Numeri macchina e propagazione degli errori

Alvise Sommariva

Università degli Studi di Padova

15 marzo 2023

(Rappresentazione floating point dei numeri reali)

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

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

dove

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

è la funzione segno.

Esempio (Base binaria e decimale)

Casi particolari di

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

- *in base 2 (detta anche base binaria, usato nello standard IEEE 754-1985),*

$$x = sgn(x) \cdot 2^e \sum_{k=1}^{+\infty} d_k \cdot 2^{-k}, \quad d_1 = 1,\ d_k = 0, 1,\ e \in \mathbb{Z}$$

- *in base 10 (detta anche base decimale, usata nella vita quotidiana),*

$$x = sgn(x) \cdot 10^e \sum_{k=1}^{+\infty} d_k \cdot 10^{-k}, \quad d_1 = 1, \ldots, 9,\ d_k = 0, \ldots, 9,\ e \in \mathbb{Z}$$

Esempio

Rappresentare il numero

$$\pi = 3,14159265358979323\ldots$$

nella forma

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

per $\beta = 10$ *e* $\beta = 2$.

Base 10: $3,14159265358979323\ldots$ *secondo la rappresentazione*

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

risulta

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

che viene spesso denotato come

$$\pi = (+1) \cdot (0.314159265358979323\ldots)_{10} \cdot 10^1;$$

Base 2: *il numero* π*, si scrive come*

$$+1 \cdot 2^2 \cdot (1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} \ldots) = +4(1/2 + 1/4 + \ldots),$$

che viene spesso denotato come

$$\pi = (+1) \cdot (0.11001001000011111101101010100010001000010101\ldots)_2 \cdot 2^2.$$

Esempio

Rappresentare il numero

$$\exp(10) = 22026.465794806717894971 37\ldots$$

nella forma

$$sgn(\exp(10))\beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k}, \quad 0 < d_1,\ 0 \leq d_k \leq \beta - 1,\ e \in \mathbb{Z}$$

per $\beta = 10$.

Per quanto visto

$$\begin{aligned} \exp(10) &= (+1) \cdot 10^5 \cdot (2 \cdot 10^{-1} + 2 \cdot 10^{-2} + 0 \cdot 10^{-3} + 2 \cdot 10^{-4} + \ldots). \\ &= (+1) \cdot 10^5 \cdot (0.22026465794806717894971 37\ldots)_{10} \end{aligned}$$

La particolarità della sommatoria in (1), ovvero

$$x = \mathrm{sgn}(x)\beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k}, \quad 0 < d_1,\ 0 \leq d_k \leq \beta - 1,\ e \in \mathbb{Z} \tag{3}$$

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

Definizione (Numeri macchina)

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

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

con $0 \leq d_k \leq \beta - 1$ dove

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

1. *La parola normalizzato sottolinea che $d_1 > 0$ e comporta l'unicitá della rappresentazione.*
2. *Ogni elemento di $F(\beta, t, L, U)$ è detto numero macchina.*
3. *Ogni numero macchina é razionale.*
4. *Si può provare che $\sum_{k=1}^{t} d_k \beta^{-k} < 1$ e quindi $e = \texttt{floor}(1 + \log_\beta(y))$.*

Osserviamo che se $e > 0$ allora

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

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

In altri termini, se $e > 0$, le prime "e" cifre determinano la parte intera, le altre la parte frazionaria.

Nota.

Si noti che la parte intera ha potenze non negative di β e quindi effettivamente è un numero naturale, mentre la parte frazionaria è somma di potenze negative di β, che si mostra con un po' di tecnica essere un numero in $[0,1)$.

Esempio

Con riferimento a

$$\exp(10) = (+1) \cdot 10^5 \cdot (0.22026465794806717894971 37)_{10}.$$

visto che $e = 5 > 0$, *le prime 5 cifre della mantissa sono relative alla parte intera, le altre alla parte frazionaria.*
In effetti

$$\exp(10) = 22026.465794806717894971 37\ldots$$

Nota.

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) perchè l'intervallo di valori rappresentati è modesto e la precisione dei numeri frazionari scarsa.

Di ogni numero macchina x

$$x = \mathrm{sgn}(x) \cdot \overbrace{(0.d_1 \ldots d_t)_\beta}^{\text{mantissa}} \beta^e = \mathrm{sgn}(x) \cdot \beta^e \sum_{k=1}^{t} d_k \beta^{-k}, \quad d_1 > 0,$$

si immagazzinano

- una cifra per il segno (ovvero 0 per positivo e 1 per negativo),
- le cifre dell'esponente, diciamo m,
- le cifre della mantissa, diciamo t.

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.

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

$$y = sgn(y) \cdot (1.d_2 \ldots d_t)_\beta \beta^e = sgn(y) \cdot \beta^e (1 + \sum_{k=2}^{t} d_k \beta^{-k}) \tag{4}$$

e non come in precedenza

$$\begin{aligned} y &= sgn(y) \cdot (1.d_1 \ldots d_{t-1})_\beta \beta^e \\ &= sgn(y) \cdot \beta^e (\beta^{-1} + \sum_{k=2}^{t} d_{k-1} \beta^{-k}). \end{aligned} \tag{5}$$

Questo potenzialmente può creare qualche confusione. Scegliamo di utilizzare (5) perchè più frequente nei manuali di analisi numerica.

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

Proposizione.

Il piú piccolo numero macchina positivo (normalizzato) é β^{L-1}.

Dimostrazione.

Ricordiamo che un generico numero macchina x é

$$x = \text{sgn}(x)\beta^e \sum_{k=1}^{t} d_k \beta^{-k},\ d_1 > 0, L \leq e \leq U.$$

Ponendo $d_1 = 1$, $d_2 = \ldots = d_t = 0$ ed $e = L$, otteniamo che il più piccolo numero macchina (e quindi normalizzato!) positivo rappresentabile è

$$x^* = \beta^L(1 \cdot \beta^{-1} + \sum_{k=2}^{t} 0 \cdot \beta^{-k}) = \beta^{L-1}.$$

△

Proposizione.

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 facoltativa.

La somma $\sum_{k=1}^{t} d_k \beta^{-k}$ risulta piu' grande possibile quando $d_k = \beta - 1$. Essendo

$$\sum_{k=1}^{t} \gamma^k = \frac{\gamma^{t+1} - 1}{\gamma - 1} - 1$$

necessariamente tale numero m^* é

$$\begin{aligned} m^* &= \sum_{k=1}^{t} (\beta-1)\beta^{-k} = (\beta-1)\sum_{k=1}^{t} \beta^{-k} = (\beta-1)\left(\frac{\beta^{-(t+1)} - 1}{\beta^{-1} - 1} - 1\right) \\ &= (\beta-1)\left(\beta\frac{\beta^{-(t+1)} - 1}{1-\beta} - 1\right) = -\beta^{-t} + \beta - \beta + 1 = 1 - \beta^{-t}, \end{aligned}$$

con $1 - \beta^{-t} < 1$ in quanto $\beta \geq 2$, $t \geq 1$. $\triangle$

Proposizione.

Il piú grande numero macchina (normalizzato) é $\beta^U(1-\beta^{-t})$, dove t é il numero di cifre della mantissa.

Dimostrazione facoltativa.

Osserviamo che

- ponendo $d_k = \beta - 1$, per $k = 1, \ldots, t$ ed $e = U$ in $\beta^e \sum_{k=1}^{t} d_k \beta^{-k}$,
- essendo la somma massima relativa alla mantissa è $1 - \beta^{-t}$,

otteniamo che il massimo numero macchina rappresentabile M^* è

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

△

Alcune proprietá di $F(\beta, t, L, U)$

Proposizione.

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

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

Dimostrazione facoltativa.

Per una fissata mantissa, a t cifre, il numero piú piccolo che possiamo scrivere è $(1\,0\ldots 0\,0)_\beta$ mentre il più grande è

$$(\underbrace{\beta-1\,\beta-1\ldots\beta-1\,\beta-1}_{t})_\beta.$$

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 $\beta-1$, e quindi la loro cardinalità risulta

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

Quindi, al variare dell'esponente e in $L, L+1, \ldots, U-1, U$, e del segno $+$ o $-$, i numeri floating-point sono $2(\beta-1)\beta^{t-1}(U-L+1)$. △

Nota.

La cardinalitá di $F(\beta, t, L, U)\cup\{0\}$ é $2(\beta-1)\beta^{t-1}(U-L+1)+1$. In realtá i numeri rappresentati dal computer sono di piú se si considera che per approssimare alcune quantitá molto piccole o grandi in modulo, si usano numeri di tipo non normalizzato.

Proposizione.

1. *I numeri macchina sono numeri razionali.*
2. *I numeri macchina che hanno esponente $e \geq t$ sono interi.*

Dimostrazione facoltativa.

La scrittura dei numeri floating-point come somma di un numero finito di numeri frazionari implica che i numeri risultanti siamo 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 $i = 1, \ldots, t \geq e$ che β^{e-i} è intero, come d'altra parte i d_i, lo è anche x. $\triangle$

Proposizione.

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

Dimostrazione facoltativa.

