

# Numeri macchina<sup>1</sup>

A. Sommariva<sup>2</sup>

---

*Keywords:* Errore di troncamento e di arrotondamento, rappresentazione floating-point dei reali, precisione di macchina; operazioni aritmetiche con numeri approssimati, potenziale instabilità della sottrazione, stabilizzazione della formula risolutiva per equazioni di secondo grado, condizionamento di funzioni, propagazione degli errori in algoritmi iterativi (successione di Archimede per il calcolo di pigreco), esempi di costo computazionale degli algoritmi numerici: schema di Hoerner per i polinomi, potenza rapida tramite codifica binaria dell'esponente, calcolo della funzione  $\exp(x)$  con scalatura della variabile e formula di Taylor, calcolo di determinanti.

---

*Revisione:* 7 gennaio 2019

---

## 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

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

dove

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

è la funzione segno.

Dall'avvio dello standard IEEE 754-1985, in cui venne realizzato lo standard per l'aritmetica binari floating point, la base più utilizzata risulta  $\beta = 2$ , seppure col nuovo standard 854 si stiano avviando studi anche relativamente a  $\beta = 10$ . [4].

Vediamo quale esempio come rappresentare il numero  $\pi$  al variare di due classiche scelte di  $\beta$ :

- base 10: 3,141592653589793238462643383279502884197169399... secondo la rappresentazione sopra descritta e'

$$+1 \cdot 10^1 \cdot (3 \cdot 10^{-1} + 1 \cdot 10^{-2} + 4 \cdot 10^{-3} + \dots)$$

che viene spesso denotato come

$$\pi = (+1) \cdot (0.3141592653589793238462643383279502884197169399 \dots)_{10} \cdot 10^1;$$

- base 2: 11, 0010010000111111011010101000100010000101 ... secondo la rappresentazione sopra descritta e'  
 $+1 \cdot 2^2 \cdot (1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} \dots + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} \dots) = +4(1/2 + 1/4 + 1/32 + \dots),$

che viene spesso denotato come

$$\pi = (+1) \cdot (0.110010010000111111011010101000100010000101 \dots)_2 \cdot 2^2.$$

La particolarità della sommatoria in (1), ovvero

$$x = \text{sign}(x) \beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k}, \quad 0 < d_1, 0 \leq d_k \leq \beta - 1, e \in \mathbb{Z} \quad (2)$$

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  $d_k$ .

Ad esempio, non è difficile vedere che:

- i numeri irrazionali come  $\pi$ , indipendentemente dalla base  $\beta$ , hanno sempre un numero infinito di cifre  $d_k$ ;
- in base 10 alcuni numeri razionali hanno comunque un numero infinito di cifre  $d_k$  (si pensi ai numeri periodici come  $1/3 = 0.333 \dots$ , ma comunque in certe basi possono avere una rappresentazione finita (nel nostro esempio  $1/3 = 1 \cdot 3^{-1}$  per  $\beta = 3$ ).

**Nota. 1.1.** Osserviamo che storicamente, la scelta della base  $\beta$  è risultata molto eterogenea:

- la base utilizzata dalla maggior parte dei computer attuali è  $\beta = 2$ ,
- il computer russo SETUN ( $\approx 1957$ ) usava la base  $\beta = 3$ ,
- il PDP-10 ( $\approx 1965$ ), Burroughs 570 e 6700 utilizzavano la base ottale, cioè  $\beta = 8$ ,
- la base che usiamo nella vita di tutti i giorni è  $\beta = 10$ , ed è stata una scelta comune anche per computer del passato come l'HP-11C, HP-15C ( $\approx 1980$ ) o Eniac ( $\approx 1947$ ),
- esistono popoli in Nigeria, India, Nepal, la cui è base  $\beta = 12$ ,
- molti computers, come l'IBM 360, l'IBM 3033 ( $\approx 1977$ ), usavano la base  $\beta = 16$ ,
- i Maya usavano la base  $\beta = 20$ .
- in tavolette babilonesi si utilizzava la base  $\beta = 60$ .

### 1.1. Numeri floating-point

In virtù di di considerazioni sulla finitezza delle cifre immagazzibili, ipotesi necessaria perché il numero sia utilizzabile da un computer, risulta naturale la seguente

**Definizione.** L'insieme dei numeri macchina  $F(\beta, t, L, U)$  di  $\mathbb{R}$  è costituito da quei numeri  $y$  che sono 0 oppure sono rappresentati con il *sistema a virgola mobile normalizzata o floating point normalizzato* da

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

con

- $\beta > 1$  numero naturale detto *base*,  $t$  numero prefissato di cifre di *mantissa*,
- $e \in \mathbb{Z}$  l'esponente intero tale che  $L \leq e \leq U$ .

La parola *normalizzato* sottolinea che  $d_1 > 0$ . Si può provare che  $\sum_{k=1}^t d_k \beta^{-k} < 1$  e quindi  $e = \text{floor}(1 + \log_\beta(x))$ .

Ogni elemento di  $F(\beta, t, L, U)$  è detto *numero macchina*, ed evidentemente è un numero razionale.

**Nota. 1.2.** Osserviamo che se  $e > 0$  allora

$$\begin{aligned} y &= \text{sign}(y) \cdot (0.d_1 \dots d_t)_\beta \beta^e = \text{sign}(y) \cdot \beta^e \sum_{k=1}^t d_k \beta^{-k} \\ &= \text{sign}(y) \cdot \underbrace{\left( \sum_{k=1}^e d_k \beta^{e-k} \right)}_{\text{parte intera}} + \underbrace{\left( \sum_{k=e+1}^t d_k \beta^{e-k} \right)}_{\text{parte frazionaria}} \end{aligned}$$

dove l'ultima somma è nulla per definizione se  $t < e + 1$ .

**Nota. 1.3.** Si osservi che il sistema *floating-point* è solo una delle scelte possibili, in quanto una volta si utilizzava un sistema a virgola fissa, ovvero con un numero fissato di cifre prima della virgola e un certo numero di cifre dopo la virgola, abbandonato (nonostante presentasse alcuni vantaggi) perché l'intervallo di valori rappresentati è modesto e la precisione dei numeri frazionari scarsa.

Di ogni numero macchina si immagazzinano

- una cifra per il segno (ovvero 0 per positivo e 1 per negativo),
- le cifre dell'esponente,
- le cifre della mantissa.

segno	esponente	mantissa
1 cifra	$m$ cifre	$t$ cifre

Ognuna di queste cifre è detta *bit*. Ogni numero è registrato in un vettore detto *parola* di un certo numero di  $M$  *bits*.

**Nota. 1.4.** *Quello che proponiamo è un modello dei numeri floating-point. Le implementazioni sono molte più complesse e posso avere qualche trascurabile scostamento da quanto affermato.*

*Ad esempio, nella notazione binaria normalizzata, immagazzinate le cifre  $(1, d_1, \dots, d_{t-1})_\beta$  e l'esponente  $e$ , a volte si intende*

$$y = \text{sign}(y) \cdot (1.d_2 \dots d_t)_\beta \beta^e = \text{sign}(y) \cdot \beta^e \left(1 + \sum_{k=2}^t d_k \beta^{-k}\right) \quad (3)$$

*e non come in precedenza*

$$y = \text{sign}(y) \cdot (1.d_1 \dots d_{t-1})_\beta \beta^e = \text{sign}(y) \cdot \beta^e \left(\beta^{-1} + \sum_{k=2}^t d_{k-1} \beta^{-k}\right). \quad (4)$$

*Questo potenzialmente può creare qualche confusione. Di seguito scegliamo la notazione più utilizzata nei manuali di analisi numerica, ovvero (4).*

## 1.2. Alcune proprietà di $F(\beta, t, L, U)$

Vediamo di seguito, alcune proprietà di  $F(\beta, t, L, U)$ .

**Proposizione 1.5.** *Il più piccolo numero macchina positivo (normalizzato) è  $\beta^{L-1}$ .*

**Dimostrazione.** Ponendo  $d_1 = 1, d_2 = \dots = d_t = 0$  ed  $e = L$  otteniamo che il più piccolo numero macchina (e quindi normalizzato!) positivo rappresentabile è  $\beta^{L-1}$ .

**Proposizione 1.6.** *Il più grande numero macchina positivo (normalizzato) è  $\beta^U(1 - \beta^{-t})$ , dove  $t$  è il numero di cifre della mantissa.*

**Dimostrazione.** Ponendo  $d_k = \beta - 1$ , per  $k = 1, \dots, t$  ed  $e = U$  otteniamo che il massimo numero macchina rappresentabile è

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

**Proposizione 1.7.** *La somma relativa alla mantissa, cioè*

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

*appartiene all'intervallo  $(0, 1 - \beta^{-t}]$  e quindi positiva e strettamente minore di 1.*

**Dimostrazione.** La dimostrazione è come la precedente, prendendo 0 invece di  $U$ .

Inoltre

**Proposizione 1.8.** *L'insieme dei numeri macchina  $F(\beta, t, L, U)$  ha un numero finito di elementi e la sua cardinalità è*

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

**Dimostrazione.** Per una fissata mantissa, a  $t$  cifre, il numero più piccolo che possiamo scrivere è

$$(10 \dots 00)_\beta$$

mentre il più grande è

$$\underbrace{(\beta - 1 \ \beta - 1 \dots \beta - 1 \ \beta - 1)}_t)_\beta.$$

Così non è difficile vedere che i numeri scrivibili con  $t$  cifre di mantissa sono quelli in cui la prima cifra va da 1 a  $\beta - 1$  e le altre da 0 a  $\beta - a$ , e quindi la loro cardinalità risulta

$$(\beta - 1) \cdot \underbrace{\beta \dots \beta}_{t-1} = (\beta - 1)\beta^{t-1}.$$

Quindi, al variare dell'esponente  $e$  in  $L, L + 1, \dots, U - 1, U$ , e del segno  $+$  o  $-$ , i numeri floating-point sono

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

**Proposizione 1.9.** *I numeri macchina sono numeri razionali. In particolare quelli che hanno esponente  $e \geq t$  sono interi.*

**Dimostrazione.** La scrittura dei numeri floating-point come somma di un numero finito di numeri frazionari implica che i numeri risultanti siano razionali.

Per vedere se  $e \geq t$  allora sono interi, da

$$x = \text{sgn}(x)\beta^e \sum_{i=1}^t d_i \beta^{-i} = \text{sgn}(x) \sum_{i=1}^t d_i \beta^{e-i}$$

visto che per  $i = 1, \dots, t \geq e$  che  $\beta^{e-i}$  è intero, come d'altra parte i  $d_i$ , lo è anche  $x$ .

**Proposizione 1.10.** *L'insieme dei numeri macchina  $F(\beta, t, L, U)$  non consiste di punti equispaziati.*

**Dimostrazione.** Per capirlo, il numero positivo più piccolo, e quindi più vicino a 0 è  $\beta^{1-L} \ll 1$  mentre i numeri floating-point che hanno esponente  $e \geq t$  sono interi e quindi la distanza di due tali numeri successivi è sicuramente maggiore o uguale 1. In definitiva, due numeri successivi vicino a 0 distano molto poco, e due numeri successivi con esponente molto grande distano molto, e quindi l'insieme dei floating-point non consiste di punti equispaziati.

**Esempio.** In Matlab, in precisione doppia, il numero successivo a 0 è circa  $2.2251 \cdot 10^{-308}$  mentre i due numeri più grandi distano l'uno dall'altro  $\approx 1.9958 \cdot 10^{+292}$ . Di conseguenza l'insieme numerico del Matlab non consiste di punti equispaziati perchè  $1.9958 \cdot 10^{292} \gg 2.2251 \cdot 10^{-308}$ .

