Definizione di errore

Siano $\alpha \in \mathbb{R}$ e α^* una sua approssimazione.

- ERRORE ASSOLUTO: $E_a = |\alpha - \alpha^*|$
- ERRORE RELATIVO: $(\alpha \neq 0)$: $E_r = \dfrac{E_a}{|\alpha|}$
- ERRORE PERCENTUALE: $(\alpha \neq 0)$: $E_p = (E_r \times 100)\%$

ESEMPI.

$\alpha = 0.3 \cdot 10^1 \quad \alpha^* = 0.31 \cdot 10^1 \quad E_a = 0.1 \quad \begin{cases} E_r = 0.3333\ldots \cdot 10^{-1} \\ E_p = 3.33\ldots\% \end{cases}$

$\alpha = 0.3 \cdot 10^{-3} \quad \alpha^* = 0.31 \cdot 10^{-3} \quad E_a = 0.1 \cdot 10^{-4} \quad \begin{cases} E_r = 0.3333\ldots \cdot 10^{-1} \\ E_p = 3.33\ldots\% \end{cases}$

$\alpha = 0.3 \cdot 10^4 \quad \alpha^* = 0.31 \cdot 10^4 \quad E_a = 0.1 \cdot 10^3 \quad \begin{cases} E_r = 0.3333\ldots \cdot 10^{-1} \\ E_p = 3.33\ldots\% \end{cases}$

Tutti i numeri reali si possono approssimare con numeri aventi una rappresentazione finita.

Due tipi di approssimazione: troncamento e arrotondamento

Sia $\alpha \in \mathbb{R}$,$\alpha \neq 0$, $\alpha = \pm(\sum_{i=1}^{\infty} a_i \beta^{-i})\beta^p = \pm m\beta^p$.

- TRONCAMENTO di α alla t-esima cifra: $\alpha = 0.2345347 \cdot 10^{-3}$

$$\alpha_t = \pm \left(\sum_{i=1}^{t} a_i \beta^{-i} \right)_{\beta,t} \beta^p$$

TRONCAMENTO alla seconda cifra: $0.23 \cdot 10^{-3}$

- ARROTONDAMENTO di α alla t-esima cifra (β pari):

$$\alpha_{rr} = \pm \left(\sum_{i=1}^{\infty} a_i \beta^{-i} + \frac{1}{2}\beta^{-t} \right)_{\beta,t} \beta^p$$

ARROTONDAMENTO alla seconda cifra: $0.23 \cdot 10^{-3}$
ARROTONDAMENTO alla terza cifra: $0.235 \cdot 10^{-3}$

dove

$$\alpha_{rr} = \begin{cases} \alpha_t & \text{se } 0 \leqslant a_{t+1} < \beta/2 \\ & \text{round down} \\ \pm \left(\sum_{i=1}^{t-1} a_i \beta^{-i} + (a_t + 1)\beta^{-t} \right) \beta^p & \text{se } \beta/2 \leqslant a_{t+1} \leqslant \beta - 1 \\ & \text{round up} \end{cases}$$

Valutazione degli errori nel caso di troncamento

- TRONCAMENTO di $\alpha = \pm m\beta^p$ alla t-esima cifra:

$$\alpha_t = \pm\left(0.a_1 a_2 \dots a_t\right)_\beta \beta^p$$

$$\begin{aligned} E_a = |\alpha - \alpha_t| &= \left|0.a_1 a_2 \dots a_t a_{t+1} a_{t+2} \dots \ \beta^p - 0.a_1 a_2 \dots a_t \ \beta^p\right| \\ &= 0.\underbrace{000\dots0}_{t} a_{t+1} \dots \ \beta^p = 0.a_{t+1} a_{t+2} \dots \ \beta^{-t}\beta^p \end{aligned}$$

$0.a_{t+1}a_{t+2}\dots < 1 \ \Rightarrow \ E_a < \beta^{-t}\beta^p$.

$$E_r = \frac{|\alpha - \alpha_t|}{|\alpha|} = \frac{|\alpha - \alpha_t|}{m\beta^p} < \frac{\beta^p\beta^{-t}}{\beta^p\beta^{-1}} = \beta^{1-t}$$

$E_r < \beta^{1-t}$

perché la mantissa di α è $m \geqslant \dfrac{1}{\beta}$ e dunque $\dfrac{1}{m} \leqslant \beta$.