Per capirlo, il numero positivo più piccolo, e quindi più vicino a 0 è $\beta^{L-1} \ll 1$ 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* $2.2251 \cdot 10^{-308}$,
- *i due numeri piú 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

$$1.9958 \cdot 10^{292} \gg 2.2251 \cdot 10^{-308}.$$

(Riassunto)

- *Il più piccolo numero macchina positivo (normalizzato) é* β^{L-1}.
- *Il più grande numero macchina (normalizzato) é* $\beta^U(1-\beta^{-t})$, *dove* t *é il numero di cifre della mantissa.*
- *L'insieme dei numeri macchina* $F(\beta, t, L, U)$ *ha un numero finito di elementi e la sua cardinalità è*

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

- *I numeri macchina sono numeri razionali.*
- *I numeri macchina che hanno esponente* $e \geq t$ *sono interi.*
- *L'insieme dei numeri macchina* $F(\beta, t, L, U)$ *non consiste di punti equispaziati.*

Osservazione (facoltativa: Matlab e rappresentazione dei numeri)

Nel caso della notazione

$$y \quad = \quad sgn(y) \cdot (1.d_2 \ldots d_t)_\beta \beta^e = sgn(y) \cdot \beta^e (1 + \sum_{k=2}^{t} d_k \beta^{-k}) \tag{6}$$

abbiamo i seguenti risultati per $F(\beta, t, L, U)$*:*

- *la somma degli elementi della mantissa* $(1, d_1 \ldots d_{t-1}$ *è un numero nell'intervallo* $[1, 2 - \beta^{-(t-1)}] \subset [1, 2)$*;*
- *Il più piccolo numero positivo è* β^L *(usare quale mantissa* $(1.0 \ldots 0)$*);*
- *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* $(1, d_1 \ldots d_{52})$ *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* $(2 - \beta^{-(t-1)}) \cdot \beta^U = (2 - \beta^{-(52)}) \cdot 2^{1023} = 1.797693134862316e + 308.$

Alcune proprietá di $F(\beta, t, L, U)$. Osservazione facoltativa.

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

Per quanto riguarda numeri speciali

- *per rappresentare* `+inf` *si utilizza il segno pari a* 0*, tutte le cifre degli esponenti uguali a* 1 *e tutta la mantissa uguale a* 0*;*
- *per rappresentare* `-inf` *si utilizza il segno pari a* 1*, tutte le cifre degli 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 lEEE 754 i NaN sono rappresentati con il campo dell'esponenziale riempito di "*1*" e un numero diverso da zero nel campo della mantissa.*

Descriviamo di seguito i più comuni insiemi di numeri floating-point.

1. 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.

2. 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,

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

(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*).*
  (a) Dato che i numeri 0 *e* 255 *vengono utilizzati per significati speciali, sono utilizzati esclusivamente gli interi* N_e *da* 1 *a* 254*.*
  (b) Fissato `bias`$=127 = 2^7 - 1$ *per ottenere gli esponenti e li si codifica con* N_e *tale che* $e = N_e - \texttt{bias}$.

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

Nota. (Sulla precisione singola in Matlab)

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*

$$\approx 4.2950e + 09.$$

(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*).*
  (a) Dato che i numeri 0 *e* 2047 *vengono utilizzati per significati speciali, sono utilizzati esclusivamente gli interi* N_e *da* 1 *a* 2046*.*
  (b) Fissato `bias`$=1023 = 2^{10} - 1$ *per ottenere gli esponenti e li si codifica con* N_e *tale che* $e = N_e - \texttt{bias}$*.*

  Evidentemente
  - *il piú piccolo esponente è* $e_{min} = (1 - 1023) = -1022$
  - *il piú grande esponente è* $e_{max} = 2046 - 1023 = 1023$*.*

Nota. (Sulla precisione doppia in Matlab)

Nell'implementazione Matlab,

- *Il minimo numero positivo in precisione doppia é*

$$x_{min} = 2.2250738585072 01e - 308;$$

- *Il massimo numero positivo in precisione doppia é*

$$x_{max} = 1.7976931348623 16e + 308;$$

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

Osservazione (facoltativa)

Si può credere che i numeri utilizzati per le precisioni singole e doppie siano sempre stati questi, ma non è cosí. 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].*

Se x è il numero reale

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

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

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

in cui

- se si effettua il troncamento

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

- 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.

Se in particolare la base è 10 e

$$x = \mathrm{sgn}(x)(0.d_1, \ldots d_t, \ldots)_{10} 10^e = \mathrm{sgn}(x) \cdot 10^e \sum_{k=1}^{+\infty} d_k \cdot 10^{-k}, \quad d_1 > 0$$

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

$$fl(x) = \mathrm{sgn}(x)(0.d_1, \ldots, d_{t-1}, \overline{d_t}) \cdot 10^e = \mathrm{sgn}(x) 10^e (\sum_{k=1}^{t-1} d_k \cdot 10^{-k} + \overline{d_t} \cdot 10^{-t})$$

in cui se viene effettuato l'arrotondamento.

$$\overline{d_t} = \begin{cases} d_t, & d_{t+1} < 5 \\ d_t + 1, & d_{t+1} \geq 5 \end{cases}$$

Esempio

Si consideri

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

- *Se lo tronchiamo alla quarta cifra decimale otteniamo che il corrispondente numero floating point è*

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

- *se lo arrotondiamo otteniamo che il corrispondente numero floating point è*

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

Nota.

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 del minimo esponente L, si commette un'errore di underflow,
- maggiore del massimo esponente U, si commette un'errore di overflow.

Se invece

- $e \in [L, U]$,
- $x = \pm\beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k}$,
- 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.

Osservazione

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

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

Definizione

Si dice precisione di macchina, quel numero $\texttt{eps}=\beta^{1-t}$ *che rappresenta la distanza tra* 1 *ed il successivo numero in virgola mobile.*

Si vede facilmente che il numero x^* (normalizzato!) successivo a 1 ha mantissa $d_1 = 1, d_2 = 0, d_3 = 0, \ldots, d_{t-1} = 0, d_t = 1$ e esponente $e = 1$ e quindi è

$$x^* = \beta \sum_{k=1}^{t} d_k \beta^{-k} = \beta^1 (1 \cdot \beta^{-1} + 1 \cdot \beta^{-t}) = 1 + \beta^{1-t},$$

per cui $\texttt{eps} = (1 + \beta^{1-t}) - 1 = \beta^{1-t}$.

Nota.

Si noti che eps *non è il numero* $x^* > 1$ *e piú vicino a 1, ovvero* $x^* = 1 + \beta^{1-t}$, *bensı'* $|x^* - 1| = \beta^{1-t}$.

Si sottolinea che la precisione di macchina eps non coincide in generale con il più piccolo numero positivo rappresentabile dal calcolatore, in quanto se $t \neq 2 - L$, come accade per la precisione singola e doppia, allora

$$\texttt{eps} := \beta^{1-t} \neq \beta^{L-1}.$$

In Matlab, supposto si usi la precisione doppia, x_{min} è molto inferiore di eps, in quanto

- il piú piccolo numero positivo in modulo è

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

- la precisione di macchina è

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

Si supponga di approssimare un numero $x^* \in \mathbb{R}$, con $x \in \mathbb{R}$ (ad esempio $x = fl(x^*)$).

Definizione (Errore assoluto e relativo)

Si definisce

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

$$|x - x^*|$$

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

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

Osservazione

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

Esempio

Si consideri $\pi = 3.141592653589793\ldots$.

In precisione singola

$$fl(\pi) = 3.1415927$$

e abbiamo che, ponendo nella definizione precedente $x^* = \pi$, $x = fl(\pi)$

- *l'errore assoluto è* $|x - x^*| = |fl(\pi) - \pi| \approx 8.7422777 \cdot 10^{-8}$
- *l'errore relativo è* $\frac{|x-x^*|}{|x^*|} = \frac{|fl(\pi)-\pi|}{\pi} \approx 2.7827534 \cdot 10^{-8}$.

Si osservi che, come diremo in seguito, la precisione di macchina per precisione singola di Matlab (che usa l'arrotondamento!) è $\mathtt{eps}_S \approx 1.1920929 \cdot 10^{-7}$ *e si ha*

$$2.7827534 \cdot 10^{-8} \approx \frac{|\pi - fl(\pi)|}{\pi} \leq \mathtt{eps}_S/2 \approx 5.9604645 \cdot 10^{-8}.$$

Quindi l'errore relativo compiuto é inferiore o al piú uguale a $\mathtt{eps}_S/2$

Nota.

Fissato un vettore $x^* = (x_1^*, \ldots, x_n^*) \in \mathbb{R}^n$, approssimato da x si definisce

1. *errore assoluto tra x e x^* la quantità $\|x - x^*\|$ dove*
   - *$\|\cdot\|$ è una norma (cf. [24]); ad esempio $\|\cdot\|$ puó essere la cosidetta norma 2 o euclidea, ovvero se $y = (y_1, \ldots, y_n)$ allora $\|y\|_2 = \sqrt{\sum_{i=1}^n y_i^2}$,*
   - *se $x = (x_1, \ldots, x_n)$ allora $x - x^* = (x_1 - x_1^*, \ldots, x_n - x_n^*)$.*
2. *errore relativo tra x e $x^* \neq 0$ la quantità*

$$\|x - x^*\| / \|x^*\|.$$

Osservazione

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 $|\cdot|$.