**Proposizione 1.11.** *L'insieme dei numeri macchina  $F(\beta, t, L, U)$  non consiste di punti equispaziati.*

**Dimostrazione.** Per capirlo, il numero positivo più piccolo, e quindi più vicino a 0 è  $\beta^{1-L} \ll 1$  mentre i numeri floating-point che hanno esponente  $e \geq t$  sono interi e quindi la distanza di due tali numeri successivi è sicuramente maggiore o uguale 1. In definitiva, due numeri successivi vicino a 0 distano molto poco, e due numeri successivi con esponente molto grande distano molto, e quindi l'insieme dei floating-point non consiste di punti equispaziati.

**Nota. 1.12.** *Nel caso della notazione*

$$y = \text{sign}(y) \cdot (1.d_2 \dots d_t)_\beta \beta^e = \text{sign}(y) \cdot \beta^e \left(1 + \sum_{k=2}^t d_k \beta^{-k}\right) \quad (5)$$

abbiamo i seguenti risultati per  $F(\beta, t, L, U)$ , che si possono dimostrare con tecniche simili a quelle viste sopra:

- la somma degli elementi della mantissa  $(1, d_1 \dots d_{t-1})$  è un numero nell'intervallo  $[1, 2 - \beta^{-(t-1)}] \subset [1, 2)$ ;
- Il più piccolo numero positivo è  $\beta^L$  (usare quale mantissa  $(1.0 \dots 0)$ );
- Il più grande numero positivo è  $(2 - \beta^{-(t-1)}) \cdot \beta^U$ ;

Gli altri asserti sono uguali a quelli visti per la notazione classica.

Matlab utilizza questa notazione e quindi per quanto riguarda ad esempio la precisione doppia  $F(2, 53, -1022, 1023)$ :

- la somma degli elementi della mantissa  $(1, d_1 \dots d_{52})$  è un numero nell'intervallo  $[1, 2 - \beta^{-(52)}] \approx [1, 2 - 2.22 \cdot 10^{-16}]$ ;
- Il più piccolo numero positivo è  $\beta^L = 2^{-1022} \approx 2.225073858507201e - 308$ ;

- Il più grande numero positivo è  $(2 - \beta^{-(t-1)}) \cdot \beta^U = (2 - \beta^{-(52)}) \cdot 2^{1023} = 1.797693134862316e + 308$ .

Di seguito faremo esempi di queste quantità relativamente agli insiemi  $F(2, 24, -126, 127)$  e  $F(2, 53, -1022, 1023)$ .

**Nota. 1.13.** Per quanto riguarda numeri speciali

- per rappresentare  $+\text{inf}$  si utilizza il segno pari a 0, tutti gli esponenti uguali a 1 e tutta la mantissa uguale a 0;
- per rappresentare  $-\text{inf}$  si utilizza il segno pari a 1, tutti gli esponenti uguali a 1 e tutta la mantissa uguale a 0;
- per rappresentare lo 0 si distingue  $+0$  da  $-0$  (generalmente utilizzati per denotare un numero troppo piccolo in modulo che viene arrotondato a 0, pur mantenendo il segno); a parte il segno (positivo di default), tutte le altre cifre di esponente e mantissa sono uguali a 0;
- per rappresentare con NaN il risultato di un'operazione (numerica) eseguita su operandi non validi (specialmente in calcoli in virgola mobile). Secondo IEEE 754 i NaN sono rappresentati con il campo dell'esponentiale riempito di "1" e un numero diverso da zero nel campo della mantissa.

### 1.3. Precisione singola e doppia

Descriviamo di seguito i più comuni insiemi di numeri floating-point. Usualmente se  $M$  è il numero di bits necessari per un numero in precisione *singola*, nel caso di precisione *doppia* questo è usualmente composto da 2 parole di  $M$  bits.

Classici esempi sono per quanto riguarda i processori che seguono lo standard IEEE754 (inizialmente proposto da INTEL, studiato dal 1977 e introdotto nel 1985), e i successivi aggiornamenti fino all'attuale versione 2008 e alle proposte di IEEE854,

- per la precisione singola  $F(2, 24, -126, 127)$  in cui ogni numero macchina occupa 32 bit (o 4 byte essendo un byte pari a 8 bit)
- per la doppia  $F(2, 53, -1022, 1023)$  in cui ogni numero macchina occupa 64 bit.

**Nota. 1.14.** Ricordiamo che nella implementazione effettiva di IEEE754 ci sono molti dettagli tecnici che la rendono leggermente diversa da quanto analizzato, anche se essenzialmente i risultati sono gli stessi del nostro modello scelto.

### 1.3.1. Precisione singola

Nel caso della precisione singola  $F(2, 24, -126, 127)$ ,

- Si usa la base 2.
- Si usa un bit per il segno.
- Si usa una mantissa di 24 cifre binarie, ma visto che si utilizza la notazione normalizzata, solo 23 sono necessarie.
- Visto che si compone di 32 bits, di cui 24 per mantissa e segno, 8 possono essere dedicate all'esponente; quindi si possono rappresentare numeri interi da 0 a 255; in questo ambito i numeri 0 e 255 vengono utilizzati per significati speciali e quindi utilizzati esclusivamente gli interi  $N_e$  da 1 a 254; fissato  $\text{bias}=127 = 2^7 - 1$  per ottenere gli esponenti  $e$  li si codifica con  $N_e$  tale che

$$e = N_e - \text{bias}.$$

Evidentemente il piú piccolo esponente  $e_{\min} = 1 - 127 = -126$ , mentre il piú grande é  $e_{\max} = 254 - 127 = 127$ .

Nell'implementazione Matlab,

- Il minimo numero positivo in precisione singola é

$$x_{\min} = 1.1754944e - 38;$$

- Il massimo numero positivo in precisione singola é

$$x_{\max} = 3.4028235e + 38;$$

- la cardinalità dell'insieme  $F(2, 24, -126, 127)$  è

$$\approx 4.2950e + 09.$$

### 1.3.2. Precisione doppia

Nel caso della precisione doppia  $F(2, 53, -1022, 1023)$ ,

- Si usa la base 2.
- Si usa un bit per il segno.
- Si usa una mantissa di 53 cifre binarie, ma visto che si utilizza la notazione normalizzata, solo 52 sono necessarie.



- Visto che si compone di 64 bits, di cui 53 per mantissa e segno, 11 possono essere dedicate all'esponente; quindi si possono rappresentare numeri interi da 0 a 2047; in questo ambito i numeri 0 e 2047 vengono utilizzati per significati speciali e quindi utilizzati esclusivamente gli interi  $N_e$  da 1 a 2046; fissato  $\text{bias}=1023 = 2^{10} - 1$  per ottenere gli esponenti  $e$  li si codifica con  $N_e$  tale che

$$e = N_e - \text{bias}.$$

Evidentemente il più piccolo esponente  $e_{\min} = -(1 - 1023) = -1022$ , mentre il più grande é  $e_{\max} = 2046 - 1023 = 1023$ .

Nell'implementazione Matlab,

- Il minimo numero positivo in precisione doppia é

$$x_{\min} = 2.225073858507201e - 308;$$

- Il massimo numero positivo in precisione doppia é

$$x_{\max} = 1.797693134862316e + 308;$$

- la cardinalità dell'insieme  $F(2, 53, -1022, 1023)$  è  $\approx 3.7038e + 19$ .

**Nota. 1.15.** Si può credere che i numeri utilizzati per le precisioni singole e doppie siano sempre stati questi, ma non è così. In altri sistemi, differentemente da IEEE754, il numero di cifre per esponente e mantissa sono risultati molteplici, ad esempio su

- Zuse Z1 ( $\approx 1936$ ), la mantissa corrispondeva a 22 cifre (14 per la mantissa e 8 per l'esponente),
- IBM 3033 in precisione doppia si aveva, oltre alla base  $\beta = 16$ , una mantissa di 14 cifre,
- PRIME 850, che lavorava in base  $\beta = 2$ , si aveva in precisione singola 23 cifre di mantissa, mentre in precisione doppia 47 cifre di mantissa, [2, p.13].

**Nota. 1.16.** In generale la cardinalità dei numeri macchina è molto alta, spesso ordini di grandezza maggiore di quanto sia necessaria nelle scienze applicate [19].

*On a computer, the interval  $[1, 2]$ , for example, is approximated by about  $10^{16}$  numbers. It is interesting to compare the fineness of this discretization with that of the discretizations of physics. In a handful of solid or liquid or a balloonful of gas, the number of atoms or molecules in a line from one point to another is on the order of 108 (the cube root of Avogadro's number).*

*Such a system behaves enough like a continuum to justify our definitions of physical quantities such as density, pressure, stress, strain, and temperature. Computer arithmetic, however, is more than a million times finer than this.*

*Another comparison with physics concerns the precision to which fundamental constants are known, such as (roughly) 4 digits for the gravitational constant  $G$ , 7 digits for Planck's constant  $h$  and the elementary charge  $e$ , and 12 digits for the ratio  $\mu_e/\mu_B$  of the magnetic moment of the electron to the Bohr magneton. At present, almost nothing in physics is known to more than 12 or 13 digits of accuracy.*

*Thus IEEE numbers are orders of magnitude more precise than any number in science. (Of course, purely mathematical quantities like  $\pi$  are another matter.)*

### 1.3.3. Troncamento e arrotondamento

Se  $x$  è il numero reale

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

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

$$\text{fl}(x) = \text{sign}(x)(0.d_1, \dots, d_{t-1}, \overline{d}_t)\beta^e = \text{sign}(x)\beta^e \left( \sum_{k=1}^{t-1} d_k \beta^{-k} + \overline{d}_t \beta^{-t} \right)$$

in cui

- se si effettua il *troncamento*

$$\overline{d}_t = d_t;$$

- 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. L'ultima, è la più diffusa.

Curiosa la descrizione in [8, p.5]

A machine register acts much like the infamous Procrustes bed in Greek mythology. Procrustes was the innkeeper whose inn had only beds of one size. If a fellow came along who was too tall to fit into his beds, he cut off his feet. If the fellow was too short, he stretched him. In the same way, if a real number comes along that is too long, its tail end (not the head) is cutoff; if it is too short, it is padded by zeros at the end.

**Esempio.** Si consideri

$$\pi = (+1) \cdot (0.31415926535 \dots)_{10} \cdot 10^1.$$

Se lo tronchiamo alla quarta cifra decimale otteniamo

$$fl(\pi) = +1 \cdot (0.3141)_{10} \cdot 10^1,$$

mentre se lo arrotondiamo

$$fl(\pi) = +1 \cdot (0.3142)_{10} \cdot 10^1.$$

Evidentemente, per effettuare un arrotondamento, il calcolatore deve immagazzinare in un extra-bit la cifra  $d_{t+1}$  (rifletterci un po').

Se l'esponente  $e$  del numero  $x$  è

- minore di  $L$  si commette un'errore di *underflow*,
- 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.

#### 1.4. La precisione di macchina

Una delle costanti rilevanti all'interno del sistema floating point è la precisione di macchina.