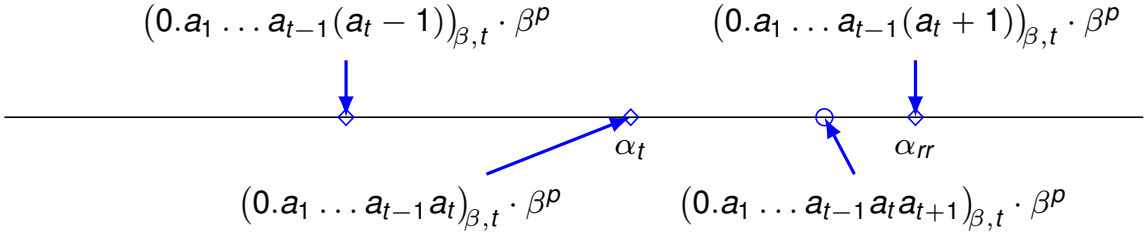
Valutazione degli errori nel caso di arrotondamento

- ARROTONDAMENTO di α alla t-esima cifra (β pari):

$$\alpha_{rr} = \pm\left(0.a_1 \dots a_t a_{t+1} \dots + \frac{1}{2}\beta^{-t}\right)_{\beta,t} \beta^p$$

$$= \begin{cases} \pm\left(0.a_1 \dots a_t\right)_{\beta,t} \cdot \beta^p & \text{se } 0 \leqslant a_{t+1} < \beta/2 \\ \pm\left(0.a_1 \dots (a_t+1)\right)_{\beta,t} \cdot \beta^p & \text{se } \beta/2 \leqslant a_{t+1} \leqslant \beta - 1 \end{cases}$$

Graficamente...
la situazione è questa: tra un simbolo romboidale e l'altro c'è una distanza di $\beta^{-t}\beta^p$:



$$E_a = |\alpha - \alpha_{rr}| \leqslant \frac{1}{2}\beta^{-t}\beta^p$$

$$E_r = \frac{|\alpha - \alpha_{rr}|}{|\alpha|} \leqslant \frac{1}{2}\frac{\beta^p\beta^{-t}}{\beta^p\beta^{-1}} = \frac{1}{2}\beta^{1-t}$$

Esempio

$\alpha = 10\pi = 31.41592654\ldots = 0.3141592654\ldots \cdot 10^2$

- mantissa: $0.3141592654\ldots$
- parte esponente: 10^2

Troncamento alla sesta cifra ($\beta = 10$, $t = 6$, $p = 2$): $\alpha \approx 0.314159 \cdot 10^2$

$$E_a = 0.2654\ldots \cdot 10^{-4} < 10^{-4} \qquad (10^{-t+p})$$

$$E_r = \frac{0.2654\ldots \cdot 10^{-4}}{\alpha} = 0.84\ldots \cdot 10^{-6} < 10^{-5} \qquad (10^{1-t})$$

Arrotondamento alla sesta cifra ($\beta = 10$, $t = 6$, $p = 2$): $\alpha \approx 0.314159 \cdot 10^2$

$$E_a = 0.2654\ldots \cdot 10^{-4} < 0.5 \cdot 10^{-4} \qquad \left(\frac{1}{2}10^{-t+p}\right)$$

$$E_r = \frac{0.2654\ldots \cdot 10^{-4}}{\alpha} = 0.84\ldots \cdot 10^{-6} < 0.5 \cdot 10^{-5} \qquad \left(\frac{1}{2}10^{1-t}\right)$$

Arrotondamento alla settima cifra ($\beta = 10$, $t = 7$, $p = 2$): $\pi \approx 0.3141593 \cdot 10^2$

$$E_a = 0.346\ldots \cdot 10^{-5} < 0.5 \cdot 10^{-5} \qquad \left(\frac{1}{2}10^{-t+p}\right)$$

$$E_r = \frac{0.346\ldots \cdot 10^{-5}}{\alpha} = 0.11\ldots \cdot 10^{-6} < 0.5 \cdot 10^{-6} \qquad \left(\frac{1}{2}10^{1-t}\right)$$

3.141593 è un'approssimazione di π con 7 cifre significative.

Numeri finiti o numeri di macchina

A causa della limitata lunghezza della parola di memoria, sono rappresentabili effettivamente su calcolatore:

- un intervallo limitato di interi (a virgola fissa o *fixed point numbers*);
- un insieme finito di numeri razionali (numeri a virgola mobile o *floating point numbers*).

Numeri a virgola fissa (β, t)

Fissiamo un intero $\beta > 1$ e sia $t+1$ il numero di cifre a disposizione per la rappresentazione di un intero:

- β si usa come base di rappresentazione;
- t cifre si usano per la rappresentazione del valore assoluto del numero e 1 cifra si usa per l'informazione sul segno.