Proposito.

Intendiamo calcolare l'errore relativo tra un numero reale e il suo floating point associato, nel caso si usi il troncamento o l'arrotondamento.

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

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

e $x = fl(x^*)$ la sua approssimazione in virgola mobile, ovvero

$$\begin{aligned} fl(x^*) &= \text{sgn}(x^*)(0.d_1, \ldots, d_{t-1}, \overline{d_t})\beta^e \\ &= \text{sgn}(x^*)\beta^e\Big(\sum_{k=1}^{t-1} d_k \beta^{-k} + \overline{d_t}\beta^{-t}\Big), \quad d_1 > 0. \end{aligned} \tag{7}$$

Visto che $d_1 \geq 1$, $d_k \geq 0$, $\beta > 0$, abbiamo

$$\sum_{k=1}^{+\infty} d_k \beta^{-k} = d_1\beta^{-1} + d_2\beta^{-2} + \ldots \geq d_1\beta^{-1} \geq \beta^{-1}$$

e quindi visto che i numeri macchina sono normalizzati

$$|x^*| = \beta^e \sum_{k=1}^{+\infty} d_k \beta^{-k} \geq \beta^e \cdot \beta^{-1} = \beta^{e-1}.$$

- Da
$$|x^*| \geq \beta^{e-1}$$
ricaviamo immediatamente
$$\frac{1}{|x^*|} \leq \frac{1}{\beta^{e-1}};$$
- osserviamo che $\sum_{k=1}^{+\infty} d_k \beta^{-k} - \sum_{k=1}^{t} d_k \beta^{-k} = \sum_{k=t+1}^{+\infty} d_k \beta^{-k}$.

Quindi essendo $|\mathrm{sgn}(x^*)| = 1$,

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

Nel caso si effettui il troncamento da

1. $\frac{|x^* - fl(x^*)|}{|x^*|} \leq \beta \cdot \left|\sum_{k=t}^{+\infty} d_k \beta^{-k} - \overline{d_t}\beta^{-t}\right|$ (ultima disuguaglianza),
2. $d_t = \overline{d_t}$ (troncamento!),
3. $\sum_{k=t}^{+\infty} d_k \beta^{-k} - d_t \beta^{-t} = (d_t \beta^{-t} + \sum_{k=t+1}^{+\infty} d_k \beta^{-k}) - d_t \beta^{-t} = \sum_{k=t+1}^{+\infty} d_k \beta^{-k}$,
4. $\sum_{k=t+1}^{+\infty} d_k \beta^{-k} = \beta^{-t} \sum_{k=t+1}^{+\infty} d_k \beta^{t-k} = \beta^{-t} \sum_{j=1}^{+\infty} d_{t+j} \beta^{-j} \leq \beta^{-t}$ (dopo qualche conto, vedi nota successiva),
5. $\mathtt{eps} = \beta^{1-t}$,

si ha

$$\begin{aligned} \frac{|x^* - fl(x^*)|}{|x^*|} &\leq \beta \cdot \left|\sum_{k=t}^{+\infty} d_k \beta^{-k} - \overline{d_t}\beta^{-t}\right| = \beta \cdot \left|\sum_{k=t}^{+\infty} d_k \beta^{-k} - d_t \beta^{-t}\right| \\ &= \beta \sum_{k=t+1}^{+\infty} d_k \beta^{-k} \leq \beta \cdot \beta^{-t} = \beta^{1-t} = \mathtt{eps} \end{aligned}$$

da cui

$$\frac{|x^* - fl(x^*)|}{|x^*|} \leq \mathtt{eps}.$$

Da

$$\frac{|x^* - fl(x^*)|}{|x^*|} \leq \beta \cdot \left| \sum_{k=t}^{+\infty} d_k \beta^{-k} - \overline{d}_t \beta^{-t} \right|,$$

nel caso si effettui l'arrotondamento abbiamo dopo alcuni calcoli che

$$\frac{|x^* - fl(x^*)|}{|x^*|} \leq \frac{\beta^{1-t}}{2} = \frac{\texttt{eps}}{2}.$$

La quantità eps/2 è detta unità di arrotondamento ed è ovviamente inferiore alla precisione di macchina eps.

Osservazione

Da queste stime sull'errore relativo compiuto nell'approssimare x con $fl(x)$ via arrotondamento e troncamento, si capisce perchè il primo risulti il sistema sia più utilizzato.

Osservazione (Facoltativa)

Uno dei punti chiave é che

$$\sum_{k=t+1}^{+\infty} d_k \beta^{-k} \leq \beta^{-t}$$

In effetti,

$$\begin{aligned} \sum_{k=t+1}^{+\infty} d_k \beta^{-k} &= \beta^{-t} \sum_{s=1}^{+\infty} d_{s+1} \beta^{-s} = \beta^{-t} (0.d_{s+1} d_{s+2} \ldots)_\beta \\ &\leq \beta^{-t} \end{aligned} \tag{8}$$

in quanto $(0.d_{s+1}d_{s+2}\ldots)_\beta$ *é una mantissa e quindi é una quantitá in* $(0,1)$.

Indicato con $\mathrm{fl}(x)$ il numero macchina che corrisponde a "x", denotiamo con

$$x \oplus y := \mathrm{fl}(\mathrm{fl}(x) + \mathrm{fl}(y)) \tag{9}$$
$$x \ominus y := \mathrm{fl}(\mathrm{fl}(x) - \mathrm{fl}(y)) \tag{10}$$
$$x \otimes y := \mathrm{fl}(\mathrm{fl}(x) \cdot \mathrm{fl}(y)) \tag{11}$$
$$x \oslash y := \mathrm{fl}(\mathrm{fl}(x)/\mathrm{fl}(y)) \tag{12}$$

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

$$\epsilon_x := \frac{|x - \mathrm{fl}(x)|}{|x|}, \quad \epsilon^{\odot}_{x,y} := \frac{|(x \,\mathrm{op}\, y) - (x \odot y)|}{|x \,\mathrm{op}\, y|}$$

Osservazione

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

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

Proprietá 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,
- non valgono quella associativa e 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.

Mostriamo come non valga la proprietà associativa della somma.

Esempio

Consideriamo $F(10, 4, -10, +10)$ *con* fl *basato sul troncamento. Sia* $a = 2000$, $b = 2.5$, $c = 7.8$. *Allora non è vero che*

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

in quanto

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

e quindi $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$.

Proprietá commutativa, associativa e distributiva delle operazioni floating point: esempio

Mostriamo come non valga la proprietà distributiva.

Esempio

In $F(10, 6, -10, +10)$, *posti*

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

Allora non è vero che $a \otimes (b \ominus c) = (a \otimes b) \ominus (a \otimes c)$ *in quanto*

$$\begin{aligned} a \otimes (b \ominus c) &= 0.800000 \cdot 10^9 \otimes 0.1 \cdot 10^{-2} = 0.800000 \cdot 10^6 \\ (a \otimes b) \ominus (a \otimes c) &= (0.800000 \cdot 10^9 \otimes 0.500009 \cdot 10^4) \\ &\quad \ominus (0.800000 \cdot 10^9 \otimes 0.500008 \cdot 10^4) \\ &\rightarrow \quad NaN \end{aligned}$$

(il `NaN` *è dovuto al fatto che in* $F(10, 6, -10, +10)$, *si ha che* $0.8 \cdot 10^9 \otimes 0.500009 \cdot 10^4 \approx 0.8 \cdot 10^9 \otimes 0.500008 \cdot 10^4 \approx 0.4 \cdot 10^{13} > 10^{10}$ *sono overflow!).*

Teorema (cf. [5, p.78])

Valgono le seguenti stime

$$\epsilon^{\oplus}_{x,y} = \frac{|(x+y)-(x\oplus y)|}{|x+y|} \leq \left|\frac{x}{x+y}\right|\epsilon_x + \left|\frac{y}{x+y}\right|\epsilon_y, \quad x+y\neq 0 \tag{13}$$

$$\epsilon^{\ominus}_{x,y} = \frac{|(x-y)-(x\ominus y)|}{|x-y|} \leq \left|\frac{x}{x-y}\right|\epsilon_x + \left|\frac{y}{x-y}\right|\epsilon_y, \quad x-y\neq 0 \tag{14}$$

$$\epsilon^{\otimes}_{x,y} = \frac{|(x\cdot y)-(x\otimes y)|}{|x\cdot y|} \lessapprox \epsilon_x+\epsilon_y, \; x\cdot y\neq 0 \tag{15}$$

$$\epsilon^{\oslash}_{x,y} = \frac{|(x/y)-(x\oslash y)|}{|x/y|} \lessapprox \left|\epsilon_x-\epsilon_y\right|, \; y\neq 0, \; x/y\neq 0 \tag{16}$$

dove per ogni operazione floating point $\odot$ *relativamente alla generica operazione* $\mathtt{op}$ *(ovvero* $+$, $-$, $*$, $/$*)*

$$\begin{aligned}\epsilon_x &= |x-fl(x)|/|x|,\\ \epsilon_y &= |y-fl(y)|/|y|,\\ \epsilon^{\odot}_{x,y} &= |(x\odot y)-(x \;\mathtt{op}\; y)|/|x \;\mathtt{op}\; y|\end{aligned}$$

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

Proposizione.