**Definizione.** Si dice *precisione di macchina*, quel numero  $\epsilon_{ps}$  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  $d_1 = 1, \dots, d_{t-1} = 0, d_t = 1$  e esponente  $e = 1$  e quindi è  $x = 1 + \beta^1 \cdot \beta^{-t} = 1 + \beta^{1-t}$ , per cui

$$\epsilon_{ps} = \beta^{1-t}.$$

Si sottolinea che la precisione di macchina  $\epsilon_{ps}$  non coincide in generale con il più piccolo numero positivo rappresentabile dal calcolatore, in quanto se  $t \neq -L$ , come usualmente accade, allora

$$\beta^{1-t} \neq \beta^{L-1}.$$

In Matlab ad esempio,  $x_{min}$  è molto inferiore di  $\epsilon_{ps}$ , in quanto

- il più piccolo numero positivo in modulo è

$$x_{min} = 2^{-1022} = 2.225073858507201 \cdot 10^{-308},$$

- la precisione di macchina è

$$\text{eps} = 2^{-52} = 2.220446049250313 \cdot 10^{-16}.$$

**Nota. 1.17.** Usualmente

- un numero macchina il cui esponente è più piccolo di  $L$  produce il cosiddetto *underflow*,
- un numero macchina il cui esponente è più grande di  $U$  produce il cosiddetto *overflow*.

Osserviamo che esistono numeri denormalizzati, il cui range estende quello dei normalizzati. Il più piccolo numero positivo denormalizzato è  $0.494 \cdot 10^{-323}$ , permettendo di passare gradualmente dall'*underflow* allo 0.

### 1.5. Errori relativi e assoluti

Fissato un numero da approssimare

$$x^* \in \mathbb{R},$$

si definisce

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

$$|x - x^*|$$

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

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

Si noti che se  $|x^*| = 0$ , cioè  $x^* = 0$ , allora non ha senso la scrittura propria dell'errore relativo.

**Nota. 1.18.** Fissato un vettore da approssimare

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

si definisce

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

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

dove

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

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

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

Si noti che se  $\|x^*\| = 0$ , cioè  $x^* = 0$  per la proprietà delle norme [2, 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  $|\cdot|$ .  $\square$

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

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

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

$$\begin{aligned} \frac{|x - fl(x)|}{|x|} &\leq \frac{|\text{sign}(x)\beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k} - (\text{sign}(x)\beta^e (\sum_{k=1}^{t-1} d_k \beta^{-k} + \overline{d}_t \beta^{-t}))|}{|\beta^{e-1}|} \\ &= \beta \cdot \left| \sum_{k=t}^{+\infty} d_k \beta^{-k} - \overline{d}_t \beta^{-t} \right| \end{aligned} \quad (6)$$

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

$$\beta \cdot \left| \sum_{k=t}^{+\infty} d_k \beta^{-k} - \overline{d}_t \beta^{-t} \right| = \beta \cdot \left| \sum_{k=t+1}^{+\infty} d_k \beta^{-k} \right| < \beta \cdot (\beta \cdot \beta^{-t-1}) = \beta^{1-t} = \text{eps}$$

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

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

La quantità  $\frac{\epsilon_{ps}}{2}$  è detta *unità di arrotondamento*.

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

**Nota. 1.19.** *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 (se così fosse, visto che i numeri più grandi sono interi, pure  $\epsilon_{ps}$  lo sarebbe).*

## 2. Operazioni con i numeri macchina

Per quanto concerne le quattro operazioni aritmetiche, si può dimostrare che la divisione e la moltiplicazione di tipo floating-point producono un errore relativo piccolo rispetto al risultato esatto, cosa che non succede sempre per somma e sottrazione.

Più precisamente se  $fl(x)$  è il numero macchina che *corrisponde* a  $x$ , denotiamo con

$$x \oplus y := fl(fl(x) + fl(y)) \quad (7)$$

$$x \ominus y := fl(fl(x) - fl(y)) \quad (8)$$

$$x \otimes y := fl(fl(x) \cdot fl(y)) \quad (9)$$

$$x \oslash y := fl(fl(x) : fl(y)) \quad (10)$$

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

$$\epsilon_x := \frac{|x - fl(x)|}{|x|} \quad (11)$$

$$\epsilon_{x,y}^{\odot} := \frac{|(x \text{ op } y) - (x \odot y)|}{|x \text{ op } y|} \quad (12)$$

**Nota. 2.1.** *Osserviamo, che a volte nei manuali si trova qualcosa di diverso. Ad esempio,*

$$x \oplus y = fl(x + y).$$

*In realtà il sistema floating point, utilizzando ulteriori bits nei calcoli, garantisce che sia*

$$x \oplus y = fl(fl(x) + fl(y)) = fl(x + y).$$

*In effetti, da [9]*

*The IEEE standard requires that the result of addition, subtraction, multiplication and division be exactly rounded. That is, the result must be computed exactly and then rounded to the nearest floating-point number (using round to even).*

*The section Guard Digits pointed out that computing the exact difference or sum of two floating-point numbers can be very expensive when their exponents are substantially different. That section introduced guard digits, which provide a practical way of computing differences while guaranteeing that the relative error is small.*

*However, computing with a single guard digit will not always give the same answer as computing the exact result and then rounding. By introducing a second guard digit and a third sticky bit, differences can be computed at only a little more cost than with a single guard digit, but the result is the same as if the difference were computed exactly and then rounded. Thus the standard can be implemented efficiently.*

Altre volte si trova equivalentemente [19]

*If  $*$  is one of these four operations in its ideal form and  $\odot$  is the same operation as realized on the computer, then for any floating-point numbers  $x$  and  $y$ , assuming that there is no underflow or overflow,*

$$x \odot y = (x * y)(1 + \epsilon).$$

*Here  $\epsilon$  is a very small quantity, no greater in absolute value than a number known as machine epsilon, denoted by  $\epsilon_{mach}$ , that measures the accuracy of the computer. In the IEEE system,  $\epsilon_{mach} = 2^{-53} \approx 1.1 \times 10^{-16}$ .*

*I calcoli che faremo successivamente per studiare le quattro operazioni in floating-point, possono essere eseguiti anche in questo modo.*

## 2.1. Proprietà commutativa, associativa e distributiva delle operazioni floating point

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.

Quindi l'ordine in cui sono eseguite le operazioni può variare il risultato.

Inoltre esistono l'elemento neutro della moltiplicazione, divisione, addizione e sottrazione, come pure l'opposto di ogni numero floating-point.

**Esempio.** 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:

$$\begin{aligned}(a \oplus b) \oplus c &= 0.2002 \cdot 10^4 \oplus 0.7800 \cdot 10^1 = 0.2009 \cdot 10^4 \\ a \oplus (b \oplus c) &= 0.2000 \cdot 10^4 \oplus 0.1030 \cdot 10^2 = 0.2010 \cdot 10^4\end{aligned}$$

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  $a = 0.800000 \cdot 10^9$ ,  $b = 0.500009 \cdot 10^4$ ,  $c = 0.500008 \cdot 10^4$

$$\begin{aligned} a \otimes (b \ominus c) &= 0.800000 \cdot 10^9 \otimes 0.1 \cdot 10^{-2} = 0.800000 \cdot 10^9 \\ (a \otimes b) \ominus (a \otimes c) &= (0.800000 \cdot 10^9 \otimes 0.500009 \cdot 10^4) \ominus \\ &\quad \ominus (0.800000 \cdot 10^9 \otimes 0.500008 \cdot 10^4) \rightarrow \text{overflow(13)} \end{aligned}$$

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

## 2.2. Errori nelle operazioni e loro propagazione

Consideriamo ora come si propagano gli errori in  $\oplus$ ,  $\ominus$ ,  $\otimes$ ,  $\oslash$ . Si mostra che (cf. [5, p.78])

$$\epsilon_{x,y}^{\oplus} \leq \left| \frac{x}{x+y} \right| \epsilon_x + \left| \frac{y}{x+y} \right| \epsilon_y, \quad x+y \neq 0 \quad (14)$$

$$\epsilon_{x,y}^{\ominus} \leq \left| \frac{x}{x-y} \right| \epsilon_x + \left| \frac{y}{x-y} \right| \epsilon_y, \quad x-y \neq 0 \quad (15)$$

$$\epsilon_{x,y}^{\otimes} \lesssim \epsilon_x + \epsilon_y \quad (16)$$

$$\epsilon_{x,y}^{\oslash} \leq |\epsilon_x - \epsilon_y| \quad (17)$$

dove per ogni operazione floating point  $\odot$  relativamente all'operazione  $\circ p$  (somma, sottrazione, moltiplicazione o divisione)

$$\begin{aligned} \epsilon_x &= |x - fl(x)|/|x|, \\ \epsilon_y &= |y - fl(y)|/|y|, \\ \epsilon_{x,y}^{\odot} &= |fl(x \circ p y) - (x \circ p y)|/|x \circ p y| \end{aligned}$$

Proviamo di seguito sotto opportune ipotesi il caso della somma, sottrazione e della moltiplicazione, lasciando al lettore quello della divisione. Con  $a \lesssim b$  si intende  $a$  inferiore o uguale circa a  $b$ .

**Proposizione 2.2.** *Se,*

- $x + y \neq 0$ ,  $x \neq 0$ ,  $y \neq 0$ ,
- $fl(fl(x) + fl(y)) = fl(x) + fl(y)$ ,
- $\epsilon_x = |x - fl(x)|/|x|$ ,
- $\epsilon_y = |y - fl(y)|/|y|$ ,
- $\epsilon_{x,y}^{\oplus} = |fl(x \oplus y) - (x + y)|/|x + y|$

*allora*

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



**Dimostrazione.** Nelle ipotesi precedenti abbiamo

$$\begin{aligned}\epsilon_{x,y}^{\oplus} &= \frac{|(x+y) - (x \oplus y)|}{|x+y|} = \frac{|(x+y) - \text{fl}(\text{fl}(x) + \text{fl}(y))|}{|x+y|} = \frac{|(x - \text{fl}(x)) + (y - \text{fl}(y))|}{|x+y|} \\ &\leq \frac{|x|}{|x+y|} \cdot \frac{|x - \text{fl}(x)|}{|x|} + \frac{|y|}{|x+y|} \cdot \frac{|y - \text{fl}(y)|}{|y|} = \left| \frac{x}{x+y} \right| \epsilon_x + \left| \frac{y}{x+y} \right| \epsilon_y.\end{aligned}\quad (18)$$

Il punto chiave è che per  $x+y \approx 0$  abbiamo che per piccoli errori sui dati  $\epsilon_x, \epsilon_y$  possono aversi errori rilevanti in  $\epsilon_{x,y}^{\oplus}$  (tale fenomeno è noto come cancellazione).

**Proposizione 2.3.** Se,

- $x - y \neq 0, x \neq 0, y \neq 0$ ,
- $\text{fl}(\text{fl}(x) - \text{fl}(y)) = \text{fl}(x) - \text{fl}(y)$
- $\epsilon_x = |x - \text{fl}(x)|/|x|$ ,
- $\epsilon_y = |y - \text{fl}(y)|/|y|$ ,
- $\epsilon_{x,y}^{\ominus} = |\text{fl}(x \ominus y) - (x+y)|/|x+y|$

allora

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

**Dimostrazione.** Nelle ipotesi precedenti abbiamo

$$\begin{aligned}\epsilon_{x,y}^{\ominus} &= \frac{|(x-y) - (x \ominus y)|}{|x-y|} = \frac{|(x-y) - \text{fl}(\text{fl}(x) - \text{fl}(y))|}{|x-y|} = \frac{|(x - \text{fl}(x)) - (y - \text{fl}(y))|}{|x-y|} \\ &\leq \frac{|x|}{|x-y|} \cdot \frac{|x - \text{fl}(x)|}{|x|} + \frac{|y|}{|x-y|} \cdot \frac{|y - \text{fl}(y)|}{|y|} = \left| \frac{x}{x-y} \right| \epsilon_x + \left| \frac{y}{x-y} \right| \epsilon_y\end{aligned}\quad (19)$$

Nuovamente, per  $x-y \approx 0$  abbiamo che per piccoli errori sui dati  $\epsilon_x, \epsilon_y$  possono aversi errori rilevanti in  $\epsilon_{x,y}^{\ominus}$ . Qualora ciò accada, si parla nuovamente di *fenomeno di cancellazione*.

