilon_{x,y}^{\oplus} &= \frac{|(x+y)-(x\oplus y)|}{|x+y|} = \frac{|(x+y)-\text{fl}(\text{fl}(x)+\text{fl}(y))|}{|x+y|} = \frac{|(x-\text{fl}(x))+(y-\text{fl}(y))|}{|x+y|} \\
&\leq \frac{|x|}{|x+y|}\cdot\frac{|x-\text{fl}(x)|}{|x|} + \frac{|y|}{|x+y|}\cdot\frac{|y-\text{fl}(y)|}{|y|} = \left|\frac{x}{x+y}\right|\epsilon_x + \left|\frac{y}{x+y}\right|\epsilon_y
\end{aligned} \tag{3.12}$$

Il caso in cui $\text{fl}(\text{fl}(x)+\text{fl}(y)) \neq \text{fl}(x)+\text{fl}(y)$ è più complicato ma i risultati restano simili dal punto di vista teorico.

Nel caso del prodotto, supposti $x+y \neq 0$, $x \neq 0$, $y \neq 0$, e $\text{fl}(\text{fl}(x)\cdot\text{fl}(y)) = \text{fl}(x)\cdot\text{fl}(y)$ (cioè $\text{fl}(x)\cdot\text{fl}(y)$ numero macchina), abbiamo

$$\begin{aligned}
\epsilon_{x,y}^{\otimes} &= \frac{|x\cdot y - x\otimes y|}{|x\cdot y|} = \frac{|x\cdot y - \text{fl}(\text{fl}(x)\cdot\text{fl}(y))|}{|x\cdot y|} = \frac{|x\cdot y - \text{fl}(x)\cdot\text{fl}(y)|}{|x\cdot y|} \\
&= \frac{|x\cdot y - x\cdot\text{fl}(y) + x\cdot\text{fl}(y) - \text{fl}(x)\cdot\text{fl}(y)|}{|x\cdot y|} \\
&\leq \frac{|x\cdot(y-\text{fl}(y))| + |\text{fl}(y)\cdot(x-\text{fl}(x))|}{|x\cdot y|} \\
&= \frac{|y-\text{fl}(y)|}{|y|} + \frac{|\text{fl}(y)\cdot(x-\text{fl}(x))|}{|x\cdot y|} \\
&\approx \epsilon_x + \epsilon_y
\end{aligned} \tag{3.13}$$

dove si ricorda che

$$\frac{|\mathrm{fl}(y)|}{|x \cdot y|} \approx \frac{1}{|x|}.$$

NOTA 3.1. *Osserviamo che dalla formula di Taylor in due variabili, posto per brevità di notazione $f'_x = \partial f/\partial x$, $f'_y = \partial f/\partial y$,*

$$f(x,y) \approx f(\overline{x},\overline{y}) + f'_x(\overline{x},\overline{y})(x-\overline{x}) + f'_y(\overline{x},\overline{y})(y-\overline{y}) \tag{3.14}$$

necessariamente, se $x, y, f(x,y) \neq 0$ allora posti $\epsilon_x = |x-\overline{x}|/|x|$, $\epsilon_y = |y-\overline{y}|/|y|$ gli errori relativi sui dati

$$\begin{aligned}\frac{|f(x,y)-f(\overline{x},\overline{y})|}{|f(x,y)|} &\lessapprox \frac{|f'_x(\overline{x},\overline{y})||x-\overline{x}||x|}{|f(x,y)||x|} + \frac{|f'_y(\overline{x},\overline{y})||y-\overline{y}||y|}{|f(x,y)||y|} \\ &= \frac{|f'_x(\overline{x},\overline{y})||x|}{|f(x,y)|}\cdot\epsilon_x + \frac{|f'_y(\overline{x},\overline{y})||y|}{|f(x,y)|}\cdot\epsilon_y \end{aligned} \tag{3.15}$$

Da (3.15), si possono ottenere risultati simili ai precedenti per $\oplus$, $\otimes$.