Se

1. $x + y \neq 0$, $x \neq 0$, $y \neq 0$,
2. $fl(fl(x) + fl(y)) = fl(x) + fl(y)$,
3. $\epsilon_x = |x - fl(x)|/|x|$,
4. $\epsilon_y = |y - fl(y)|/|y|$,
5. $\epsilon^{\oplus}_{x,y} = |(x \oplus y) - (x + y)|/|x + y|$

allora

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

Dimostrazione.

Nelle ipotesi precedenti, visto che

- $\epsilon_x = |x - fl(x)|/|x|$,
- $\epsilon_y = |y - fl(y)|/|y|$,
- $\mathrm{fl}(\mathrm{fl}(x) + \mathrm{fl}(y)) = \mathrm{fl}(x) + \mathrm{fl}(y)$,
- $|a + b| \leq |a| + |b|$, $a, b, \in \mathbb{R}$,

$$
\begin{aligned}
\epsilon^{\oplus}_{x,y} &= \frac{|(x+y) - (x \oplus y)|}{|x+y|} = \frac{|(x+y) - \mathrm{fl}(\mathrm{fl}(x) + \mathrm{fl}(y))|}{|x+y|} \\
&= \frac{|(x+y) - (\mathrm{fl}(x) + \mathrm{fl}(y))|}{|x+y|} = \frac{|(x - \mathrm{fl}(x)) + (y - \mathrm{fl}(y))|}{|x+y|} \\
&\leq \frac{|x - \mathrm{fl}(x)| + |y - \mathrm{fl}(y)|}{|x+y|} = \frac{|x - \mathrm{fl}(x)|}{|x+y|} + \frac{|y - \mathrm{fl}(y)|}{|x+y|} \\
&= \frac{|x|}{|x+y|} \cdot \frac{|x - \mathrm{fl}(x)|}{|x|} + \frac{|y|}{|x+y|} \cdot \frac{|y - \mathrm{fl}(y)|}{|y|} = \left|\frac{x}{x+y}\right| \epsilon_x + \left|\frac{y}{x+y}\right| \epsilon_y
\end{aligned}
$$

△

Nota. (Cancellazione)

Si noti che per $x + y \approx 0$ da

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

abbiamo che per piccoli errori sui dati ϵ_x, ϵ_y possono aversi errori rilevanti in $\epsilon_{x,y}^{\oplus}$ (fenomeno noto come cancellazione).

Nota. (Facoltativa e tecnica)

Nel precedente asserto, abbiamo supposto che $fl(fl(x) + fl(y)) = fl(x) + fl(y)$. Richiesto che si effettui l'arrotondamento per determinare l'approssimante floating-point, per qualche $\delta \in [0, \mathtt{eps}/2)$,

$$fl(fl(x) + fl(y)) = (fl(x) + fl(y))(1 + \delta)$$

ma il risultato che otteniamo, a parte di aumentare la difficoltà nei calcoli, è essenzialmente lo stesso.

Proposizione.

Se,

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

allora

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

Facoltativa.

Nelle ipotesi precedenti, visto che

- $\epsilon_x = |x - fl(x)|/|x|$,
- $\epsilon_y = |y - fl(y)|/|y|$,

abbiamo, visto che $\mathrm{fl}(\mathrm{fl}(x) - \mathrm{fl}(y)) = \mathrm{fl}(x) - \mathrm{fl}(y)$,

$$\begin{aligned}
\epsilon_{x,y}^{\ominus} &= \frac{|(x-y) - (x \ominus y)|}{|x-y|} = \frac{|(x-y) - \mathrm{fl}(\mathrm{fl}(x) - \mathrm{fl}(y))|}{|x-y|} \\
&= \frac{|(x - \mathrm{fl}(x)) - (y - \mathrm{fl}(y))|}{|x-y|} \leq \frac{|x|}{|x-y|} \cdot \frac{|x - \mathrm{fl}(x)|}{|x|} + \frac{|y|}{|x-y|} \cdot \frac{|y - \mathrm{fl}(y)|}{|y|} \\
&= \left|\frac{x}{x-y}\right| \epsilon_x + \left|\frac{y}{x-y}\right| \epsilon_y \qquad (17)
\end{aligned}$$

△

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

Esempio (Cancellazione)

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

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

Facilmente,

- $fl(a) = 0.73416, \qquad fl(b) = 0.73402$
- $a - b = 0.00013426, \qquad a \ominus b = 0.00014;$
- $\epsilon_a \approx 3.051115 \cdot 10^{-6}$, $\epsilon_b \approx 4.768240 \cdot 10^{-6}$,
- $\frac{|a|}{|a-b|} \approx 5468.1793534956$, $\frac{|b|}{|a-b|} \approx 5467.1793534956$,

Si verifica numericamente che $\epsilon^{\ominus}_{a,b} = |a|/|a-b|\epsilon_a + |b|/|a-b|\epsilon_b$ *e che l'errore*

$$\epsilon^{\ominus}_{a,b} = \frac{|(a-b) - (a \ominus b)|}{|a-b|} \approx 0.0427528675710549$$

e' notevole rispetto ϵ_a, ϵ_b, *a causa dei rilevanti fattori di amplificazione* $\frac{|a|}{|a-b|}$, $\frac{|b|}{|a-b|}$.

Proposizione.

Se

1. $x \neq 0$, $y \neq 0$, $|fl(y)|/|y| \approx 1$,
2. $fl(fl(x) \cdot fl(y)) = fl(x) \cdot fl(y)$,

posto

- $\epsilon_x = \frac{|x - fl(x)|}{|x|}$,
- $\epsilon_y = \frac{|y - fl(y)|}{|y|}$,
- $\epsilon^{\otimes}_{x,y} = |x \otimes y - x \cdot y|/|x \cdot y|$

allora

$$\epsilon^{\otimes}_{x,y} \lessapprox \epsilon_x + \epsilon_y.$$

dove $\lessapprox$ *significa inferiore o circa uguale.*

Dimostrazione.

Nelle ipotesi precedenti abbiamo

$$
\begin{aligned}
\epsilon_{x,y}^{\otimes} &= \frac{|x \cdot y - x \otimes y|}{|x \cdot y|} = \frac{|x \cdot y - \mathrm{fl}(\mathrm{fl}(x) \cdot \mathrm{fl}(y))|}{|x \cdot y|} = \frac{|x \cdot y - \mathrm{fl}(x) \cdot \mathrm{fl}(y)|}{|x \cdot y|} \\
&= \frac{|x \cdot y - x \cdot \mathrm{fl}(y) + x \cdot \mathrm{fl}(y) - \mathrm{fl}(x) \cdot \mathrm{fl}(y)|}{|x \cdot y|} \\
&= \frac{|x \cdot (y - \mathrm{fl}(y)) + \mathrm{fl}(y) \cdot (x - \mathrm{fl}(x))|}{|x \cdot y|} \\
&\leq \frac{|x \cdot (y - \mathrm{fl}(y))| + |\mathrm{fl}(y) \cdot (x - \mathrm{fl}(x))|}{|x \cdot y|} \\
&= \frac{|x \cdot (y - \mathrm{fl}(y))|}{|x \cdot y|} + \frac{|\mathrm{fl}(y) \cdot (x - \mathrm{fl}(x))|}{|x \cdot y|} \\
&= \underbrace{\frac{|y - \mathrm{fl}(y)|}{|y|}}_{\epsilon_y} + \underbrace{\frac{|\mathrm{fl}(y)|}{|y|}}_{\approx 1} \cdot \underbrace{\frac{|x - \mathrm{fl}(x)|}{|x|}}_{\epsilon_x} \approx \epsilon_x + \epsilon_y.
\end{aligned}
$$

△

Facoltativo.

Per quanto riguarda la divisione x/y, forniamo una traccia. Basta

- *studiare il caso* $1/y$,
- *ricordare i risultati del prodotto* $x \cdot \frac{1}{y}$.

per ottenere nelle ipotesi precedenti

$$\epsilon^{\oslash}_{x,y} = \frac{|(x/y) - (x \oslash y)|}{|x/y|} \lessapprox \left|\epsilon_x - \epsilon_y\right|, \ y \neq 0, \ x/y \neq 0.$$

Osservazione (Difficile, necessario aver seguito il corso di Analisi II)

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

$$f(x,y) \approx f(\overline{x},\overline{y}) + f'_x(\overline{x},\overline{y})(x-\overline{x}) + f'_y(\overline{x},\overline{y})(y-\overline{y}) \tag{18}$$

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*

$$\begin{aligned} \frac{|f(x,y)-f(\overline{x},\overline{y})|}{|f(x,y)|} &\lessapprox \frac{|f'_x(\overline{x},\overline{y})||x-\overline{x}||x|}{|f(x,y)||x|} + \frac{|f'_y(\overline{x},\overline{y})||y-\overline{y}||y|}{|f(x,y)||y|} \\ &= \frac{|f'_x(\overline{x},\overline{y})||x|}{|f(x,y)|}\cdot\epsilon_x + \frac{|f'_y(\overline{x},\overline{y})||y|}{|f(x,y)|}\cdot\epsilon_y \end{aligned} \tag{19}$$

Da (19), 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* $exp(x+y)$.