Si indica con $\mathrm{fi}(N)$ **la rappresentazione *fixed point* del numero** N**.**

Rappresentazione di un intero N non negativo.
Sia $N = (d_p d_{p-1} \dots d_1)_\beta$ la rappresentazione di N in base β.

$$\mathrm{fi}(N) = \begin{cases} (\underbrace{000\dots00}_{t-p \text{ zeri}}\, d_p d_{p-1} \dots d_1)_{\beta,t} & \text{se } t \geqslant p \quad \Rightarrow \quad \mathrm{fi}(N) = N \\ (d_t d_{t-1} \dots d_1)_{\beta,t} & \text{se } t < p \quad \Rightarrow \quad \mathrm{fi}(N) \neq N \end{cases}$$

Allora $\mathrm{fi}(N) = N$ se $t \geqslant p$, altrimenti $\mathrm{fi}(N) \neq N$.
In questo caso si verifica un OVERFLOW intero, che non arresta la macchina.
Sono rappresentabili solo le t **cifre meno significative.**
Esempio. Per $\beta = 2$ e $t+1 = 16$

$N = 1235 = (10011010011)_2$ $\quad$ $\mathrm{fi}(N) = (0\ 0000010011010011)$
$N = 70231 = (10001001001010111)_2$ $\quad$ $\mathrm{fi}(N) = (0\ 0010010010101 11) = (4695)_{10}$
si cancella 10: **OVERFLOW!**

Numeri a virgola vissa (β, t)

Il più grande intero rappresentabile esattamente è dato da:

$$0ccc...ccc = (\beta-1)\beta^{t-1} + \dots + (\beta-1)\beta^0 = \beta^t - 1$$

ove $c = \beta - 1$.
Il punto radice è omesso poiché è pensato a destra della cifra meno significativa in posizione fissa (da cui fixed point).
Sono rappresentabili esattamente solo gli interi non negativi compresi tra $[0, \beta^t - 1]$**.**

Esempio. Per $\beta = 2$ e $t+1 = 16$, il massimo intero rappresentabile esattamente vale

$$(0111111111111111)_2 = 2^{15} - 1 = 32767$$

Numeri a virgola vissa (β, t)

Rappresentazione di un intero N negativo.

$$\text{fi}(N) = (\beta^{t+1} - |N|)_{t+1}$$

Questa si dice **rappresentazione complemento alla base β in $t+1$ cifre**.

Per ottenere $\text{fi}(N)$, si prende il valore assoluto di N rappresentato in $t+1$ cifre e poi si complementa alla base la cifra meno significativa diversa da 0 e a $\beta - 1$ le altre cifre più significative. Pertanto la cifra $(t+1)$-esima è uguale a $c = \beta - 1$.

Esempio. Con $\beta = 10$, $t+1 = 5$, $N = -9320$ diventa

$$\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ & 0 & 9 & 3 & 2 & 0 \\ \hline & 9 & 0 & 6 & 8 & 0 \end{array}$$

$\text{fi}(-9320) = 90680$.

In base 2, il complemento alla base in $t+1$ cifre si ottiene ponendo 1 in corrispondenza della cifra meno significativa diversa da 0 e poi scambiando 1 con 0 e 0 con 1 per le cifre più significative.
Un'altra regola è di scambiare 0 con 1 e 1 con 0 (toggle) e poi di aggiungere 1.

Esempi

- $\beta = 2, t+1 = 16$;

$$\begin{aligned} \text{fi}((1235)_{10}) &= \text{fi}((10011010011)_2) = 0000010011010011 \\ \text{fi}(-(1235)_{10}) &= \text{fi}(-(10011010011)_2) = 1111101100101100 + 1 \\ &= 1111101100101101 \\ \text{fi}(-1) &= 1111111111111111 \\ \text{fi}(-8) &= \text{fi}(-(1000)_2) = 1111111111110111 + 1 \\ &= 1111111111111000 \end{aligned}$$

Se un numero è negativo, il bit $(t+1)$-esimo vale sempre 1!!!

Gli estremi rappresentabili

Il più piccolo intero esattamente rappresentabile è:

$$c000...00 = \text{fi}(-1000...00) = -\beta^t$$

Infatti $\beta^{t+1} - |\alpha| = (\beta - 1)\beta^t \Rightarrow |\alpha| = \beta^{t+1} - (\beta - 1)\beta^t = (\beta - \beta + 1)\beta^t$. Dunque $\alpha = -\beta^t$.

Sono esattamente rappresentabili $2\beta^t$ interi, appartenenti all'intervallo $[-\beta^t, \beta^t - 1]$. Al di fuori dell'intervallo si incorre nell'overflow intero.

- $\beta = 2, t + 1 = 32$; sono rappresentabili esattamente gli interi in $[-2^{31}, 2^{31} - 1]$.
- $\beta = 2, t + 1 = 16$; sono rappresentabili gli interi in $[-32768, 32767]$.

Osservazione

Per $\beta = 2$, la cifra $(t + 1)$-esima permette di stabilire in modo veloce il segno di un numero: se il bit $t + 1$ vale 0 il numero è positivo o nullo, altrimenti è negativo.