4. Valutazione di una funzione. Nel valutare una funzione continua $f : \Omega \subseteq \mathbb{R} \to \mathbb{R}$ in un punto $x \in \Omega$ bisogna tener conto di vari dettagli. Per prima cosa il dato x può essere per varie ragioni affetto da errori, tipicamente di misura o di rappresentazione al calcolatore e conseguentemente approssimato con un certo $\overline{x}$. Di conseguenza, spesso invece di $f(x)$ si valuta $f(\overline{x})$. Ci si domanda se ci siano considerevoli differenze nel valutare la funzione in $\overline{x}$ invece che in x.

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 discutere la quantità

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

La sua valutazione in un punto x si dice *bencondizionata* se a piccole variazioni relative sui dati corrispondono piccole variazioni sui risultati, *malcondizionata* se ciò non avviene. Se f è derivabile con continuità nel più piccolo intervallo $\mathcal{I}$ aperto contenente x ed x_c, per il teorema della media, essendo

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

e quindi da $|f(x) - f(x_c)|/|x - x_c| = f'(\xi) \approx f'(x)$ ricaviamo facilmente che

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

La quantità $\mathcal{K}(f,x)$ si chiama *condizionamento* di f nel punto x. Più piccola risulta $\mathcal{K}(f,x)$ e meno la funzione amplifica un piccolo errore sul dato x. Naturalmente la funzione può essere *bencondizionata* su un certo dato $x_1 \in \Omega$ e *malcondizionata* su un certo altro $x_2 \in \Omega$. Osserviamo che il condizionamento di una funzione non dipende dall'algoritmo con cui viene valutata, ma è inerente alla funzione stessa f e al dato x.

4.1. Esercizio sul condizionamento. 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*)?

5. Stabilità. Nella sezione precedente abbiamo analizzato il condizionamento di una funzione continua f in un punto x del suo dominio. In particolare, abbiamo supposto che f sia valutabile esattamente e quindi il bencondizionamento dipende esclusivamente da f e dal dato x e non dall'algoritmo utilizzato. Un algoritmo per la risoluzione di un certo problema si dice *stabile* se amplifica *poco* gli errori di arrotondamento introdotti dalle singole operazioni. Il problema non è necessariamente la valutazione di una funzione continua in un punto, ma può essere la risoluzione di una equazione di secondo grado, il calcolo di π greco, la valutazione di una successione, etc. Osserviamo inoltre che nel caso della valutazione in un punto x di una funzione f bencondizionata, si possono ottenere risultati non sufficientemente corretti se il problema è affrontato con un algoritmo instabile.

5.1. Consistenza, stabilità e convergenza. Vediamo i dettagli del concetto di stabilità e li legheremo ad altre due proprietà, quelle di *consistenza* e *convergenza* (cf.[9, p.39-41]). Supponiamo di voler risolvere il problema

$$F(x, d) = 0, \tag{5.1}$$

e che questo sia ben posto, cioè a piccole variazioni di d corrispondano piccole variazioni di x. Nella precedente scrittura, la variabile d è fissata e si cerca x che risolva il problema (5.1).

Un metodo numerico per la soluzione approssimata di $F(x, d)$ costruisce una successione di problemi

$$F_n(x_n, d_n) = 0, \quad n \geq 1$$

dipendenti da n, col desiderio che sia

$$x_n \to x, \quad n \to \infty.$$

Se il dato d è ammissibile per F_n, cioè ha senso risolvere $F_n(x, d) = 0$, si dice che il metodo è *consistente* se per x soluzione del problema $F(x, d) = 0$ si ha

$$F_n(x, d) = F_n(x, d) - F(x, d) \to 0, \quad n \to \infty.$$

Il metodo si dice *fortemente consistente* se in particolare

$$F_n(x, d) = F_n(x, d) - F(x, d) = 0, \quad \forall n.$$

Come detto in precedenza, un metodo è stabile (talvolta si dice *ben posto*) se per ogni n fissato esiste e sia unica la soluzione x_n del problema $F(x_n, d_n) = 0$ e che x_n dipenda con continuità dai dati cioè

$$\forall \eta > 0, \ \ \exists K_n(\eta, d_n) : \ \ \|\delta d_n\| < \eta \Rightarrow \|\delta x_n\| \leq K_n(\eta, d_n)\|\delta d_n\|.$$