**Cancellazione** Consideriamo il sistema  $F(10, 5, -5, +5)$  (con arrotondamento) e i due numeri

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

Naturalmente,

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

e quindi

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

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!).

**Proposizione 2.4.** *Supposti,*

- $x \cdot y \neq 0, x \neq 0, y \neq 0, |fl(y)|/y \approx 1,$
- $fl(fl(x) \cdot fl(y)) = fl(x) \cdot fl(y)$  (cioè  $fl(x) \cdot fl(y)$  numero macchina),

posto  $\epsilon_x = \frac{|x - fl(x)|}{|x|}$  ed  $\epsilon_y = \frac{|y - fl(y)|}{|y|}$ ,  $\epsilon_{x,y}^\otimes = |x \otimes y - x \cdot y|/|x \cdot y|$  allora

$$\epsilon_{x,y}^\otimes \lesssim \epsilon_x + \epsilon_y.$$

**Dimostrazione.** Nelle ipotesi precedenti abbiamo

$$\begin{aligned} \epsilon_{x,y}^\otimes &= \frac{|x \cdot y - x \otimes y|}{|x \cdot y|} = \frac{|x \cdot y - fl(fl(x) \cdot fl(y))|}{|x \cdot y|} = \frac{|x \cdot y - fl(x) \cdot fl(y)|}{|x \cdot y|} \\ &= \frac{|x \cdot y - x \cdot fl(y) + x \cdot fl(y) - fl(x) \cdot fl(y)|}{|x \cdot y|} \\ &\leq \frac{|x \cdot (y - fl(y))| + |fl(y) \cdot (x - fl(x))|}{|x \cdot y|} \\ &= \frac{|y - fl(y)|}{|y|} + \frac{|fl(y) \cdot (x - fl(x))|}{|x \cdot y|} \\ &\approx \epsilon_x + \epsilon_y \end{aligned} \tag{20}$$

dove si ricorda che

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

Per quanto riguarda la divisione, forniamo una traccia: basta studiare il caso  $1/y$  e poi ricordare i risultati del prodotto.

**Nota. 2.5.** *Per stabilità di una generica operazione macchina op (si pensi ad esempio \* come alla somma oppure al prodotto), si intende che se  $\odot$  è l'omologa versione floating-point allora esiste  $C > 0$  tale che*

$$\epsilon_{x,y}^\odot \leq C(\epsilon_y \text{ op } \epsilon_y)$$

per ogni  $x, y$  per cui la scrittura abbia significato.

Ne consegue che la somma e la sottrazione non sono stabili mentre lo sono il prodotto e la divisione [14, p.10].

**Nota. 2.6 (Difficile, necessario aver seguito il corso di Analisi II).** Osserviamo che dalla formula di Taylor in due variabili, posto per brevità di notazione  $f'_x = \partial f / \partial x$ ,  $f'_y = \partial f / \partial y$ ,

$$f(x, y) \approx f(\bar{x}, \bar{y}) + f'_x(\bar{x}, \bar{y})(x - \bar{x}) + f'_y(\bar{x}, \bar{y})(y - \bar{y}) \tag{21}$$

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

$$\begin{aligned} \frac{|f(x, y) - f(\bar{x}, \bar{y})|}{|f(x, y)|} &\lesssim \frac{|f'_x(\bar{x}, \bar{y})||x - \bar{x}||x|}{|f(x, y)||x|} + \frac{|f'_y(\bar{x}, \bar{y})||y - \bar{y}||y|}{|f(x, y)||y|} \\ &= \frac{|f'_x(\bar{x}, \bar{y})||x|}{|f(x, y)|} \cdot \epsilon_x + \frac{|f'_y(\bar{x}, \bar{y})||y|}{|f(x, y)|} \cdot \epsilon_y \end{aligned} \quad (22)$$

Da (22), si possono ottenere risultati simili ai precedenti per  $\oplus$ ,  $\otimes$ . Uno dei vantaggi è che si possono studiare operazioni più generali come ad esempio cosa succeda se si esegue  $\exp(x + y)$ .

### 3. Alcune problematiche numeriche

Si consideri un problema scientifico che abbia a che fare con una procedura di calcolo. Sussistono varie tipologie di errore nei calcoli:

- errori di modellizzazione: vengono utilizzate equazioni per rappresentare un evento fisico, e il modello matematico è solo una semplificazione di quello reale;
- errori di tabulazione: vengono immessi dal programmatore o dal programma dati errati (cosa che ad esempio era comune una volta quando bisognava tabulare manualmente numeri con molte cifre); un errore di questo tipo ha causato nel 1996 la perdita del razzo Ariane 5 per una cattiva conversione di un numero;

Il 4 giugno 1996, è avvenuta un'esplosione a soli 40 secondi dal lancio dovuta a un overflow per una conversione di un numero reale memorizzato usando 64 bit in un numero intero con soli 16 bit. Il numero che rappresentava la velocità orizzontale del razzo era più grande del massimo numero rappresentabile con soli 16 bit. La missione era costata 7.5 miliardi di dollari.

Errori di questo tipo hanno causato pure il crollo di una piattaforma petrolifera nel mare del Nord (Norvegia) nel 1991 e la distruzione del veicolo spaziale Mars Climate orbiter nel 1999 (cf. [1], [11]).

- errori dovuti a misure fisiche: sono dovuti a una imprecisa osservazione sperimentale, spesso dovuta agli strumenti utilizzati, ad esempio la velocità della luce nel vuoto è

$$c = (2.997925 + \epsilon) \cdot 10^{10} \text{ cm/s}, \quad |\epsilon| \leq 0.000003$$

e quindi nei calcoli che hanno a che vedere con  $c$  necessariamente si compie un errore;

- errori di rappresentazione e di aritmetica di macchina: sono errori dovuti all'arrotondamento o troncamento di un numero, e sono inevitabili quando si usa l'aritmetica floating-point; sono la sorgente principale di errore di alcuni problemi come ad esempio la soluzione di sistemi lineari.

Di seguito vediamo alcune problematiche dovute agli errori di rappresentazione e di aritmetica di macchina. Per una buona serie di esempi si veda [3, p.46].

#### 4. Valutazione di una funzione

Nel valutare una funzione continua  $f : \Omega \subseteq \mathbb{R} \rightarrow \mathbb{R}$  in un punto  $x \in \Omega$  bisogna tener conto che il dato  $x$  può essere per varie ragioni affetto da errori, tipicamente di misura o di rappresentazione al calcolatore, così da essere approssimato con un certo  $\bar{x}$ . Di conseguenza, invece di  $f(x)$  si valuta  $f(\bar{x})$  e ci si domanda se ci siano considerevoli differenze.

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

$$\frac{|x - x_c|}{|x|}$$

corrispondano grandi errori (relativi) sui risultati

$$\frac{|f(x) - f(x_c)|}{|f(x)|}.$$

Quindi è importante discutere la quantità

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

La valutazione di  $f$  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 abbiamo

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

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  $x$ .

**Esempio.** Data la funzione

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

calcoliamo analiticamente il *condizionamento*

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

Ricordiamo che

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

da cui per  $x \in (-1, 1)$

$$\mathcal{K}(f_1, x) = \frac{|x \cdot f_1'(x)|}{|f_1(x)|} = \frac{|x \cdot \frac{1}{2\sqrt{1-x^2}}(-2x)|}{|1 - \sqrt{1-x^2}|} = \frac{x^2}{|1 - \sqrt{1-x^2}|\sqrt{1-x^2}}.$$

Dal grafico di tale funzione nella Figura 4, si vede che la funzione  $f_1$  è ben condizionata in  $(-0.9, 0.9)$ , ma non lo è per valori prossimi a 1.

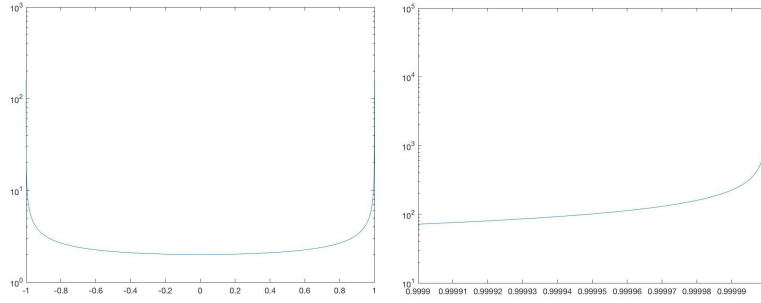


Figura 1: A sinistra il grafico di  $\mathcal{K}(f_1, x)$  in  $(-1, 1)$ , a destra quello di  $\mathcal{K}(f_1, x)$  in  $(1 - 10^{-4}, 1)$  (malcondizionamento per  $|x| \approx 1$ ).

**Esempio.** Consideriamo quale esempio

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

Non è difficile vedere che  $f_2'(x) = -1$  e quindi

$$\mathcal{K}(f_2, x) = \frac{|x \cdot f_2'(x)|}{|1 - x|} = \frac{|x \cdot (-1)|}{|1 - x|} = \frac{|x \cdot (-1)|}{|1 - x|}.$$

Dal grafico di tale funzione nella Figura 4, si vede che la funzione  $f_2$  è ben condizionata ovunque eccetto per valori prossimi a 1. Si osservi che per vicini a tale valore, in effetti, si ha il fenomeno di cancellazione.

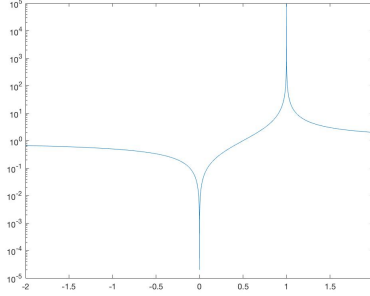


Figura 2: Il grafico di  $\mathcal{K}(f_2, x)$  in  $(-2, 2)$  (malcondizionamento per  $x \approx 1$ ).

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

$$y = -p + \sqrt{p^2 + q}. \quad (23)$$

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

$$y = -p \pm \sqrt{p^2 + q}. \quad (24)$$

dell'equazione sono reali. La soluzione descritta in (23) è la maggiore delle 2, ed è non negativa se e solo se  $q \geq 0$ . Si osserva subito che (23) 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 (23). In altri termini

$$y = -p + \sqrt{p^2 + q} = \frac{(-p + \sqrt{p^2 + q})(p + \sqrt{p^2 + q})}{(p + \sqrt{p^2 + q})} = \frac{q}{(p + \sqrt{p^2 + q})} \quad (25)$$

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 [5, p. 66].

Seguendo [5, p. 10], [5, 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. [17], p. 21, [6], p. 11)

1. il primo algoritmo (24) non è *numericamente stabile* qualora  $p \gg q > 0$ ;
2. il secondo algoritmo (25) è *numericamente stabile* qualora  $p \gg q > 0$ .

Vediamo alcuni casi rilevanti, effettuati in doppia precisione.