Si consideri un problema scientifico che abbia a che fare con una procedura di calcolo. Sussitono 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).

  Errori di questo tipo hanno causato nel 1996 la perdita del razzo Ariane 5 per una cattiva conversione di un numero, 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 velocitá della luce nel vuoto è

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

  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].

Nel valutare una funzione continua

$$f : \Omega \subseteq \mathbb{R} \to \mathbb{R}$$

in un punto $x \in \Omega$ bisogna tener conto che il dato x può essere per varie ragioni affetto da errori

- misura e/o
- di rappresentazione al calcolatore,

così da essere approssimato con un certo x_C.

Di conseguenza, invece di $f(x)$ si valuta $f(x_C)$ e ci si domanda se ci siano considerevoli differenze nei risultati.

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

$$\epsilon_{(x,x_C)} = \frac{|x - x_c|}{|x|}$$

corrispondano grandi errori (relativi) sui risultati

$$\epsilon_{(f(x),f(x_C))} = \frac{|f(x) - f(x_c)|}{|f(x)|}.$$

Quindi è importante discutere la quantità

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

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 (o di Lagrange) abbiamo

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

e quindi da

$$\frac{|f(x) - f(x_c)|}{|x - x_c|} = \frac{|f'(\xi) \cdot (x - x_c)|}{|x - x_c|} = \frac{|f'(\xi)| \cdot |x - x_c|}{|x - x_c|} = |f'(\xi)| \approx |f'(x)|$$

ricaviamo per

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

che

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

- La quantità

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

  si chiama condizionamento di f nel punto x.
- In virtu' della definizione di $\mathcal{K}(f, x, x_C)$, essendo

$$\mathcal{K}(f, x) \approx \mathcal{K}(f, x, x_C),$$

  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 $x_2 \in \Omega$.
- 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

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

da cui per $x \in (-1, 1)$*, essendo* $|x|^2 = x^2$*,*

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

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

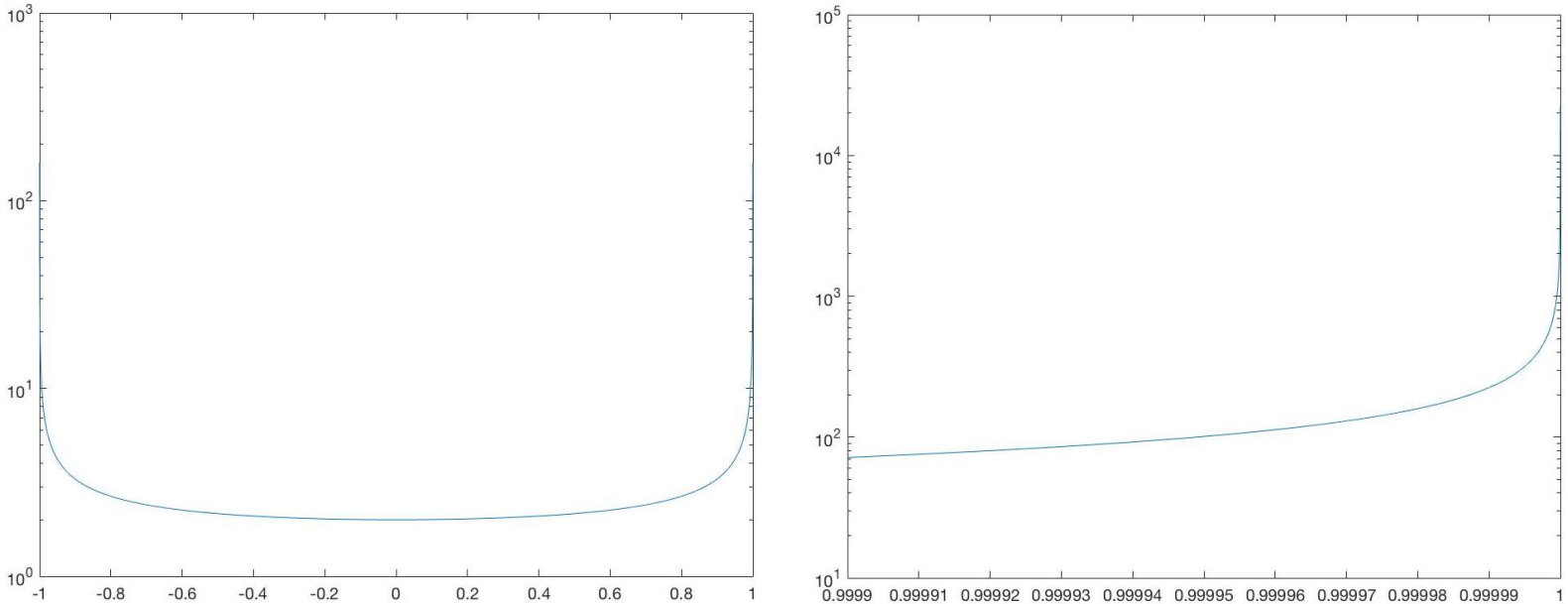

Figura: A sinistra il grafico di $\mathcal{K}(f_1, x)$ in $(-1, 1)$, a destra quello di $\mathcal{K}(f_1, x)$ in $(1 - 10^{-4}, 1)$ (malcondizionamento per $|x| \approx 1$).

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, 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.*

Valutazione di una funzione: esempio 2

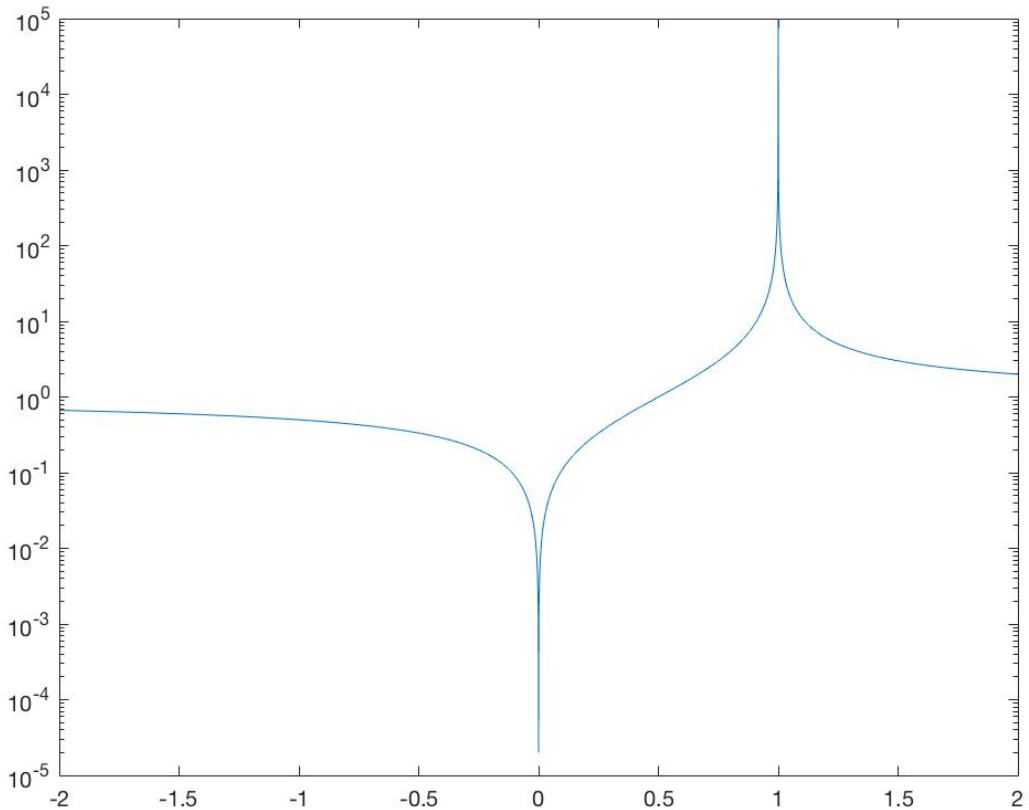

Figura: Il grafico di $\mathcal{K}(f_2, x)$ in $(-2, 2)$ (malcondizionamento per $x \approx 1$).

Nell'analisi del condizionamento $\mathcal{K}(f, x)$ di una funzione continua f in un punto x del suo dominio, abbiamo supposto che f sia valutabile esattamente e quindi che $\mathcal{K}(f, x)$ non dipenda dall'algoritmo utilizzato.

Definizione (Algoritmo stabile)

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.

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.

Esempio

Dato il polinomio di secondo grado $x^2 + 2px - q$, con $q \geq 0$ (e quindi $p^2 + q \geq 0$), calcolare la radice maggiore

$$y = -p + \sqrt{p^2 + q}. \qquad (20)$$

Osserviamo che

- essendo $p^2 + q \geq 0$ le radici

$$y = -p \pm \sqrt{p^2 + q}. \qquad (21)$$

  dell'equazione sono reali;
- la soluzione descritta in (20) è la maggiore di quelle presenti in (21) poiché $\sqrt{p^2 + q} \geq 0$,
- è non negativa visto che se $q > 0$ allora $\sqrt{p^2 + q} > p$.

1. Si osserva subito che se $p \geq 0$ allora la valutazione $y = -p + \sqrt{p^2 + q}$ è potenzialmente instabile per $p \gg q$ a causa della sottrazione tra p e $\sqrt{p^2 + q} \approx |p|$.