Insieme dei numeri fixed point interi rappresentabili con $\beta = 2, t + 1 = 4$

Ci sono $2^4 = 16$ possibili configurazioni.

fi(N)	N
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Esercizio

Sia $\beta = 2$ e $t + 1 = 16$. Rappresentare in fixed point i numeri 1023 e -31128.

$$\begin{aligned} \text{fi}(1023) &= \text{fi}\big((1111111111)_2\big) = 0000001111111111 \\ \text{fi}(-(31128)_{10}) &= \text{fi}\big(-(1111001100011000)_2\big) \\ &= 1000011001101000 \end{aligned}$$

Quando la base è 2, valgono le seguenti proprietà:

$$\begin{aligned} \text{fi}(2^t - 1) + 1 &= 011\ldots111 + 1 \\ &= 100\ldots000 \\ &= \text{fi}(-2^t) \\ \text{fi}(-2^t) - 1 &= 100\ldots000 - 1 \\ &= 0111\ldots111 \\ &= \text{fi}(2^t - 1) \end{aligned}$$

Le proprietà servono a individuare le caratteristiche della rappresentazione fixed point su una macchina.

Un codice per individuare le caratteristiche fixed point

Calcolo del numero di bit $t + 1$ usati nella rappresentazione:

$$\text{pr} = 2^t - 1 \;\Rightarrow\; \text{pr}+1 = 2^t \;\Rightarrow\; \log_{10}(\text{pr}+1) = t\log_{10}(2) \;\Rightarrow\; t = \log_{10}(\text{pr}+1)/\log_{10}(2)$$

Il seguente codice C permette di vedere le caratteristiche dei numeri fixed point (tipo **int**) sulla macchina su cui è eseguito:

```
#include <stdio.h>
#include <math.h>
int main() {
  int num = 1, t;
  while (num >= 0) ++num;
  --num;
  t = (int)( log10( (double)num + 1.0 ) / log10(2.0) );
  printf("\n max intero rappresentabile = %d\t cifre = \%d", num, t+1);
  num = -1;
  while (num < 0) --num;
  ++num;
  printf("\n min intero rappresentabile = %d \n\n",num);
}
```

Risultato ottenuto da una esecuzione (architettura Intel i7):

```
$> ./prova.x
 max intero rappresentabile = 2147483647    cifre = 32
 min intero rappresentabile = -2147483648
```

Aritmetica dei numeri fixed point

Si assume di operare con interi non negativi tali che $\mathrm{fi}(N) = N$.

SOMMA
$\mathrm{fi}(N_1 + N_2) = \mathrm{fi}\big(\mathrm{fi}(N_1) + \mathrm{fi}(N_2)\big) = (N_1 + N_2)_{t+1}$.

Si può avere un overflow intero quando si ha un riporto sulla cifra $(t+1)$-esima. In questo caso la somma di due positivi fornisce un numero che è la rappresentazione di un negativo.

Esempi. $\beta = 2, t = 5$.

- $N_1 = 010010\big(= (18)_{10}\big)$, $N_2 = 000101\big(= 5_{10}\big)$
  $\mathrm{fi}(N_1 + N_2) = 010111(= 23_{10})$.

  $$\begin{array}{r} 010010\ + \\ 000101 \\ \hline 010111 \end{array}$$

- $N_1 = 010011\big(= (19)_{10}\big)$, $N_2 = 001110\big(= (14)_{10}\big)$
  $\mathrm{fi}(N_1 + N_2) = 100001 = \mathrm{fi}\big(-(31)_{10}\big)$ invece di $(33)_{10}$.

  $$\begin{array}{r} 010011\ + \\ 001110 \\ \hline 100001 \end{array}$$

Aritmetica dei numeri a virgola fissa (fixed point)

DIFFERENZA
$\mathrm{fi}(N_1 - N_2) = \big(\mathrm{fi}(N_1) + \mathrm{fi}(-N_2)\big)_{t+1}$

In pratica:
$\mathrm{fi}(N_1 - N_2) = \big(N_1 + \beta^{t+1} - N_2\big)_{t+1} = \big(\beta^{t+1} + (N_1 - N_2)\big)_{t+1}$

Il risultato ottenuto differisce da quello esatto di β^{t+1}. Ma se non si toglie β^{t+1} e si tronca alla $(t+1)$-esima cifra significativa si ottiene esattamente la rappresentazione fixed point del risultato.
Nel caso di differenza tra due numeri dello stesso segno non si ha mai overflow.