- In [17, p.22], si suggerisce un test interessante per

$$p = 1000, q = 0.018000000081$$

la cui soluzione esatta è  $0.9 \cdot 10^{-5}$ . Si ha  $p \gg q$ . Lavorando in doppia precisione otteniamo

algoritmo	valore	err. rel.
1	0.0000089999999772772	$2.52 \cdot 10^{-9}$
2	0.0000090000000000000	0

- Secondo [6, p.11], è notevole l'esperimento in cui

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

avente soluzione esatta  $10^{-7}$ . Nuovamente  $p \gg q$ .

algoritmo	valore	err. rel.
1	0.0000001000007614493	$7.61 \cdot 10^{-6}$
2	0.0000001000000000000	0

## 5.2. Esempio 2: Approssimazione di $\pi$

Storicamente sono state scoperte diverse successioni convergenti sempre più rapidamente a  $\pi$  (cf. [27]). In questa sezione ci interesseremo a tre di queste mostrando sia come le velocità di convergenza possano essere diverse, sia come possano insorgere questioni di instabilità [7, p. 16].

Studiamo 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

$$\pi = 3,14159265358979323846264338327950288419716939 \dots$$

Implementiamo di seguito una terza successione, diciamo  $\{y_n\}$ , che si ottiene *razionalizzando*  $z_n$ , cioè moltiplicando numeratore e denominatore per

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

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

Si ottengono i seguenti risultati, che tabuliamo per semplicità esclusivamente per  $n = 5, 10, 15, 20, 25, 30, 35, 40$ .

$m$	$u_n$	$z_n$	$y_n$
5	2.963387701038571	3.121445152258050	3.121445152258053
10	3.049361635982070	3.141572940365566	3.141572940367093
15	3.079389826032086	3.141592633463248	3.141592634338565
20	3.094669524113704	3.141594125195191	3.141592653570997
25	3.103923391700576	3.142451272494133	3.141592653589779
30	3.110128728141262	4.000000000000000	3.141592653589798
35	3.114578862293132	0.000000000000000	3.141592653589798
40	3.117926198299378	0.000000000000000	3.141592653589798

Dalla figura e dalla tabella, consegue che

1. la convergenza della prima successione è estremamente lenta;



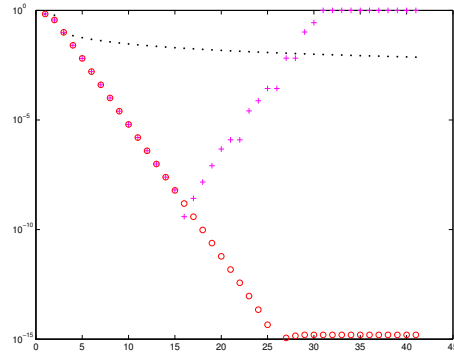


Figura 3: Le tre successioni  $u_n$  (puntino nero),  $z_n$  (punto magenta),  $y_n$  (cerchietto rosso). Risulta evidente che  $z_n$  diverge, nonostante fino a  $n \approx 14$  dia circa gli stessi valori di  $y_n$  che invece converge.

2. la seconda successione non convergente numericamente, seppure converga teoricamente; il principale problema è la cancellazione che si ha nell'argomento della radice quadrata più esterna di

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

A tal proposito, nella tabella che segue, evidenziamo i valori assunti da  $\sqrt{1 - 4^{1-n} \cdot z_n^2}$ , per  $n = 5, 10, 15, \dots, 40$ .

$n$	$\sqrt{1 - 4^{1-n} \cdot z_n^2}$
5	9.8078528040323054160154470e - 01
10	9.9998117528260110908888691e - 01
15	9.9999998161642922323011362e - 01
20	9.999999998204724960260137e - 01
25	9.99999999998245847621092e - 01
30	1.00000000000000000000000000e + 00
35	1.00000000000000000000000000e + 00
40	1.00000000000000000000000000e + 00

3. la terza successione risulta rapidamente convergente.

### 5.3. 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 [15, Esercizio 1.9, p.35]. Problemi simili con una precisa discussione della propagazione dell'errore sono trattati pure in [7, p. 23, problema 11]

Sia

$$I_n = e^{-1} \int_0^1 x^n e^x dx \quad (26)$$

**Proposizione 5.1.** *La successione  $I_n$  è positiva, decrescente e infinitesima (cioè  $\lim_n I_n = 0$ ). Inoltre*

$$I_{n+1} = 1 - (n+1) I_n, \quad I_0 = 1 - e^{-1}.$$

**Facoltativa** Per  $n = 0$  si ha

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

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. [30]) si ha che  $I_1 = e^{-1}$ , per cui effettivamente vale  $I_{n+1} = 1 - (n+1) I_n$  con  $n = 0$ , visto che si afferma

$$I_1 = 1 - I_0 = 1 - (1 - e^{-1}) = e^{-1}.$$

Più in generale integrando per parti che

$$I_{n+1} = e^{-1} \left( x^{n+1} e^x \Big|_0^1 - (n+1) \int_0^1 x^n e^x dx \right) = 1 - (n+1) I_n. \quad (27)$$

Da (26) essendo l'integranda  $x^n \exp x > 0$  per  $x \in (0, 1)$  si ha

$$I_n > 0.$$

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

$$I_{n+1} < I_n.$$

La successione  $I_n$  è positiva e decrescente e quindi ammette limite  $L$  finito. Da (27), calcolando il limite  $L$  per  $n \rightarrow \infty$  ad ambo i membri si ottiene portando 1 a primo membro

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

L'unica possibilità affinché  $\lim_n (n+1) I_n$  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. Come primo tentativo calcoliamo  $I_n$  per  $n = 1, \dots, 100$  mediante la successione *in avanti*

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

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

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

Si osservi che per raggiungere tale obiettivo bisogna calcolare i termini

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

Notiamo che tale 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

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

Dopo aver eseguito i calcoli otteniamo per  $n = 10, 20, \dots, 100$  la seguente tabella

$n$	$\sqrt{1 - 4^{1-n} \cdot z_n^2}$	
10	$8.387707005829270e - 02$	$8.387707010339417e - 02$
20	$-3.019239488558378e + 01$	$4.554488407581805e - 02$
30	$-3.296762455608386e + 15$	$3.127967393216808e - 02$
40	$-1.014081007335147e + 31$	$2.382272866903348e - 02$
50	$-3.780092725853228e + 47$	$1.923775443433938e - 02$
60	$-1.034194991387092e + 65$	$1.613316466384628e - 02$
70	$-1.488787166378969e + 83$	$1.389153285400528e - 02$
80	$-8.895191521232202e + 101$	$1.219691454831806e - 02$
90	$-1.846559775027300e + 121$	$1.087083605943215e - 02$
100	$-1.159928542966359e + 141$	$9.804855005376052e - 03$

La tabella mostra che

1.  $s_n$  non converge numericamente seppure lo faccia teoricamente,
2.  $t_n$  converge numericamente e teoricamente, qualora per calcolare  $t_n$ ,  $n < N$  si parta da un valore  $m \gg N$ . Nel nostro esempio, per  $N = 100$ , siamo partiti da  $m = 500$ .

La motivazione di tale comportamento consiste nel fatto che la prima successione, cioè quella in avanti, amplifica enormemente l'errore in avanti mentre la seconda, detta all'indietro, fa esattamente il contrario.

**Nota. 5.2.** In effetti anche se operassimo in aritmetica esatta, per quanto concerne la successione in avanti  $I_n$ , se  $I_0^* = I_0 + \epsilon$  allora la successione calcolata

$$I_{n+1}^* = 1 - (n+1)I_n^*$$

è tale che  $I_n^* - I_n = n!\epsilon$ , giustificando i risultati ottenuti.

Osserviamo infatti che ragionando per induzione,

- l'asserto  $I_n^* - I_n = n!\epsilon$  vale per  $n = 0$ , in quanto  $I_0^* = I_0 + 0!\epsilon = I_0 + \epsilon$ ;
- se supponiamo  $I_n^* - I_n = n!\epsilon$ , ovvero  $I_n^* = I_n - n!\epsilon$  allora

$$\begin{aligned} I_{n+1}^* &= 1 - (n+1)I_n^* = 1 - (n+1)(I_n - n!\epsilon) \\ &= 1 - (n+1)I_n + (n+1) \cdot (n!\epsilon) \\ &= I_n^* + (n+1)!\epsilon. \end{aligned} \tag{28}$$

#### 5.4. Esempio 4: sulla somma $\frac{(1+\eta)-1}{\eta}$

Calcoliamo l'errore relativo tra 1 e il valore che si ottiene valutando in doppia precisione

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

con  $\eta = 10^{-1}, 10^{-2}, \dots, 10^{-15}$ . Ovviamente il risultato esatto di questa divisione è sempre 1, per qualsiasi  $\eta$ .

Si tratta di una funzione ben condizionata visto che  $f(x) = 1$ , che però può essere valutata in forme diverse, più o meno favorevoli (come nel nostro caso). Risulta quindi importante capire che nell'esempio in questione, il fatto che avremo risultati inaspettati, diversamente da quanto visto per funzioni  $f$  malcondizionate, sarà dovuto all'algoritmo di calcolo.

Notiamo che il calcolatore effettuerà invece  $f^*(\eta)$  ottenuta

1. valutando  $a_1 = fl(fl(1) + fl(\eta))$ ;
2. valutando  $a_2 = fl(fl(a_1) - fl(1))$ ;
3. valutando  $a_3 = fl(fl(a_2) : fl(\eta))$ .

in cui ovviamente  $fl(1) = 1$ ,  $fl(a_1) = a_1$ ,  $fl(a_2) = a_2$  visto che  $a_1, a_2$  sono numeri floating point.

I risultati ottenuti nel valutare  $f^*(\eta)$  sono potenzialmente diversi da  $f(\eta) = 1$  (ma non per gli interi, perchè?).

Ricordando la notazione esponenziale in cui ad esempio  $3.1e-02=3.1 \cdot 10^{-2}$ , e che l'errore relativo risulta

$$\frac{|f(\eta) - f^*(\eta)|}{|f(\eta)|} = \frac{|1 - f^*(\eta)|}{1} = |1 - f^*(\eta)|$$

abbiamo

$\eta$	$f^*(\eta)$	rel.err.
$1.00e-01$	$1.000000000000001e+00$	$8.881784e-16$
$1.00e-02$	$1.000000000000001e+00$	$8.881784e-16$
$1.00e-03$	$9.99999999998899e-01$	$1.101341e-13$
$1.00e-04$	$9.99999999998899e-01$	$1.101341e-13$
$1.00e-05$	$1.00000000006551e+00$	$6.551204e-12$
$1.00e-06$	$9.99999999177334e-01$	$8.226664e-11$
$1.00e-07$	$1.000000000583867e+00$	$5.838672e-10$
$1.00e-08$	$9.99999939225290e-01$	$6.077471e-09$
$1.00e-09$	$1.000000082740371e+00$	$8.274037e-08$
$1.00e-10$	$1.000000082740371e+00$	$8.274037e-08$
$1.00e-11$	$1.000000082740371e+00$	$8.274037e-08$
$1.00e-12$	$1.000088900582341e+00$	$8.890058e-05$
$1.00e-13$	$9.992007221626409e-01$	$7.992778e-04$
$1.00e-14$	$9.992007221626409e-01$	$7.992778e-04$
$1.00e-15$	$1.110223024625157e+00$	$1.102230e-01$

La perdita di precisione è dovuta essenzialmente al fenomeno di cancellazione, visto che per  $\eta \approx 0$  si ha  $1 + \eta \approx 1$ . Ricordato che `eps` è un numero macchina e che per numeri più piccoli  $\eta$  in modulo di `eps` si ha  $1 + x = 1$  non sorprende inoltre sia

$\eta$	$f^*(\eta)$	rel.err.
<code>eps</code>	$1.000000000000000e+00$	$0.000000e+00$
<code>eps/2</code>	$0.000000000000000e+00$	$1.000000e+00$
<code>eps/3</code>	$0.000000000000000e+00$	$1.000000e+00$

In effetti, per questi ultimi valori  $\eta$  più piccoli in modulo di `eps`, visto che comunque  $fl(\eta) > 0$  (pensarci su!),

$$\frac{(1 + fl(\eta)) - 1}{fl(\eta)} = \frac{1 - 1}{fl(\eta)} = 0.$$

##### 5.5. Esempio 5: Sulla valutazione di $\tan(\arctan(x)) = x$

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

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

Ci domandiamo se l'implementazione numerica di questa funzione goda ancora di questa proprietà.