   A tal proposito, dopo averla implementata in Matlab, verificheremo numericamente la perdita di accuratezza per opportune scelte dei coefficienti p e q.

2. Risultati migliori si ottengono con una formula alternativa (e stabile) ricavata razionalizzando $y = -p + \sqrt{p^2 + q}$:

$$\begin{aligned} y &= -p + \sqrt{p^2 + q} = \frac{(-p + \sqrt{p^2 + q})(p + \sqrt{p^2 + q})}{p + \sqrt{p^2 + q}} \\ &= \frac{-p^2 + (p^2 + q)}{p + \sqrt{p^2 + q}} = \frac{q}{p + \sqrt{p^2 + q}}. \end{aligned} \tag{22}$$

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 rispettivamente di

- algoritmo bencondizionato o stabile,
- algoritmo malcondizionato o instabile [5, p. 66].

Importante.

Seguendo [5, p. 10], [5, p. 78], indipendentemente dall'algoritmo utilizzato, il problema della determinazione della radice $y = -p + \sqrt{p^2 + q}$ *è*

- *bencondizionato per* $q > 0$,
- *malcondizionato per* $q \approx -p^2$.

Di conseguenza il problema che abbiamo considerato era bencondizionato (in quanto si supponeva $q > 0$*).*

Usando dei classici ragionamenti dell'analisi numerica si mostra che (cf. [14], p. 21, [6], p. 11)

1. il primo algoritmo (21) non è numericamente stabile qualora $p \gg q > 0$;
2. il secondo algoritmo (22) è numericamente stabile qualora $p \gg q > 0$.

nonostante in queste ipotesi il problema sia stabile.

Vediamo alcuni casi rilevanti, effettuati in doppia precisione.

1. In [14, 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	0.000008999999772772	$2.52 \cdot 10^{-9}$
2	0.000009000000000000	0

2. 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 $p \gg q$.

Lavorando in doppia precisione otteniamo

algoritmo	valore	err. rel.
1	0.00000010000007614493	$7.61 \cdot 10^{-6}$
2	0.00000010000000000000	0

Esempio

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 $\pi = 3,1415926535897932 38 \ldots$.

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 $m = 2, 3, \ldots, 40$ (che teoricamente dovrebbero approssimare π).

Esempio 2: Approssimazione di π

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

n	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

Esempio 2: Approssimazione di π

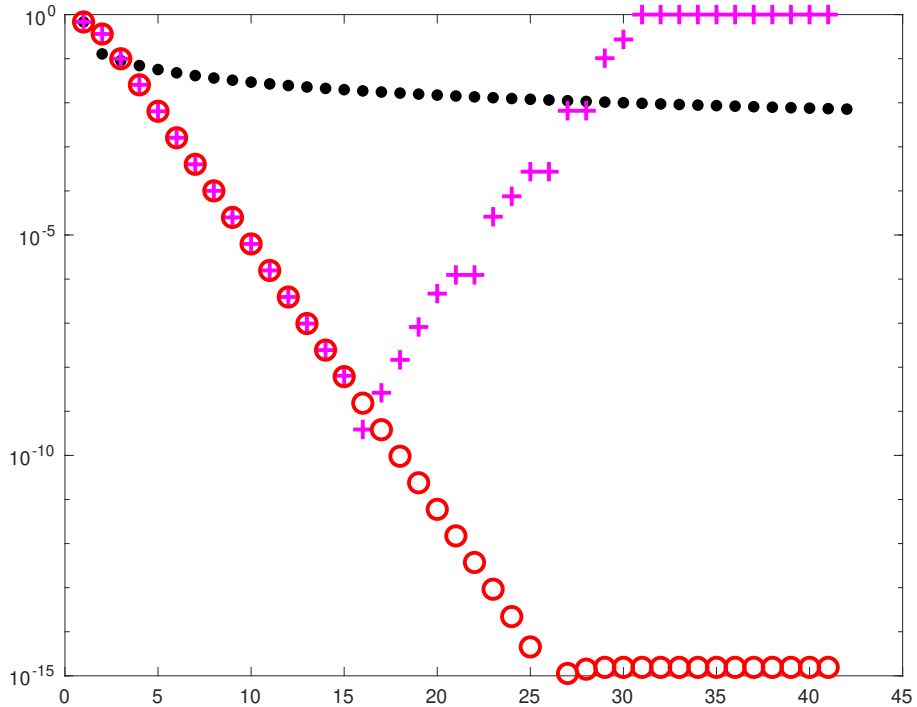

Figura: Errori relativi delle tre successioni u_n (puntino nero), z_n (punto magenta), y_n (cerchietto rosso). Risulta evidente che z_n diverge, nonostante fino a $n \approx 14$ dia circa gli stessi valori di y_n che invece converge.

Dalla figura e dalla tabella, consegue che

- la convergenza della prima successione è estremamente lenta, mentre quella della terza è più rapida;
- la seconda successione non converge numericamente, seppure converga teoricamente; il principale problema è la cancellazione che si ha nell'argomento della radice quadrata più esterna di

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

In tabella evidenziamo i valori assunti da $\sqrt{1-4^{1-n}\cdot z_n^2}$, per $n = 5, 10, 15, \ldots, 40$.

n	$\sqrt{1-4^{1-n}\cdot z_n^2}$	$\sqrt{1-\sqrt{1-4^{1-n}\cdot z_n^2}}$
5	$9.80785280403230541601544 70e-01$	$1.38617e-01$
10	$9.99811752826011090888869 1e-01$	$4.33875e-03$
15	$9.99999816164292232301136 2e-01$	$1.35586e-04$
20	$9.99999999820472496026013 7e-01$	$4.23707e-06$
25	$9.99999999999824584762109 2e-01$	$1.32444e-07$
30	$1.00000000000000000000000 0e+00$	$0.00000e+00$
35	$1.00000000000000000000000 0e+00$	$0.00000e+00$
40	$1.00000000000000000000000 0e+00$	$0.00000e+00$

L'ultima colonna evidenzia che per $n \geq 30$ si ha che $z_n = 0$, in quanto $z_{n+1} = 2^{n-\frac{1}{2}}\sqrt{1-\sqrt{1-4^{1-n}\cdot z_n^2}}$.

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.

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

Esempio

Si valuti numericamente la successione

$$I_n = e^{-1} \int_0^1 x^n e^x \, dx \qquad (23)$$

Proposizione.

La successione I_n è positiva, decrescente e infinitesima (cioè $\lim_n I_n = 0$) ed é

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

Esempio 3: Successione ricorrente

- Essendo
$$I_{n+1} = 1 - (n+1)\, I_n, \quad I_0 = 1 - e^{-1},$$
abbiamo che tutti i termini sono valutabili ricorsivamente, ad esempio
$$\begin{aligned} I_1 &= 1 - I_0 = 1 - (1 - e^{-1}) = e^{-1}, \\ I_2 &= 1 - 2 \cdot I_1 = 1 - 2 \cdot e^{-1}. \end{aligned}$$
- Portando I_{n+1} e $-(n+1)\, I_n$ risp. a primo e secondo membro, si ha
$$I_{n+1} = 1 - (n+1)\, I_n \Leftrightarrow (n+1)\, I_n = 1 - I_{n+1}$$
e dividendo ambo i membri di quest'ultima per $n+1$
$$I_n = \frac{1 - I_{n+1}}{n+1}$$
per cui visto che per $n^* \gg 0$ si ha $I_{n^*} \approx 0$, in quanto la successione è infinitesima, si può considerare la successione all'indietro

$$I_n = \frac{1 - I_{n+1}}{(n+1)}, I_{n^*} = 0, \ n = n^* - 1, \ldots, 1.$$

Riassumendo, se desideriamo calcolare $I_1, \ldots, I_{100}$:

1. Come primo tentativo calcoliamo I_n per $n = 1, \ldots, 100$ 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 $m = 500$, calcoliamo la successione all'indietro $\{t_n\}_{n=1,\ldots,500}$ definita come

$$\begin{cases} t_{500} = 0 \\ t_{n-1} = \frac{1-t_n}{n} \end{cases}$$

Si osservi che per raggiungere tale obiettivo bisogna calcolare i termini

$$t_{500},\ t_{499}, \ldots, t_{100}, t_{99}, \ldots, t_2, t_1.$$

Dopo aver eseguito i calcoli, otteniamo per $n = 10, 20, \ldots, 100$ la seguente tabella in cui si evince che in effetti

- s_n a differenza di I_n non è infinitesima, positiva e decrescente,
- t_n fornisce risultati compatibili con le proprietá della successione.

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.

n	s_n	t_n
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	$-1.159928542966359e+141$	$9.804855005376052e-03$

Osservazione (Facoltativa)

In effetti anche se operassimo in aritmetica esatta, supponiamo

- *successione esatta:*

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

- *successione valutata esattamente ma con errore iniziale:*

$$I^*_{n+1} = 1 - (n+1)I^*_n,\ I^*_0 = I_0 + \epsilon = (1 - e^{-1}) + \epsilon,$$

allora si vede che

$$I^*_n - I_n = n!\epsilon$$

in quanto

- *l'asserto* $I^*_n - I_n = n!\epsilon$ *vale per* $n = 0$, *visto che* $I^*_0 - I_0 = 0!\epsilon = \epsilon$;
- *se supponiamo* $I^*_n - I_n = n!\epsilon$, *ovvero* $I^*_n = I_n - n!\epsilon$ *allora*

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

Esempio

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

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

con $\eta = 10^{-1}, 10^{-2}, \ldots, 10^{-15}$.