Aritmetica dei numeri a virgola fissa (fixed point)

1. Sia $N_1 \geqslant N_2$. Allora $N_1 - N_2$ è positivo o nullo.
   $\beta^{t+1} + N_1 - N_2$ ha una cifra 1 nella $(t+2)$-esima posizione significativa: **troncare alla $(t+1)$-esima cifra equivale a ottenere la rappresentazione corretta del numero positivo o nullo $N_1 - N_2$.**
   Se $\beta = 2$, $t+1 = 6$, $N_1 = 001111 (= (15)_{10})$, $N_2 = 000111 (= 7_{10})$,

$$\begin{aligned} \text{fi}(N_1 - N_2) &= (001111 + 111001)_{2,\, t+1=6} = (1001000)_{2,\, t+2=7} \\ &= (001000)_{2,\, t+1=6} = \text{fi}\big(8_{10}\big) \end{aligned}$$

2. Sia $N_1 < N_2$. Allora il risultato è il numero negativo $-(N_2 - N_1)$, di valore assoluto $N_2 - N_1$.

$$\beta^{t+1} + N_1 - N_2 = \beta^{t+1} - (N_2 - N_1) = \text{fi}(-(N_2 - N_1))$$

   Tale numero esprime la rappresentazione complemento alla base in $(t+1)$-cifre del risultato.
   Se $N_1 = 000111 (= 7_{10})$, $N_2 = 001111 (= 15_{10})$,

$$\begin{aligned} \text{fi}(N_1 - N_2) &= (000111 + 110001)_{2,\, t+1=6} = (111000)_{2,\, t+1=6} \\ &= \text{fi}\big(-(001000)_2\big) = \text{fi}(-8_{10}). \end{aligned}$$

Aritmetica dei numeri a virgola fissa (fixed point)

PRODOTTO

Il prodotto in virgola fissa è riconducibile ad addizioni e scorrimenti delle cifre a destra. Poiché il prodotto di interi in $t+1$ cifre può produrre un risultato in $2(t+1)$ cifre, spesso si incorre in overflow intero. Si usa un accumulatore B di $2(t+1)$ cifre, la cui parte più significativa R è inizialmente nulla.

QUOZIENTE

Si usano due accumulatori A e B. In A si mette il divisore e nella parte più significativa R di B il dividendo. Si riconduce a sottrazioni e scorrimenti.

ATTENZIONE

L'aritmetica tra numeri fixed point è esatta purchè si resti nell'intervallo rappresentabile.

Numeri a virgola mobile

L'insieme dei numeri reali è SIMULATO su un calcolatore mediante un **insieme $\mathcal{F}$ di numeri finiti o numeri *floating point***; tale insieme dipende da quattro parametri:

- β: base di rappresentazione;
- t: numero di cifre per la rappresentazione della mantissa;
- L: valore del più piccolo esponente rappresentabile;
- U: valore del più grande esponente rappresentabile.

Si denota tale insieme con $\mathcal{F}(\beta, t, L, U)$.

In particolare, si descrive il **formato IEEE (Institute of Electrical and Electronic Engineerings) Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985), definita dalle convenzioni contenute nel documento 754 dell'ANSI (1985, riconfermato nel 2008 con 754r)**. È noto anche come IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems*.
Tale aritmetica usa $\beta = 2$.
Si scrive un numero reale $\alpha \neq 0$ espresso in base 2 nella notazione speciale (**normalizzazione IEEE**):

$$\alpha = (-1)^s (1.a_2 a_3 \ldots a_t a_{t+1} \ldots) 2^p$$

Numeri a virgola mobile

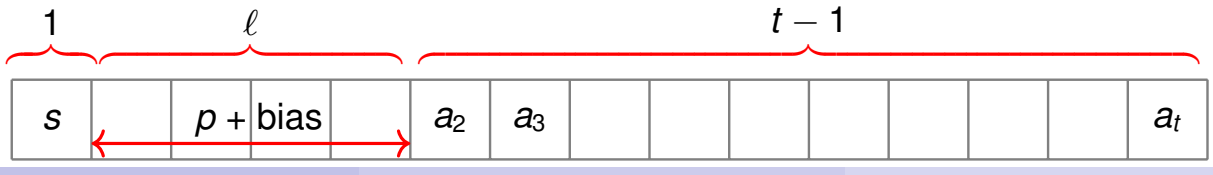
Si rappresenta:

- segno: si rappresenta in un bit il valore di s, pari a 0 se $\alpha > 0$ e a 1 se $\alpha < 0$;
- esponente p: è un intero che deve essere compreso tra L e U; la rappresentazione di p è ***per traslazione in ℓ bit***, ossia:

  $$\text{rappresentazione di } p = p + bias$$

  dove il **bias** è, dunque, la rappresentazione dello 0;
- mantissa: vengono rappresentati i t bit più significativi, troncando; *fisicamente* poiché il primo bit è sempre 1, vengono rappresentati solo $t-1$ bit ($a_2 a_3 \ldots a_t$) (*hidden bit*).

	SEMPLICE PRECISIONE	DOPPIA PRECISIONE	QUADRUPLA PRECISIONE
N. totale di bit	32 (4 byte)	64 (8 byte)	128 (16 byte)
segno s	1 bit	1 bit	1
t	24	53	113
ℓ	8	11	15
bias	127 (01111111)	1023 (01111111111)	16383 (011111111111111)
U	127	1023	16383
L	-126	-1022	-16382



Esempi

Si usano le convenzioni della singola precisione.