Il proposito di un metodo è ovviamente di costruire, attraverso problemi numerici

$$F_n(x_n, d_n) = 0,$$

una successione di x_n che si avvicini sempre più a x al crescere di n, cioè il metodo sia *convergente*. In termini più rigorosi, se $F(x(d), d) = 0$ e $F_n(x_n(d + \delta d_n), d + \delta d_n) = 0$

$$\begin{aligned} &\forall \epsilon > 0 \ \ \exists n_0(\epsilon), \exists \delta(n_0, \epsilon) > 0 : \forall n > n_0(\epsilon), \forall \|\delta d_n\| < \delta(n_0, \epsilon) \\ &\quad \Rightarrow \|x(d) - x_n(d + \delta d_n)\| \leq \epsilon. \end{aligned} \tag{5.2}$$

Il teorema di Lax-Richtmyer mostra come queste tre proprietà siano interconnesse, cioè

TEOREMA 5.1. *Per un metodo numerico consistente, la stabilità è equivalente alla convergenza.*

5.2. Esempio 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

$$y = -p + \sqrt{p^2 + q}. \tag{5.3}$$

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

$$y = -p \pm \sqrt{p^2 + q}. \tag{5.4}$$

dell'equazione sono reali. La soluzione descritta in (5.3) è la maggiore delle 2, ed è non negativa se e solo se $q \geq 0$. Consideriamo la funzione che fissato $q \geq 0$, mappa un numero reale p in $-p \pm \sqrt{p^2 + q}$. Si osserva subito che (5.3) 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 (5.3). In altri termini

$$y = -p + \sqrt{p^2 + q} = \frac{(-p + \sqrt{p^2 + q})(p + \sqrt{p^2 + q})}{(p + \sqrt{p^2 + q})} = \frac{q}{(p + \sqrt{p^2 + q})} \tag{5.5}$$

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 (5.4) non è *numericamente stabile* qualora $p \gg q > 0$;
2. il secondo algoritmo (5.5) 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 effettivemente $p \gg q$. 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 \cdot 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]: 0.00000899999999772772
[ALG.2] [1]: 0.00000900000000000000
[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 \cdot 10^(+4); q=10^(-2); sol=10^(-7);
```

5.4. Esempio 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].
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).
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;
```

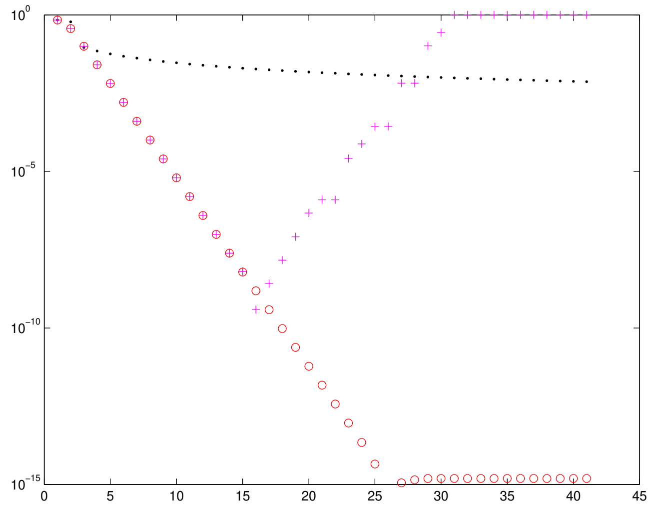

FIGURA 5.1. *Grafico che illustra le 3 successioni, rappresentate rispettivamente da* `.`, + *e* o.

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).

5.7. Esempio 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

$$I_n = e^{-1} \int_0^1 x^n \, e^x \, dx \tag{5.6}$$

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 $I_1 = e^{-1}$ e più in generale integrando per parti che

$$I_{n+1} = e^{-1}\left(x^{n+1}\,e^x\,|_0^1 - (n+1)\int_0^1 x^n\,e^x\,dx\right) = 1-(n+1)\,I_n. \tag{5.7}$$

Da (5.6) 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 (5.7), 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".
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.
subplot(2,1,1)
semilogy(1:length(s),abs(s),'k-'); hold on;
semilogy(1:length(s),abs(t(1:length(s))),'m-');
hold off;