Valutiamo la funzione  $F$  numericamente, mediante la corrispettiva versione numerica  $F^*$  per

$$x = 10^k, \text{ per } k = -20 + 40h, \text{ e } h = 0, 0.1, 0.2, \dots, 1$$

nonchè l'errore relativo compiuto, ovvero

$$\text{rel.err.} := |F(x) - F^*(x)|/|F(x)|,$$

notando che  $F(x) \neq 0$  visto che nei casi in questione si ha  $x \neq 0$ . Descriviamo questi risultati nella tabella che segue.

$x$	$F^*(x)$	rel.err.
$1.0e - 20$	$1.0000000000000000e - 20$	$0.00000e + 00$
$1.0e - 16$	$1.0000000000000000e - 16$	$0.00000e + 00$
$1.0e - 12$	$1.0000000000000000e - 12$	$0.00000e + 00$
$1.0e - 08$	$1.0000000000000004e - 08$	$0.00000e + 00$
$1.0e - 04$	$1.0000000000000000e - 04$	$0.00000e + 00$
$1.0e + 00$	$9.999999999999999e - 01$	$1.11022e - 16$
$1.0e + 04$	$9.999999999990539e + 03$	$9.46056e - 13$
$1.0e + 08$	$9.99999999542369e + 07$	$4.57631e - 11$
$1.0e + 12$	$1.000071916854289e + 12$	$7.19117e - 05$
$1.0e + 16$	$1.633123935319537e + 16$	$3.87677e - 01$
$1.0e + 20$	$1.633123935319537e + 16$	$6.12223e + 03$

Osserviamo che i valori critici sono tra 1 e  $10^4$ . In effetti un'ulteriore analisi propone

$x$	$F^*(x)$	rel.err.
$1.0000000000000000e + 00$	$9.999999999999999e - 01$	$1.11022e - 16$
$2.511886431509580e + 00$	$2.511886431509580e + 00$	$0.00000e + 00$
$6.309573444801933e + 00$	$6.309573444801929e + 00$	$7.03834e - 16$
$1.584893192461114e + 01$	$1.584893192461114e + 01$	$1.12081e - 16$
$3.981071705534973e + 01$	$3.981071705534988e + 01$	$3.56961e - 15$
$1.0000000000000000e + 02$	$1.000000000000010e + 02$	$1.00897e - 14$
$2.511886431509580e + 02$	$2.511886431509614e + 02$	$1.38042e - 14$
$6.309573444801930e + 02$	$6.309573444802187e + 02$	$4.07210e - 14$
$1.584893192461114e + 03$	$1.584893192461186e + 03$	$4.54778e - 14$
$3.981071705534973e + 03$	$3.981071705533795e + 03$	$2.95849e - 13$
$1.0000000000000000e + 04$	$9.999999999990539e + 03$	$9.46056e - 13$

Quindi numericamente  $F$  e  $F^*$  non coincidono, anche per valori non eccessivi di  $x$ .

## 6. Successione di Archimede

Esistono più successioni che approssimano  $\pi$  e sono attribuite ad Archimede. Ad esempio, osservato che  $\pi$  non è altro che l'area del cerchio avente raggio 1, si inscrivono nel cerchio al variare di  $p = 2, 3, \dots$  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.

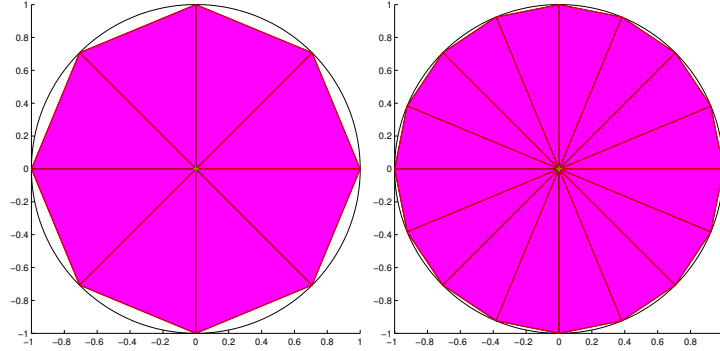


Figura 4: Grafico che illustra le suddivisioni considerate dall'algoritmo di Archimede, per  $p = 3, p = 4$ .

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  $\theta$  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  $\pi$  con stime dall'alto e dal basso, via anche poligoni circoscritti. Come citato in [21]:

Nel breve lavoro "*La misura del cerchio*" viene dimostrato anzitutto che un cerchio è 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  $\pi$ . Le stime esplicitamente ottenute limitano questo valore fra  $22/7$  e  $3 + 10/71$ .

Secondo [31], Archimede usò un altro algoritmo per approssimare  $\pi$ .

Visto che  $2\pi$  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  $\pi$  come

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

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

**A.** Si approssimi  $\pi$  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  $\pi$  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  $\pi$  con 10 cifre decimali esatte? Per  $k = 0, \dots, 4$  si ha

$$3.00000 \leq \pi \leq 3.46410 \quad (29)$$

$$3.10583 \leq \pi \leq 3.21539 \quad (30)$$

$$3.13263 \leq \pi \leq 3.15966 \quad (31)$$

$$3.13935 \leq \pi \leq 3.14609 \quad (32)$$

$$3.14103 \leq \pi \leq 3.14271 \quad (33)$$

**Nota. 6.1.** Il fatto che ci sia un  $\pi$  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  $\pi$ . In questi esercizi, per semplificare la questione si usa però  $\pi$  per calcolare  $\pi$ , 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 riuscì ad affermare che

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

## 7. Qualche esercizio

1. Posto  $h = 10^{-1}, 10^{-2}, \dots, 10^{-15}$ , 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  $h$ ?



2. **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.
3. **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.

## 8. Complessità computazionale

Descriviamo in questa sezione alcuni problemi in cui mostriamo come pur fornendo in aritmetica esatta lo stesso risultato, abbiano complessità computazionale diversa, ovvero compiono un numero diverso di operazioni. Tipicamente, visto che nei primi computer il costo tra operazioni moltiplicative e somme era molto diverso, si considerano solo prodotti (e quindi potenze) e divisioni.

### 8.1. Valutazione di polinomi

Si supponga di dover valutare il polinomio di grado  $n = 4$

$$p_4(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4.$$

1. Se si implementa un metodo brutale, necessitano

- 4 somme;
- 3 potenze, e visto che  $x^4 = x \cdot x$ ,  $x^3 = x \cdot x \cdot x$ ,  $x^4 = x \cdot x \cdot x \cdot x$ , necessitano  $1 + 2 + 3$  prodotti;
- 4 prodotti del tipo  $a_k \cdot x^k$ .

Con questa tecnica in generale servono per valutare un polinomio di grado  $n$  nel generico punto  $x$

- $n$  somme;
- $n - 1$  potenze, e visto che  $x^k = x \cdot \dots \cdot x$ , necessitano  $1 + \dots + (n - 1) = (n - 1)n/2$  prodotti.
- $n$  prodotti del tipo  $a_k \cdot x^k$ .

e quindi  $\frac{(n-1)n}{2} + n = \frac{(n+1)n}{2}$  operazioni moltiplicative

2. Facendo un pó di attenzione è facile osservare che in realtà , per valutare  $p_4$  servono

- 4 somme;
- 3 potenze, e visto che  $x^2 = x \cdot x$ ,  $x^3 = x^2 \cdot x$ ,  $x^4 = x^3 \cdot x$ , necessitano solo 3 prodotti;
- 4 prodotti del tipo  $a_k \cdot x^k$ .

Quindi, il numero di operazioni moltiplicative con un algoritmo più furbo è pari a 7.

In generale, si vede che con questa tecnica servono  $2n - 1$  operazioni moltiplicative per valutare un polinomio  $p$  di grado esattamente  $n$ , nel generico punto  $x$ .

3. Osserviamo che nel caso dell'esempio del polinomio di grado 4,

$$\begin{aligned}
 p_4(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 \\
 &= a_0 + x(a_1 + a_2x + a_3x^2 + a_4x^3) \\
 &= a_0 + x(a_1 + x(a_2 + a_3x + a_4x^2)) \\
 &= a_0 + x(a_1 + x(a_2 + x(a_3 + a_4x)))
 \end{aligned}$$

e quindi posti

- $b_3 := a_3 + a_4x$ ,
- $b_2 := a_2 + b_3x$ ,
- $b_1 := a_1 + b_2x$ ,
- $b_0 := a_0 + b_1x$ ,

otteniamo da (34)

$$\begin{aligned}
 p_4(x) &= a_0 + x(a_1 + x(a_2 + x(a_3 + a_4x))) \\
 &= a_0 + x(a_1 + x(a_2 + b_3x)) \\
 &= a_0 + x(a_1 + b_2x) \\
 &= a_0 + b_1x \\
 &= b_0.
 \end{aligned} \tag{34}$$

Quindi per valutare il polinomio nel generico punto  $x$  servono 4 somme e solo 4 prodotti, operazioni necessarie per calcolare nell'ordine  $b_3, \dots, b_0$ .

Non è difficile vedere che in generale, con questa regola, usualmente detta di *Ruffini-Horner* [28], occorrono  $n$  operazioni moltiplicative per valutare il polinomio  $p$  di grado  $n$  nel generico punto  $x$ , quindi in numero inferiore che con il primo e secondo metodo.

**Nota storica. 8.1.** In [28], relativamente alla paternità del metodo si riporta che

*Horner è ricordato principalmente per il metodo, la regola di Horner, per risolvere le equazioni algebriche che Augustus De Morgan e altri gli attribuirono.*

*Pubblicò un articolo sul metodo il 1 luglio del 1819 sulla rivista *Philosophical Transactions della Royal Society* di Londra.*

*Ma Fuller fece notare che, contrariamente all'opinione di De Morgan, questo articolo non conteneva il metodo, che fu invece pubblicato da Horner solo nel 1830. Fuller scoprì che Theophilus Holdred, un orologiaio inglese, pubblicò il metodo in un articolo del 1820, e paragonò il plagio di Horner ad un vero e proprio ladrocinio.*

*Questa discussione è alquanto discutibile dal momento che il metodo fu anticipato nel XIX secolo in Europa da Paolo Ruffini (per il quale vinse la medaglia d'oro attribuita a chi avesse migliorato le soluzioni numeriche delle equazioni dalla Societ Italiana per le Scienze Matematiche), ma è stato, in ogni caso, considerato da Zhu Shijie in Cina nel XIII secolo.*

## 8.2. Calcolo delle potenze

Si supponga di dover calcolare esclusivamente

$$a^n$$

dove  $a \in \mathbb{R}$  e  $n \in \mathbb{N}$ .

1. Un metodo brutale, osservato che

$$a^n = a \cdot \dots \cdot a$$

richiede  $n - 1$  operazioni moltiplicative.