- Il numero $1 = 1.00 \cdot 2^{(0)_2}$ è rappresentato come
  0 01111111 00000000000000000000000
  segno esponente mantissa
- Il numero $(4.25)_{10} = (100.01)_2 = 1.0001 \cdot 2^{(10)_2}$ è rappresentato come
  0 10000001 00010000000000000000000
  segno esponente mantissa
  $127 + 2 = 129$
- Il numero $(3.141592)_{10} = (11.001001000011111110101111..)_2$
  $= 1.10010010000111111010111 \cdot 2^{(1)_2}$ è rappresentato come
  0 10000000 10010010000111111010111
  segno esponente mantissa
  $127 + 1 = 128$
- Il numero $(0.333333)_{10} = (0.01010101010101010010011111\ldots)_2$
  $= 1.01010101010101010010011111\ldots \cdot 2^{(-10)_2}$ è rappresentato come
  0 01111101 01010101010101010010011111
  segno esponente mantissa
  $127 - 2 = 125$

Gli estremi dell'intervallo rappresentabile

Più piccolo numero rappresentabile in valore assoluto:
$2^L = 1.0000\ldots \cdot 2^{-126} \simeq 10^{-38}$
0 00000001 00000000000000000000000
segno esponente mantissa
$-126 + 127 = 1$

Più grande numero rappresentabile in valore assoluto:
$(1 + 1 - \beta^{-t+1})2^U = 1.1111\ldots1 \cdot 2^{127} = (1 + 1 - 2^{-23})2^{127} \simeq 10^{38}$
0 11111110 11111111111111111111111
segno esponente mantissa
$127 + 127 = 254$

Se si cerca di rappresentare un numero in valore assoluto più piccolo del più piccolo rappresentabile, si incorre nell'**UNDERFLOW floating point**; il calcolatore rappresenta il numero con 0 (o con numeri denormalizzati o subnormalizzati) e prosegue l'elaborazione.

Se si cerca di rappresentare un numero più grande del più grande rappresentabile in valore assoluto, si incorre nell'**OVERFLOW floating point** che arresta l'elaborazione.

Rappresentazione degli esponenti

Nella **rappresentazione degli esponenti** valgono le seguenti regole:

$p = 0$ è rappresentato con 01111111 (**bias**)

$p = 1$ è rappresentato con 10000000

$p = -1$ è rappresentato con 01111110

$p = 127 = U$ è rappresentato con $11111110 = (254)_{10}$

$p = -126 = L$ è rappresentato con $00000001 = (1)_{10}$

Gli esponenti negativi o nulli hanno il bit più significativo uguale a 0, gli esponenti positivi uguale a 1.

Paragone dell'interpretazione fra la rappresentazione in complemento e la rappresentazione in traslazione (caso con 4 bit):

fi(N)	N	trasl(N)	N	
1111	-1	1111	8	(speciale)
1110	-2	1110	7	
1101	-3	1101	6	
1100	-4	1100	5	
1011	-5	1011	4	
1010	-6	1010	3	
1001	-7	1001	2	
1000	-8	1000	1	
0111	7	0111	0	(bias)
0110	6	0110	-1	
0101	5	0101	-2	
0100	4	0100	-3	
0011	3	0011	-4	
0010	2	0010	-5	
0001	1	0001	-6	
0000	0	0000	-7	(speciale)

Rappresentazioni speciali

mantissa	esponente		
0	0	rappresenta	$(-1)^s 0$
0	$11111111 = (255)_{10}$	rappresenta	$\pm\infty$
$\neq 0$	$11111111 = (255)_{10}$	rappresenta	NaN

NUMERI DENORMALIZZATI: sono numeri più piccoli di 2^{-126} (gradual underflow).

esponente	mantissa	numero
00000000	$100\ldots0$	2^{-127}
00000000	$010\ldots0$	2^{-128}
$\vdots$		
00000000	$0\ldots001$	2^{-149}

Permette di recuperare qualche problema ma l'aritmetica diventa molto lenta.

Numeri in virgola mobile

L'insieme $\mathcal{F}(\beta, t, L, U)$ dei numeri a virgola mobile è non continuo:

- è discreto e limitato
- contiene esattamente $N_{\mathcal{F}} = 2(\beta-1)\beta^{t-1}(U-L+1)+1$ elementi, che si ottengono considerando tutte le possibili combinazioni tra segno, mantissa ed esponente (oltre allo zero):

$$\alpha = \pm\left(0.a_1 a_2 \ldots a_t\right)_\beta \cdot \beta^r$$

segno	$\pm \Rightarrow 2$ simboli
mantissa	$\left\{\begin{array}{rl} a_1 \neq 0 & \Rightarrow \beta - 1 \text{ simboli} \\ a_2 & \Rightarrow \beta \text{ simboli} \\ \vdots & \\ a_t & \Rightarrow \beta \text{ simboli} \end{array}\right\} t-1$ volte
esponente	$r \in [L, U] \cap \mathbb{Z} \Rightarrow U - L + 1$ valori interi