% ANALISI DI t(1) PER VALORI DIFFERENTI DI "m".
t_1_exact=exp(-1);
for m=1:10

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

    for n=M(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);
```

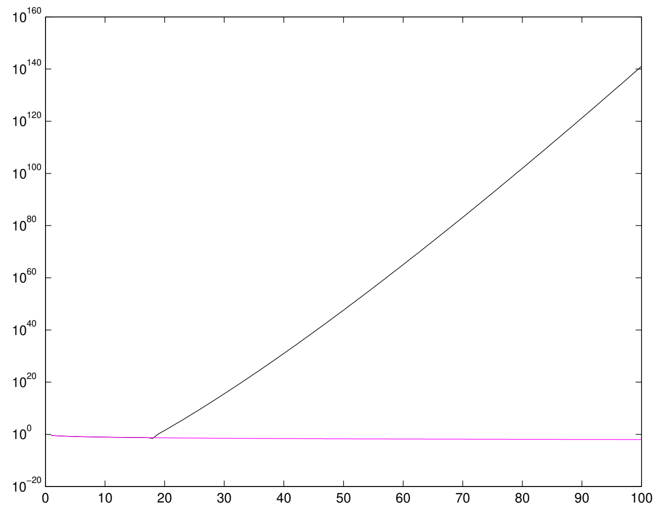

FIGURA 5.2. *Grafico che illustra i valori assoluti assunti dalla successione in avanti (in nero) e all'indietro (in rosa magenta)*

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

end

subplot(2,1,2)
semilogy(M,abs(val_t_1),'k-');
```

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 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

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

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

  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.
```

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 secondo grafico (si veda la Figura 5) per vari $M = 2 \cdot m$ mostra il comportamento del metodo all'indietro nell'appossimare il valore iniziale $\exp -1$. Evidentemente, al crescere di m, l'approssimazione diventa sempre più accurata. In particolare l'errore assoluto compiuto è stampato nel display 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

>>
```

6. Complessità computazionale. Ricordato che un *algoritmo* è una successione finita di istruzioni assegnate in modo non ambiguo, la cui esecuzione consenta di passare da una situazione iniziale (dati) ad una finale (risultati) in un tempo finito, viene naturale richiedere che esso sia adatto a risolvere una classe di problemi e che sia *ottimale* rispetto al tempo, al numero di operazioni, alla stabilità e all'accuratezza. Ovviamente ad un maggior numero di operazioni non è detto che corrisponda una miglior accuratezza dei risultati. La complessità computazionale conta il numero di operazioni necessarie per fornire l'approssimazione desiderata.

6.1. Esempio: algoritmo di Horner.. Ci poniamo il problema di valutare il polinomio

$$p(x) = a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n \tag{6.1}$$

in un punto x.

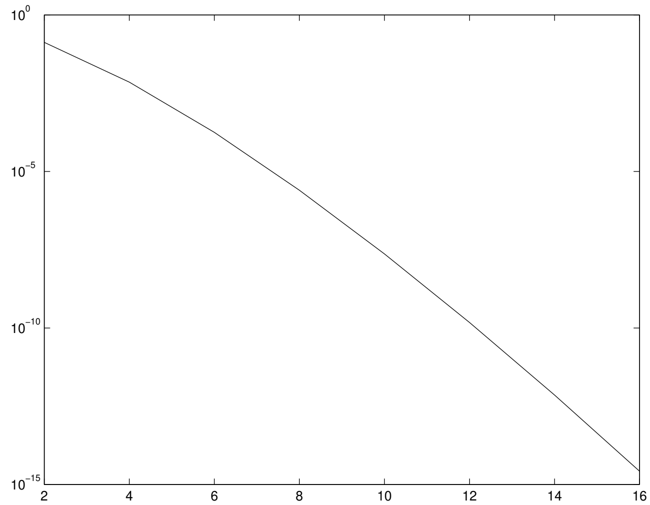

FIGURA 5.3. *Grafico che illustra il valore assoluto della successione all'indietro nell'approssimare* $\exp(-1)$.