2. Sia  $n = \sum_{j=0}^m c_j 2^j$  con  $m = \lfloor \log_2 n \rfloor$ , la codifica binaria di  $n$ , allora per le proprietà delle potenze

$$\begin{aligned} a^n &= a^{\sum_{j=0}^m c_j 2^j} = \prod_{j=0}^m a^{c_j 2^j} \\ &= a^{c_0} \cdot (a^2)^{c_1} \cdot (a^4)^{c_2} \dots (a^{2^m})^{c_m} \end{aligned} \quad (35)$$

Notiamo che

- $a^2 = a \cdot a$ ,
- $a^4 = a^2 \cdot a^2$ ,
- $a^8 = a^4 \cdot a^4$ ,
- $\dots$ ,
- $a^{2^m} = a^{2^{m-1}} \cdot a^{2^{m-1}}$ ,

e quindi per calcolare *tutte* le potenze  $a^2, a^4, \dots, a^{2^m}$  necessitano  $m = \lfloor \log_2 n \rfloor$  prodotti.

Osserviamo inoltre che

- se  $c_j = 0$  allora  $a^{c_j 2^j} = 1$ ,
- se  $c_j = 1$  allora  $a^{2^j c_j} = a^{2^j}$ .

I prodotti di termini della forma  $(a^{2^k})^{c_k}$  in (35) sono infine  $m$ .

Un codice che implementa la conversione da decimale in binario è

```
1 function c=dec2bin(n)
2 j=1; t=n;
3 while t > 0
4     q=t/2; t=floor(q);
5     if t == q
6         c(j)=0;
7     else
8         c(j)=1;
9     end
10    j=j+1;
11 end
```

che richiede  $m + 1$  divisioni per determinare i coefficienti  $c_k$ ,  $m + 1$  somme e  $m + 1$  parti intere che sono indicate con `floor` e hanno un costo molto inferiore alla divisione e alla somma.

Ne ricaviamo che per calcolare la potenza richiesta servono  $3m + 1 = 3\lfloor \log_2(n) \rfloor + 1$  operazioni moltiplicative, visto che

- la valutazione dei coefficienti  $c_0, \dots, c_m$  richiede  $m + 1$  divisioni;
- la valutazione delle potenze  $a^{2^k}$ ,  $k = 0, \dots, m$  consta di  $m$  operazioni moltiplicative;
- calcolate le potenze, la valutazione finale di (35), necessita al più  $m$  operazioni moltiplicative (alcuni coefficienti  $c_j$  potrebbero essere nulli e quindi  $a^{c_j 2^j} = 1$ ).

In tabella descriviamo  $n$  che è circa la complessità del primo algoritmo e  $3m + 1 = 3\lfloor \log_2(n) \rfloor + 1$ , che mostra come per esponenti relativamente grandi il secondo algoritmo sia di gran lunga da preferire visto che  $3\lfloor \log_2(n) \rfloor + 1 \ll n - 1$ . Con un po' più di attenzione, si vede che ciò accade per  $n > 11$ .

$n$	$3\lfloor \log_2(n) \rfloor + 1$
1.0e + 01	10.96578428466209
1.0e + 02	20.93156856932417
1.0e + 03	30.89735285398626
1.0e + 04	40.86313713864835
1.0e + 05	50.82892142331043
1.0e + 06	60.79470570797253
1.0e + 07	70.76048999263460
1.0e + 08	80.72627427729670
1.0e + 09	90.69205856195879
1.0e + 10	100.6578428466209

**Nota. 8.2.** Notiamo che se  $B, C$  sono due matrici quadrate di ordine  $s$ , il prodotto  $B \cdot C$  richiede  $s^2$  moltiplicazioni e  $(s - 1)s \approx s^2$  somme.

Quindi per determinare

$$A^n = \underbrace{A \cdot \dots \cdot A}_n$$

servono  $n - 1$  prodotti di matrici di dimensione " $s$ " ovvero  $(n - 1)s^2$  moltiplicazioni.

Se usiamo il secondo algoritmo

- la valutazione dei coefficienti  $c_0, \dots, c_m$  richiede  $m + 1$  divisioni;
- la valutazione delle potenze  $a^{2^k}$ ,  $k = 0, \dots, m$  necessita la determinazione di " $m$ " potenze di matrici (si noti che non si devono fare calcoli per ricavare  $A^0 = I$ ,  $A^1 = A$ ) e quindi  $ms^2$  moltiplicazioni (e circa le stesse somme);

- calcolate le potenze, la valutazione finale di (35), necessita al più  $m$  prodotti di matrici (alcuni coefficienti  $c_j$  potrebbero essere nulli e quindi  $a^{c_j 2^j} = 1$ ), e bisogna eseguire circa  $ms^2$  operazioni moltiplicative come pure additive.

Di conseguenza l'applicazione del secondo algoritmo richiede al più

$$(m+1) + 2ms^2 \approx 2ms^2 = 2\lfloor \log_2(n) \rfloor s^2$$

operazioni moltiplicative come pure additive.

Quindi il secondo metodo è da preferirsi qualora  $2\lfloor \log_2(n) \rfloor s^2 < (n-1)s^2$  ovvero quando  $2\lfloor \log_2(n) \rfloor < n-1$ , che si verifica per  $n > 4$  mentre per  $n = 4$  la complessità è la stessa.

### 8.3. Valutazione dell'esponenziale

Supponiamo di dover valutare  $f(x) = e^x$ , con  $x > 1$ .

Posto

$$S_n^{(1)}(x) = \sum_{i=0}^n \frac{x^i}{i!}.$$

abbiamo che

$$E_n^{(1)}(x) = |f(x) - S_n^{(1)}(x)| = \frac{\xi^{n+1}}{(n+1)!}, \quad \xi \in (1, x)$$

e se  $x$  è grande, il numeratore potrebbe essere abbastanza rilevante da rendere l'errore  $E_n^{(1)}(x)$  non troppo piccolo, vista la presenza di  $\xi > 1$ .

Vediamo una alternativa, valutando

$$S_{n,m}^{(2)}(x) = \left( \sum_{i=0}^n \frac{x^i}{i! m^i} \right)^m. \quad (36)$$

Vale il seguente

**Teorema 8.3.** Sia  $x > 1$  e  $m > \lceil x \rceil$ . Poniamo

$$S_{n,m}^{(2)}(x) = \left( \sum_{i=0}^n \frac{x^i}{i! m^i} \right)^m.$$

Allora

$$|f(x) - S_{n,m}^{(2)}(x)| \approx \frac{m}{e^x} \cdot \frac{\xi_2^{n+1}}{(n+1)!}, \quad \xi_2 \in (1/m, x/m).$$

**Dimostrazione facoltativa.** Supponiamo  $m > \lceil x \rceil$ , e osserviamo che

$$f(x) = (e^{x/m})^m = \left( \sum_{i=0}^n \frac{x^i}{i! m^i} + \frac{\xi_2^{n+1}}{(n+1)!} \right)^m, \quad \xi_2 \in (1/m, x/m).$$

Per  $t \approx 0$  abbiamo che

$$(1+t)^m \approx 1+mt$$

e quindi se  $u \ll v$  allora

$$(u+v)^m = (v(1+(u/v)))^m = v^m (1+(u/v))^m \approx v^m \left(1 + \frac{mu}{v}\right). \quad (37)$$

Posto

$$S_{n,m}^{(2)}(x) = \left(\sum_{i=0}^n \frac{x^i}{i!m^i}\right)^m$$

abbiamo che  $x/m < 1$  come pure, visto che  $S_{n,m}^{(2)}(x) \geq 1$ , necessariamente

$$(S_{n,m}^{(2)}(x))^{1/m} \geq 1$$

e infine

$$\frac{\xi_2^{n+1}}{(n+1)!} < 1,$$

da cui posto

- $u = \frac{\xi_2^{n+1}}{(n+1)!} < 1,$
- $v = (S_{n,m}^{(2)}(x))^{1/m},$

ricaviamo da (37)

$$\begin{aligned} \left((S_{n,m}^{(2)}(x))^{1/m} + \frac{\xi_2^{n+1}}{(n+1)!}\right)^m &\approx ((S_{n,m}^{(2)}(x))^{1/m})^m + \frac{m \frac{\xi_2^{n+1}}{(n+1)!}}{(S_{n,m}^{(2)}(x))^{1/m}} \\ &= S_{n,m}^{(2)}(x) + \frac{m \frac{\xi_2^{n+1}}{(n+1)!}}{(S_{n,m}^{(2)}(x))^{1/m}} \end{aligned}$$

per cui

$$\begin{aligned} E_n^{(2)}(x) &= |f(x) - S_n^{(2)}(x)| = |((S_{n,m}^{(2)}(x))^{1/m} + \frac{\xi_2^i}{(i+1)!})^m - S_{n,m}^{(2)}(x)| \\ &\approx \left| S_{n,m}^{(2)}(x) + \frac{m \frac{\xi_2^{n+1}}{(n+1)!}}{(S_{n,m}^{(2)}(x))^{1/m}} - S_{n,m}^{(2)}(x) \right| \\ &= \left| \frac{m \frac{\xi_2^{n+1}}{(n+1)!}}{(S_{n,m}^{(2)}(x))^{1/m}} \right| = \frac{m \frac{\xi_2^{n+1}}{(n+1)!}}{(S_{n,m}^{(2)}(x))^{1/m}}. \end{aligned}$$

La dimostrazione si conclude osservando che il denominatore è maggiore di 1, dell'ordine di  $e^x$  qualora  $n$  sia sufficientemente grande.

**Commento.** La quantità

$$\frac{m}{e^x} \cdot \frac{\xi_2^{n+1}}{(n+1)!}, \quad \xi_2 \in (1/m, x/m).$$

è favorevole perchè

- $\xi_2 \in (1/m, x/m) \subset (1/m, 1)$  e quindi la potenza  $\xi_2^{n+1}$  diventa *piccola* al crescere di  $n$ ;
- se scegliamo  $x < m < e^x$  abbiamo  $m/e^x < 1$ .

In qualche senso, il teorema precedente asserisce che con  $n$  non troppo grande si può realizzare mediante *troncate* di Taylor una buona approssimazione di  $e^{x/m}$  e pure di  $e^x$  dopo elevamento del risultato all' $m$ -sima potenza.

**Nota. 8.4.** Una volta calcolato  $e^{x/m}$  con una troncata di Taylor di ordine  $n$  (che abbiamo visto essere non troppo grande, qualora  $m$  sia scelto in modo appropriato), l'elevamento alla potenza  $m$ -sima del risultato può essere convenientemente realizzato utilizzando l'algoritmo rapido esposto precedentemente, ovvero quello che sfrutta la conversione di  $m$  in binario.

**Esempio.** Quale esempio calcoliamo

$$e^{10 \cdot \pi} \approx 4.403150586063198e + 13,$$

con il primo e il secondo algoritmo, in cui utilizziamo quali parametri  $m_1 = 32$ ,  $m_2 = 36$ ,  $m_3 = 41$ , che sono maggiori di  $10\pi \approx 31.4$ .

In tabella, mostriamo al variare di  $n$  gli errori *relativi*

$$\begin{aligned} \tilde{E}_n^{(1)} &= E_n^{(1)} / e^{10 \cdot \pi} \\ \tilde{E}_{n, m_k}^{(2)} &= E_{n, m_k}^{(2)} / e^{10 \cdot \pi}, \quad k = 1, 2, 3, \end{aligned}$$

che evidenziano i vantaggi del secondo approccio.