Attenzione: nel caso binario ($\beta = 2$) il numero di elementi cambia a causa della convenzione per la quale, essendo necessariamente $a_1 = 1$, tutte le t cifre decimali rappresentate possono assumere i $\beta = 2$ valori disponibili, invece che solo le ultime $t-1$ come nel caso standard. Nel caso binario, dunque, si ha $N_{\mathcal{F}} = 2 \cdot 2^t(U-L+1)+1 = 2^{t+1}(U-L+1)+1$.

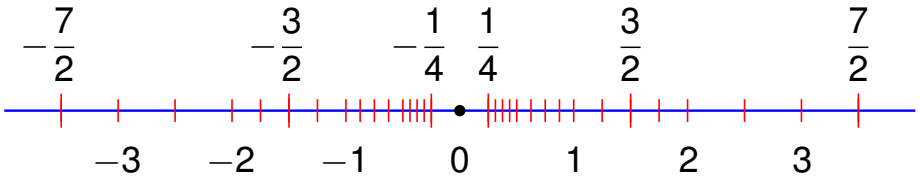
Numeri in virgola mobile

Esempio: con $\beta = 2$, $t = 2$, $L = -2$ e $U = 1$, l'insieme $\mathcal{F}(2, 2, -2, 1)$ contiene esattamente $2^3(1+2+1)+1 = 33$ elementi:

$$\alpha = \pm 1.a_2 a_3 \cdot 2^p, \quad p \in \{-2, -1, 0, 1\}, \quad a_2, a_3 \in \{0, 1\}$$

mantissa m \ esponente p	-2	-1	0	1
1.00	$\pm 1/4$	$\pm 1/2$	± 1	± 2
1.01	$\pm 5/16$	$\pm 5/8$	$\pm 5/4$	$\pm 5/2$
1.10	$\pm 6/16$	$\pm 3/4$	$\pm 3/2$	± 3
1.11	$\pm 7/16$	$\pm 7/8$	$\pm 7/4$	$\pm 7/2$

$\cup\ 0$



Numeri in virgola mobile

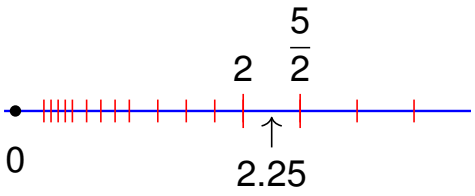
Osservazioni

- Attorno allo 0 c'è un intervallo $[-2^L, 2^L]$ di numeri reali che vengono rappresentati dal solo 0 (underflow)
- Per numeri più grandi di $\left(2 - 2^{-(t-1)}\right)2^U$ o più piccoli di $-\left(2 - 2^{-(t-1)}\right)2^U$ si incorre in overflow
- numeri di modulo piccolo sono meglio rappresentati

Come si rappresenta $\alpha \notin \mathcal{F}$? Con troncamento o con arrotondamento

Esempio: $\beta = 2$, $t = 2$, $L = -2$ e $U = 1$

$$\alpha = (2.25)_{10} = (1.001)_2 \cdot 2^1 \rightarrow \begin{cases} \text{troncamento}: & (1.00)_2 \cdot 2^1 \\ \text{arrotondamento}: & (1.01)_2 \cdot 2^1 \end{cases}$$



Approssimazione dei reali con numeri a virgola mobile

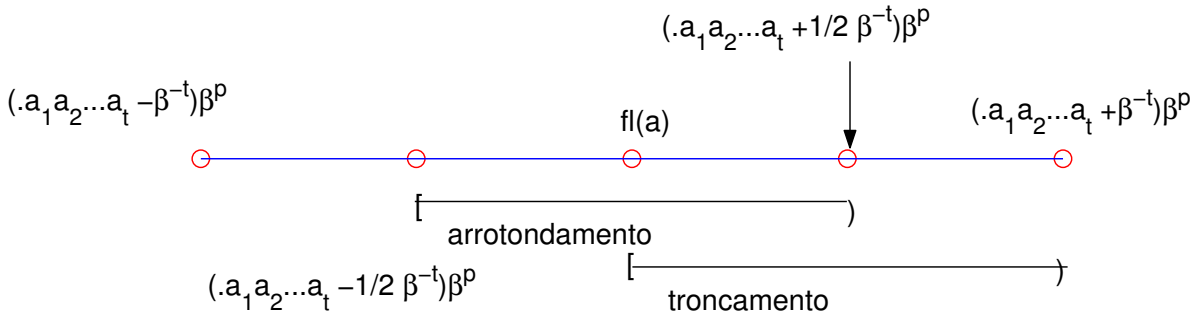
Ogni elemento di $\mathcal{F}$ rappresenta sé stesso e un intero intervallo di numeri reali.
Se $\text{fl}(\alpha) = (\sum_{i=1}^{t} a_i \beta^{-i})\beta^p$, allora $\text{fl}(\alpha)$ rappresenta l'intervallo di ampiezza $\beta^{-t}\beta^p$ dato da