Osserviamo che

$$p(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + \ldots + x \cdot (a_{n-1} + x \cdot a_n))) \qquad (6.2)$$

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 (6.1), il secondo che effettua la stessa operazione come descritto in (6.2) calcolando dapprima $s_1 = a_{n-1} + x \cdot a_n$, poi $s_2 = a_{n-2} + x \cdot s_1$ e così via. In Matlab avremo allora

```
function s=algoritmo1(a,x)

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

e

```
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
```

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

```
clear all;

a=[1 2 3 4];
x=pi;
```

```
y1=algoritmo1(a,x);
y2=algoritmo2(a,x);

format long;
y1
y2
```

otteniamo

```
>> demo_horner
ans =
     1.609171052316469e+02
y1 =
     1.609171052316469e+02
y2 =
     1.609171052316469e+02
>>
```

La differenza sta nella complessità computazionale e non nel risultato numerico. Il primo codice richiede $2n$ moltiplicazioni e n somme, mentre il secondo algoritmo n moltiplicazioni e n somme.

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

$$x^2 - 2\sqrt{3}x + 3 = 0 \tag{7.1}$$

Si nota subito che

$$x^2 - 2\sqrt{3}x + 3 = (x - \sqrt{3})^2 \tag{7.2}$$

e quindi (7.2) 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

$$P_\epsilon(z) = P(z) + \epsilon g(z), \;\; \epsilon > 0 \tag{7.3}$$

allora se α è semplice

$$\alpha(\epsilon) \approx \alpha + \epsilon \left(\frac{-g(\alpha)}{P^1(\alpha)} \right) \tag{7.4}$$

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

$$\alpha(\epsilon) \approx \alpha + \epsilon^{1/m} \left(\frac{-m! g(\alpha)}{P^{(m)}(\alpha)} \right) \tag{7.5}$$

Approssimiamo in (7.2) 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

$$\alpha(\epsilon) \approx \alpha + \epsilon^{1/m} \left(\frac{-m!\alpha}{P^{(2)}(\alpha)} \right) \tag{7.6}$$

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

$$\alpha(\epsilon) \approx \alpha + (5 \cdot 10^{-5})^{1/2} \left(\frac{-2!2\sqrt{3}}{2} \right) \tag{7.7}$$

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
>>
```

8. Esercizio. Date le funzioni

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

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

si calcoli analiticamente il *condizionamento*

$$\mathcal{K}_1(x) = \frac{|x \cdot f_1'(x)|}{|f_1(x)|}$$

$$\mathcal{K}_2(x) = \frac{|x \cdot f_2'(x)|}{|f_2(x)|}$$

Utilizzando Matlab/Octave, si plottino in scala semilogaritmica i grafici di $\mathcal{K}_1$ e $\mathcal{K}_2$ 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*)?

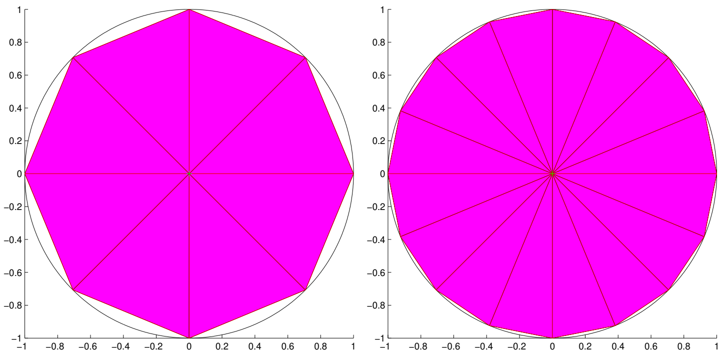

FIGURA 9.1. *Grafico che illustra le suddivisioni considerate dall'algoritmo di Archimede, per* $p = 3$, $p = 4$.

9. 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

$$I_p = 2^p \frac{\sin(\theta_p)}{2} = 2^p \frac{\sin\left(\frac{2\pi}{2^p}\right)}{2}.$$

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 9.2. *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

$$3.00000 \leq \pi \leq 3.46410 \tag{9.1}$$
$$3.10583 \leq \pi \leq 3.21539 \tag{9.2}$$
$$3.13263 \leq \pi \leq 3.15966 \tag{9.3}$$
$$3.13935 \leq \pi \leq 3.14609 \tag{9.4}$$
$$3.14103 \leq \pi \leq 3.14271 \tag{9.5}$$

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.

10. 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.*

11. 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