Ovviamente il risultato esatto di questa divisione è sempre 1, per qualsiasi η.

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, differentemente da quanto visto per funzioni f malcondizionate, sará dovuto all'algoritmo di calcolo.

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

- 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.
- Per $\eta \approx 0$, avremo problemi a calcolare correttamente il numeratore $(1 + \eta) - 1$, causa il fenomeno di cancellazione.

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

η	$f^*(\eta)$	rel.err.
$1.00e-01$	$1.000000000000001e+00$	$8.881784e-16$
$1.00e-02$	$1.000000000000001e+00$	$8.881784e-16$
$1.00e-03$	$9.999999999998899e-01$	$1.101341e-13$
$1.00e-04$	$9.999999999998899e-01$	$1.101341e-13$
$1.00e-05$	$1.000000000006551e+00$	$6.551204e-12$
$1.00e-06$	$9.999999999177334e-01$	$8.226664e-11$
$1.00e-07$	$1.000000000583867e+00$	$5.838672e-10$
$1.00e-08$	$9.999999939225290e-01$	$6.077471e-09$
$1.00e-09$	$1.000000082740371e+00$	$8.274037e-08$
$1.00e-10$	$1.000000082740371e+00$	$8.274037e-08$
$1.00e-11$	$1.000000082740371e+00$	$8.274037e-08$
$1.00e-12$	$1.000088900582341e+00$	$8.890058e-05$
$1.00e-13$	$9.992007221626409e-01$	$7.992778e-04$
$1.00e-14$	$9.992007221626409e-01$	$7.992778e-04$
$1.00e-15$	$1.110223024625157e+00$	$1.102230e-01$

Nota. (Facoltativa)

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 η in modulo di* `eps` *si ha $1+x=1$ non sorprende inoltre sia*

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

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

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

Esempio

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

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

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

Valutiamo la funzione $F = f \circ g$ numericamente, mediante la corrispettiva versione numerica F^* per

$$x = 10^k, \ \text{per } k = -20, -16, -12, -8, \ldots, 8, 12, 16, 20,$$

nonchè l'errore relativo `rel.err.` compiuto, ovvero

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

notando che $F(x) \neq 0$ visto che nei casi in questione si ha $x \neq 0$.

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

Descriviamo questi risultati nella tabella che segue.

x	$F^*(x)$	rel.err.
$1.0e-20$	$1.000000000000000e-20$	$0.00000e+00$
$1.0e-16$	$1.000000000000000e-16$	$0.00000e+00$
$1.0e-12$	$1.000000000000000e-12$	$0.00000e+00$
$1.0e-08$	$1.000000000000004e-08$	$0.00000e+00$
$1.0e-04$	$1.000000000000000e-04$	$0.00000e+00$
$1.0e+00$	$9.999999999999999e-01$	$1.11022e-16$
$1.0e+04$	$9.999999999990539e+03$	$9.46056e-13$
$1.0e+08$	$9.999999999542369e+07$	$4.57631e-11$
$1.0e+12$	$1.000071916854289e+12$	$7.19117e-05$
$1.0e+16$	$1.633123935319537e+16$	$3.87677e-01$
$1.0e+20$	$1.633123935319537e+16$	$6.12223e+03$

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

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

x	$F^*(x)$	rel.err.
$1.000000000000000e+00$	$9.999999999999999e-01$	$1.11022e-16$
$2.511886431509580e+00$	$2.511886431509580e+00$	$0.00000e+00$
$6.309573444801933e+00$	$6.309573444801929e+00$	$7.03834e-16$
$1.584893192461114e+01$	$1.584893192461114e+01$	$1.12081e-16$
$3.981071705534973e+01$	$3.981071705534988e+01$	$3.56961e-15$
$1.000000000000000e+02$	$1.000000000000010e+02$	$1.00897e-14$
$2.511886431509580e+02$	$2.511886431509614e+02$	$1.38042e-14$
$6.309573444801930e+02$	$6.309573444802187e+02$	$4.07210e-14$
$1.584893192461114e+03$	$1.584893192461186e+03$	$4.54778e-14$
$3.981071705534973e+03$	$3.981071705533795e+03$	$2.95849e-13$
$1.000000000000000e+04$	$9.999999999990539e+03$	$9.46056e-13$

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

- 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.
- Valuteremo alcuni algoritmi per la valutazione di polinomi in un punto, per la potenza di un numero e per il calcolo del determinante di una matrice.

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

$$p_4(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4.$$

1. Se si implementa un metodo brutale, necessitano
   - 4 somme;
   - 3 potenze, e visto che

$$\begin{aligned} x^2 &= x \cdot x, \\ x^3 &= x \cdot x \cdot x, \\ x^4 &= x \cdot x \cdot x \cdot x, \end{aligned}$$

     necessitano $1 + 2 + 3$ prodotti;
   - 4 prodotti del tipo $a_k \cdot x^k$.

Quindi, per $n = 4$ servono 4 operazioni additive e 10 moltiplicative.

Con questa tecnica in generale servono per valutare un polinomio di grado n

$$p_n(x) = a_0 + a_1 x + \ldots + a_n x^n.$$

nel generico punto x

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

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

In effetti nell'esempio precedente, per $\overline{n} = 4$ facevamo 10 operazioni moltiplicative e in questo caso

- $\frac{(\overline{n}+1)\cdot\overline{n}}{2} = \frac{5\cdot 4}{2} = 10$;
- $\frac{\overline{n}^2}{2} = \frac{16}{2} = 8$.

Facendo un pó di attenzione è facile osservare che in realtà, per valutare p_4 servano

- 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.

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

$$\begin{aligned} p_4(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \\ &= a_0 + x(a_1 + a_2 x + a_3 x^2 + a_4 x^3) \\ &= a_0 + x(a_1 + x(a_2 + a_3 x + a_4 x^2)) \\ &= a_0 + x\,(a_1 + x\,(a_2 + x\,\underbrace{(a_3 + a_4 x)}_{b_3})) \\ &= a_0 + x\,(a_1 + x\,\underbrace{(a_2 + x\,(a_3 + a_4 x))}_{b_2}) \\ &= a_0 + x\,\underbrace{(a_1 + x\,(a_2 + x\,(a_3 + a_4 x)))}_{b_1} \\ &= \underbrace{a_0 + x\,(a_1 + x\,(a_2 + x\,(a_3 + a_4 x)))}_{b_0} \end{aligned}$$

e quindi *valutate le parentesi*, dalla piú interna alla piú esterna

- $b_3 := a_3 + a_4 x$,
- $b_2 := a_2 + b_3 x$,
- $b_1 := a_1 + b_2 x$,
- $b_0 := a_0 + b_1 x$,

otteniamo $p_4(x) = b_0$.

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

- le 4 somme e i 10 prodotti del primo algoritmo precedente,
- le 4 somme e i 7 prodotti del secondo algoritmo precedente.

Non è difficile vedere che in generale, con questa regola, usualmente detta di Ruffini-Horner [26], occorrono n operazioni moltiplicative per valutare il polinomio p di grado n nel generico punto x, quindi in numero inferiore rispetto

- ai $\frac{(n+1)n}{2}$ prodotti del primo algoritmo precedente,
- ai $2n-1$ prodotti del secondo algoritmo precedente.

Esempio

Valutare il polinomio $p_3(x) = 2x^3 + 4x^2 + 5x + 7$ nel punto $x = 2$, mediante l'algoritmo di Ruffini-Horner.

La valutazione diretta comporta che per $x = 2$

$$x^2 = 4, \quad x^3 = x^2 \cdot x = 4 \cdot 2 = 8.$$

Quindi

$$p_3(2) = 2 \cdot 2^3 + 4 \cdot 2^2 + 5 \cdot 2 + 7 = 16 + 16 + 10 + 7 = 49.$$

Se contiamo i prodotti eseguiti, sono 2 per le potenze e 3 per la valutazione del polinomio, ovvero 5 moltiplicazioni.

Valutiamo ora il polinomio $p_3(x) = 2x^3 + 4x^2 + 5x + 7$ nel punto $x = 2$, mediante l'algoritmo di Ruffini-Horner.

Essendo $a_0 = 7$, $a_1 = 5$, $a_2 = 4$, $a_3 = 2$, valutiamo in sequenza:

- $b_2 := a_2 + a_3 x = 4 + 2 \cdot 2 = 8$,
- $b_1 := a_1 + b_2 x = 5 + 8 \cdot 2 = 21$,
- $b_0 := a_0 + b_1 x = 7 + 21 \cdot 2 = 49$,

raggiungendo il risultato con 3 moltiplicazioni (invece delle 5 del metodo diretto).