$n$	$\tilde{E}_n^{(1)}$	$\tilde{E}_{n, m_1}^{(2)}$	$\tilde{E}_{n, m_2}^{(2)}$	$\tilde{E}_{n, m_3}^{(2)}$
5	$1.0e + 00$	$1.6e - 02$	$1.0e - 02$	$5.7e - 03$
10	$1.0e + 00$	$2.4e - 07$	$8.2e - 08$	$2.5e - 08$
15	$1.0e + 00$	$3.7e - 13$	$6.6e - 14$	$5.9e - 15$
20	$9.8e - 01$	$6.6e - 15$	$9.0e - 15$	$2.7e - 15$

**Nota. 8.5.** Una tecnica alternativa, comunemente utilizzata per calcolare  $e^x$ , è la seguente.

Posto  $y = x \log_2 e$ , visto che  $y = (y - [y]) + [y]$  dove  $[y]$  è l'arrotondamento all'intero più vicino a  $y$ , abbiamo

$$e^x = 2^y = 2^{(y - [y]) + [y]} = 2^{y - [y]} 2^{[y]}.$$

Si comincia calcolando  $2^{y-[y]}$ , con  $y - [y] \in [0, 1/2]$  (il termine  $1/2$  è dovuto al fatto che abbiamo utilizzato l'arrotondamento all'intero più vicino) cosa che può essere fatta convenientemente valutando, con la regola di Horner, un certo polinomio di grado  $n$  non troppo alto che approssima l'esponenziale in  $[0, 1/2]$ , fornendo indipendentemente da  $x$  un errore inferiore dell'ordine della precisione di macchina.

Il polinomio richiesto ha la forma

$$\begin{aligned} p_9(x) = & 1 - 0.15639 \cdot 10^{-16} + (0.69314718055995154416 \\ & + (0.24022650695869054994 + (0.055504108675285271942 \\ & + (0.0096181289721472527028 + (0.0013333568212100120656 \\ & + (0.00015403075872034576440 + (0.000015265399313676342402 \\ & + (0.0000013003468358428470167 \\ & + 0.00000012113766044841794408 \cdot x) \cdot x) \cdot x) \cdot x) \cdot x) \cdot x) \cdot x) \cdot x. \end{aligned}$$

e si verifica che per  $x \in [0, 1/2]$  si ha  $|e^x - p_9(x)| \approx 10^{-16}$ .

Una volta disponibile l'approssimazione di  $2^{y-[y]}$  (bastano 9 moltiplicazioni), la si moltiplica per  $2^{[y]}$ .

Osserviamo che  $2^{[y]}$  non deve essere calcolato, visto che viene determinato in base 2 come numero con esponente  $[y]$  e mantissa  $(1, 0 \dots, 0)_2$ . Quindi per ottenere  $e^x$  basta aggiungere  $[y]$  all'esponente del numero  $2^{y-[y]}$  in notazione binaria.

#### 8.4. Calcolo del determinante

Si supponga di avere una matrice quadrata  $A = (a_{i,j}) \in \mathbb{R}^{n \times n}$ .

Sia

- la matrice  $A_{ij}^{(n-1)} \in \mathbb{R}^{(n-1) \times (n-1)}$ , la sottomatrice estratta da  $A$  cancellando la  $i$ -esima riga e la  $j$ -esima colonna.
- la quantità  $\det(A_{ij}^{(n-1)})$ , definita come *minore complementare* dell'elemento  $(i, j)$ ;
- la quantità  $(-1)^{i+j} \det(A_{ij}^{(n-1)})$ , definita come *complemento algebrico* dell'elemento  $(i, j)$ ;

Il primo teorema di Laplace afferma che  $\det(A)$  (cf. [23]) è uguale alla somma dei prodotti degli elementi di una riga qualsiasi (o una colonna qualsiasi) per i rispettivi complementi algebrici. In altri termini si definisce la *formula ricorsiva di Laplace*

$$\det(A) = \sum_{j=1}^n (-1)^{i+j} a_{ij} \det(A_{ij}^{(n-1)}), \quad i, j = 1, \dots, n.$$

il cui costo computazionale è maggiore di  $n!$  flops.

Infatti, se  $c_n$  è il numero di operazioni moltiplicative della regola di Laplace per il calcolo di  $\det(A)$  con  $A \in \mathbb{R}^{n \times n}$  allora, essendo  $c_2 = 2$  (il calcolo del determinante



di una matrice di ordine 2 richiede infatti due operazioni moltiplicative),

$$\begin{aligned}
c_n &= n + n \cdot c_{n-1} > n \cdot c_{n-1} \\
&= n((n-1) + c_{n-2}) > n(n-1) \cdot c_{n-2} \\
&> n \cdot \dots \cdot 3 \cdot c_2 = n \cdot \dots \cdot 3 \cdot 2 = n!
\end{aligned} \tag{38}$$

Se  $A = LU$  come dalla fattorizzazione  $LU$  senza pivoting (cf. [22]), con

- $L = (l_{i,j})$  triangolare inferiore, ovvero  $l_{i,j} = 0$  se  $i < j$ , con componenti diagonali uguali a 1,
- $U = (u_{i,j})$  triangolare superiore, ovvero  $u_{i,j} = 0$  se  $i > j$ ,

per il teorema di Binet

$$\det(A) = \det(LU) = \det(L) \cdot \det(U).$$

Osserviamo che

- visto che il determinante di una matrice triangolare  $T = (t_{i,j}) \in \mathbb{R}^{n \times n}$  è

$$\det(T) = \prod_{k=1}^n t_{k,k}$$

abbiamo che da  $l_{k,k} = 1, k = 1, \dots, n$ , si ricava  $\det(L) = 1$ , e quindi

$$\det(A) = \prod_{k=1}^n u_{k,k}.$$

In definitiva, visto che  $\prod_{k=1}^n u_{k,k}$  necessita solo di  $n - 1$  operazioni moltiplicative, mentre la fattorizzazione  $A = LU$  ha complessità dell'ordine di  $n^3/3$ , il calcolo del determinante può essere effettuato con circa  $n^3/3$  operazioni moltiplicative.

Nella tabella paragoniamo il tempo di calcolo necessario a un supercomputer come Summit che eroga 200 petaflops, ovvero  $2 \cdot 10^{17}$  operazioni al secondo, per calcolare il determinante di una matrice generica di ordine  $n$ . Si tenga conto che si presume che l'età dell'universo sia di circa "13.82e + 9" anni.

$n$	$CPU_L$	$CPU_{LU}$	$n$	$CPU_L$	$CPU_{LU}$
5	6.0e - 16 sec	2.1e - 16 sec	75	1.2e + 92 anni	7.0e - 13 sec
10	1.8e - 11 sec	1.7e - 15 sec	100	4.7e + 140 anni	1.7e - 12 sec
25	9.0e + 02 anni	2.6e - 14 sec	125	9.4e + 191 anni	3.3e - 12 sec
50	1.8e + 42 anni	2.1e - 13 sec	150	2.9e + 245 anni	5.6e - 12 sec

Tabella 1: In questa tabella paragoniamo il tempo di calcolo  $CPU_L, CPU_{LU}$  necessario a un supercomputer come Summit che eroga 200 petaflops, ovvero  $2 \cdot 10^{17}$  operazioni al secondo, per calcolare il determinante di una matrice generica di ordine  $n$ , rispettivamente con la regola di Laplace e via fattorizzazione  $PA = LU$ .

## Bibliografia

- [1] D. Arnold,  
<http://www.ima.umn.edu/~arnold/disasters/>
- [2] K. Atkinson, *An Introduction to Numerical Analysis*, Wiley, (1989).
- [3] K. Atkinson and W. Han, *Elementary Numerical Analysis*, third edition, Wiley, (2004).
- [4] N. Brisebarre and collaborators *Floating-Point Arithmetic*, 2009.  
<https://perso.ens-lyon.fr/jean-michel.muller/chapitre1.pdf>
- [5] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.
- [6] V. Comincioli, *Problemi di analisi numerica*, Mc Graw-Hill, 1991.
- [7] M. Frontini e E. Sormani, *Fondamenti di Calcolo Numerico, problemi in laboratorio*, Apogeo, 2005.
- [8] W. Gautschi, *Numerical Analysis*, second edition, Birkhäuser, 2012.
- [9] D. Goldberg, *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, Association for Computing Machinery, Inc, Computing Surveys, March 1991.
- [10] T. Huckle,  
<http://www.zenger.informatik.tu-muenchen.de/persons/huckle/bugse.html>.
- [11] Mars Program,  
<http://marsprogram.jpl.nasa.gov/msp98/orbiter>
- [12] J.H. Mathews e K.D. Fink, *Numerical Methods using Matlab*, Prentice Hall, 1999.
- [13] Network Theory, *GNU Octave Manual Version*,  
[http://www.network-theory.co.uk/docs/octave3/octave\\_160.html](http://www.network-theory.co.uk/docs/octave3/octave_160.html).
- [14] A. Quarteroni, *Elementi di calcolo numerico*, seconda edizione, Progetto Leonardo, 1999.
- [15] A. Quarteroni e F. Saleri, *Introduzione al calcolo scientifico*, Springer Verlag, 2006.
- [16] A. Quarteroni, R. Sacco e F. Saleri, *Matematica Numerica*, Springer Verlag, 1998.
- [17] J. Stoer, *Introduzione all'analisi numerica*, Zanichelli, 1984.
- [18] The MathWorks Inc.,  
*Numerical Computing with Matlab*, <http://www.mathworks.com/moler>.
- [19] L.N. Trefethen, *Numerical Analysis*,  
<http://people.maths.ox.ac.uk/trefethen/NAessay.pdf>.

- [20] Web Page ,  
<http://utenti.quipo.it/base5/numeri/pigreco.htm>.
- [21] Wikipedia (Archimede),  
<http://it.wikipedia.org/wiki/Archimede>
- [22] Wikipedia (Decomposizione LU),  
[https://it.wikipedia.org/wiki/Decomposizione\\_LU](https://it.wikipedia.org/wiki/Decomposizione_LU)
- [23] Wikipedia (Determinante),  
<https://it.wikipedia.org/wiki/Determinante>
- [24] Wikipedia (Floating Point),  
[http://it.wikipedia.org/wiki/Floating\\_point](http://it.wikipedia.org/wiki/Floating_point)
- [25] Wikipedia (IEEE 754),  
[http://it.wikipedia.org/wiki/IEEE\\_754](http://it.wikipedia.org/wiki/IEEE_754)
- [26] Wikipedia (IEEE 754r),  
[http://it.wikipedia.org/wiki/IEEE\\_754r](http://it.wikipedia.org/wiki/IEEE_754r)
- [27] Wikipedia (Pi Greco),  
[http://it.wikipedia.org/wiki/Pi\\_greco](http://it.wikipedia.org/wiki/Pi_greco)
- [28] Wikipedia (Regola di Horner),  
[https://it.wikipedia.org/wiki/Regola\\_di\\_Horner](https://it.wikipedia.org/wiki/Regola_di_Horner)
- [29] Wikipedia (Teorema di Laplace),  
[https://it.wikipedia.org/wiki/Teorema\\_di\\_Laplace](https://it.wikipedia.org/wiki/Teorema_di_Laplace).
- [30] Wikipedia (Teorema Fondamentale del Calcolo Integrale),  
[http://it.wikipedia.org/wiki/Teorema\\_fondamentale\\_del\\_calcolo\\_integrale](http://it.wikipedia.org/wiki/Teorema_fondamentale_del_calcolo_integrale)
- [31] Wolfram (Archimedes' Algorithm),  
<http://mathworld.wolfram.com/ArchimedesAlgorithm.html>