Esempio

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

1. Un metodo brutale, osservato che

$$a^n = a \cdot \ldots \cdot a$$

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

$$a^n = a^{\sum_{j=0}^{m} c_j 2^j} = \prod_{j=0}^{m} a^{c_j 2^j} = a^{c_0} \cdot (a^2)^{c_1} \cdot (a^{2^2})^{c_2} \ldots (a^{2^m})^{c_m}$$

Notiamo che $a^2 = a \cdot a$, $a^4 = a^2 \cdot a^2$, e in generale

$$a^{2^k} = a^{2^{k-1}} \cdot a^{2^{k-1}}$$

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

Calcolo delle potenza

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 (24) sono infine m.

Un codice che implementa la conversione di un numero naturale da forma decimale a binaria è

```
function c=dec2bin(n)
j=1; t=n;
while t > 0
    q=t/2; t=floor(q);
    if t == q
        c(j)=0;
    else
        c(j)=1;
    end
    j=j+1;
end
```

che richiede

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

Ne ricaviamo che, per $m = \log_2(n)$, calcolare la potenza richiesta serve al piú un numero di operazioni moltiplicative pari a

$$3m + 1 = 3\lfloor \log_2(n) \rfloor + 1$$

in quanto

- la valutazione dei coefficienti $c_0, \ldots, c_m$ richiede $m + 1$ divisioni;
- la valutazione delle potenze a^{2^k}, $k = 0, \ldots, m$ consta di m operazioni moltiplicative (notare che per $k = 0$ non serve fare calcoli);
- calcolate le potenze, la valutazione finale di (24), 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 (era $n-1$),
- $3m+1 = 3\lfloor \log_2(n) \rfloor + 1$,

Si vede che 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
$1.0e+02$	19
$1.0e+03$	28
$1.0e+04$	40
$1.0e+05$	49
$1.0e+06$	58
$1.0e+07$	70
$1.0e+08$	79
$1.0e+09$	88
$1.0e+10$	100

Osservazione (Facoltativa)

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

$$A^n = \underbrace{A \cdot \ldots \cdot A}_{n}$$

servono $n-1$ prodotti di matrici di dimensione "s" ovvero $(n-1)s^2$ moltiplicazioni. Se usiamo il secondo algoritmo

- *la valutazione dei coefficienti $c_0, \ldots, c_m$ richiede $m+1$ divisioni;*
- *la valutazione delle potenze a^{2^k}, $k = 0, \ldots, m$ 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 (24), 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.

Esempio

Valutare 3^{14}.

Metodo 1. Utilizzando il metodo diretto, mediante 13 prodotti abbiamo

$$3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 = 4782969$$

Metodo 2. Dalla rappresentazione in binario di 14

$$14 = 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3$$

abbiamo

$$3^{14} = (3^1)^0 \cdot (3^2)^1 \cdot (3^4)^1 \cdot (3^8)^1 = (3^2)^1 \cdot (3^4)^1 \cdot (3^8)^1.$$

Essendo

$$3^2 = 3 \cdot 3 = 9, \quad 3^4 = 3^2 \cdot 3^2 = 9 \cdot 9 = 81, \quad 3^8 = 3^4 \cdot 3^4 = 81 \cdot 81 = 6561$$

abbiamo

$$3^{14} = (3^2)^1 \cdot (3^4)^1 \cdot (3^8)^1 = 9 \cdot 81 \cdot 6561 = 4782969$$

Così per ottenere 3^{14} abbiamo eseguito solo 9 operazioni moltiplicative (di cui 5 prodotti e $4 = \lfloor \log_2(14) \rfloor + 1$ necessarie per la codifica di 14 in binario).

Esempio

Sia la matrice quadrata $A = (a_{i,j}) \in \mathbb{R}^{n\times n}$. Si calcoli il suo determinante $\det(A)$.

Sia

- $A_{ij}^{(n-1)} \in \mathbb{R}^{(n-1)\times(n-1)}$, la sottomatrice estratta da A cancellando la i-esima riga e la j-esima colonna.
- $\det(A_{ij}^{(n-1)})$, il minore complementare di indici (i, j);
- $(-1)^{i+j}\det(A_{ij}^{(n-1)})$, il complemento algebrico di indici (i, j)

Il primo teorema di Laplace afferma che

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

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

Se $A = LU$ come dalla fattorizzazione LU senza pivoting (cf. [19]), 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 $l_{k,k} = 1$, $k = 1, \ldots, n$, si ricava $\det(L) = 1$, e quindi

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

In definitiva, visto che

- $\det(A) = \prod_{k=1}^{n} u_{k,k}$ necessita solo di $n-1$ operazioni moltiplicative,
- la fattorizzazione $A = LU$ ha complessità dell'ordine di $n^3/3$,

allora 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 (il computer piú *veloce* del 2018) che eroga 200 peta-flops, 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 $13.82 \cdot 10^9$ anni (con un errore di 120 milioni di anni) e che attualmente il computer piú veloce al mondo ha una potenza che supera 2.4 exa-flops [29].

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: 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 $PA = LU$.

Esempio

Si valuti $f(x) = e^x$, per $x > 1$.

Metodo 1. Posto

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

abbiamo che, dal teorema di Lagrange,

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

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 $\xi > 1$.

Metodo 2. Per un numero naturale $m \geq x$ possiamo scrivere

$$e^x = (e^{x/m})^m$$

e quindi procedere valutando prima $a = e^{x/m}$ e poi eseguire a^m con l'algoritmo mostrato nell'esempio precedente.

Quindi valutiamo

$$S^{(2)}_{n,m}(x) = \left(\sum_{i=0}^{n} \frac{(x/m)^i}{i!} \right)^m \approx e^x. \tag{24}$$

Si noti che dal teorema di Lagrange,

$$E^{(2)}_{n,m} := \left| \exp(x/m) - \sum_{i=0}^{n} \frac{(x/m)^i}{i!} \right| = \frac{\xi^{n+1}}{(n+1)!}, \quad \xi \in [0,1]$$

che è molto favorevole rispetto alla stima del metodo precedente.

Valutazione dell'esponenziale (facoltativo)

Quale esempio calcoliamo

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

con

- il primo algoritmo;
- 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 i rispettivi errori relativi degli algoritmi

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

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	$9.8e-01$	$6.6e-15$	$9.0e-15$	$2.7e-15$

Bibliografia I

- D. Arnold, http://www.ima.umn.edu/ arnold/disasters/
- K. Atkinson, *An Introduction to Numerical Analysis*, Wiley, (1989).
- K. Atkinson and W. Han, *Elementary Numerical Analysis*, third edition, Wiley, (2004).
- N. Brisebarre and collaborators *Floating-Point Arithmetic*, 2009. https://perso.ens-lyon.fr/jean-michel.muller/chapitre1.pdf
- V. Comincioli, Analisi Numerica, metodi modelli applicazioni, Mc Graw-Hill, 1990.
- V. Comincioli, Problemi di analisi numerica, Mc Graw-Hill, 1991.
- M. Frontini e E. Sormani, Fondamenti di Calcolo Numerico, problemi in laboratorio, Apogeo, 2005.
- W. Gautschi, Numerical Analysis, second edition, Birkhäuser, 2012.
- D. Goldberg, What Every Computer Scientist Should Know About Floating-Point Arithmetic, Association for Computing Machinery, Inc, Computing Surveys, March 1991.

Bibliografia II

- T. Huckle, http://wwwzenger.informatik.tu-muenchen.de/persons/huckle/bugse.html.
- Mars Program, http://marsprogram.jpl.nasa.gov/msp98/orbiter
- J.H. Mathews e K.D. Fink, Numerical Methods using Matlab, Prentice Hall, 1999.
- Network Theory, GNU Octave Manual Version, http://www.network-theory.co.uk/docs/octave3/octave_160.html.
- J. Stoer, Introduzione all'analisi numerica, Zanichelli, 1984.
- The MathWorks Inc., Numerical Computing with Matlab,http://www.mathworks.com/moler.
- L.N. Trefethen, Numerical Analysis, http://people.maths.ox.ac.uk/trefethen/NAessay.pdf.
- Web Page , http://utenti.quipo.it/base5/numeri/pigreco.htm.

Bibliografia III

- Wikipedia (Archimede), http://it.wikipedia.org/wiki/Archimede
- Wikipedia (Decomposizione LU), https://it.wikipedia.org/wiki/Decomposizione_LU
- Wikipedia (Determinante), https://it.wikipedia.org/wiki/Determinante
- Wikipedia (Floating Point), http://it.wikipedia.org/wiki/Floating_point
- Wikipedia (IEEE 754), http://it.wikipedia.org/wiki/IEEE_754
- Wikipedia (IEEE 754r), http://it.wikipedia.org/wiki/IEEE_754r
- Wikipedia (Norma), https://it.wikipedia.org/wiki/Norma_(matematica)
- Wikipedia (Pi Greco), http://it.wikipedia.org/wiki/Pi_greco

Bibliografia IV

- Wikipedia (Regola di Horner), https://it.wikipedia.org/wiki/Regola_di_Horner
- Wikipedia (Teorema di Laplace), https://it.wikipedia.org/wiki/Teorema_di_Laplace.
- Wikipedia (Teorema Fondamentale del Calcolo Integrale), http://it.wikipedia.org/wiki/Teorema_fondamentale_del_calcolo_integrale
- Wikipedia (TOP500), https://en.wikipedia.org/wiki/TOP500.
- Wolfram (Archimedes' Algorithm), http://mathworld.wolfram.com/ArchimedesAlgorithm.html