$$\left[\left(\sum_{i=1}^{t} a_i \beta^{-i} \right) \beta^p, \left(\sum_{i=1}^{t} a_i \beta^{-i} + \beta^{-t} \right) \beta^p \right)$$

nel caso di RAPPRESENTAZIONE PER TRONCAMENTO, oppure da

$$\left[\left(\sum_{i=1}^{t} a_i \beta^{-i} - \frac{1}{2}\beta^{-t} \right) \beta^p, \left(\sum_{i=1}^{t} a_i \beta^{-i} + \frac{1}{2}\beta^{-t} \right) \beta^p \right)$$

nel caso di RAPPRESENTAZIONE PER ARROTONDAMENTO.



Numeri in virgola mobile

Troncamento: tutti i numeri in $[2,\ 5/2[$ vengono rappresentati come "2":

$$\left.\begin{aligned}(2)_{10} &= \mathrm{fl}\big((1.00)_2\cdot 2^1\big)\\ &= \mathrm{fl}\big((1.001)_2\cdot 2^1\big)\\ &= \mathrm{fl}\big((1.0011)_2\cdot 2^1\big)\\ &\vdots\end{aligned}\right\}\xrightarrow{\mathrm{fl}}\ 1.00\cdot 2^1$$

$$(2.5)_{10} = (5/2)_{10} = (1.01)_2\cdot 2^1 \xrightarrow{\mathrm{fl}} (1.01)_2\cdot 2^1$$

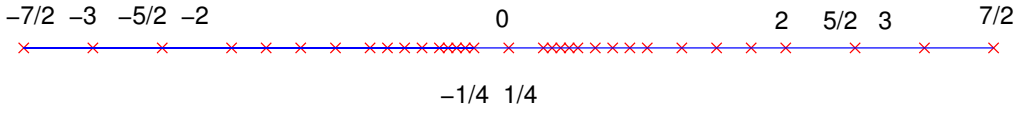
Arrotondamento: tutti i numeri in $[15/8,\ 9/4[$ vengono rappresentati come "2":

$$\left.\begin{aligned}\mathrm{fl}\big((15/8)_{10}\big) &= \mathrm{fl}\big((1.111)_2\cdot 2^0\big)\\ &= \mathrm{fl}\big((1.1111)_2\cdot 2^0\big)\\ &= \mathrm{fl}\big((1.111101)_2\cdot 2^0\big)\\ &\vdots\\ &= \mathrm{fl}\big((1.00)_2\cdot 2^1\big)\\ &= \mathrm{fl}\big((1.00011)_2\cdot 2^1\big)\\ &\vdots\end{aligned}\right\}\xrightarrow{\mathrm{fl}}(1.00)_2\cdot 2^1\ ,\ \text{mentre}\ \mathrm{fl}\big((2.25)_{10}\big) = \mathrm{fl}\big((9/4)_{10}\big) = \mathrm{fl}\big((1.001)_2\cdot 2^1\big) \xrightarrow{\mathrm{fl}} (1.01)_2\cdot 2^1$$

Ogni volta che si vuole rappresentare un numero reale che non appartiene all'insieme $\mathcal{F}$, si commette un errore.

Esempio

Nel caso di $\beta = 2, t = 3, L = -2, U = 1$,



$$(2.25)_{10} = 1.001\cdot 2^1 = \begin{cases}\text{troncamento } 1.00\cdot 2^1 = (2)_{10}\\ \text{arrotondamento } 1.01\cdot 2^1 = \dfrac{5}{2} = (2.5)_{10}\end{cases}$$

- TRONCAMENTO: tutti i numeri tra $\left[2, \dfrac{5}{2}\right)$ (numeri in base 10) sono rappresentati con $2 = 1.00\cdot 2^1$ $(1.001\cdot 2^1;\ 1.0011\cdot 2^1;\ldots)$;
- ARROTONDAMENTO: il numero $(2)_{10} = 1.00\cdot 2^1$ rappresenta tutti i numeri tra $\dfrac{1}{2}\left(\dfrac{7}{4}+2\right) = \dfrac{15}{8}$ compreso e $\dfrac{1}{2}\left(2+\dfrac{5}{2}\right) = \dfrac{9}{4} = (2.25)_{10}$ escluso $(1.0001\cdot 2^1;\ 1.111\cdot 2^0;\ldots)$.

Teorema dell'errore di rappresentazione di un numero finito

Teorema (errore di rappresentazione di un numero reale)

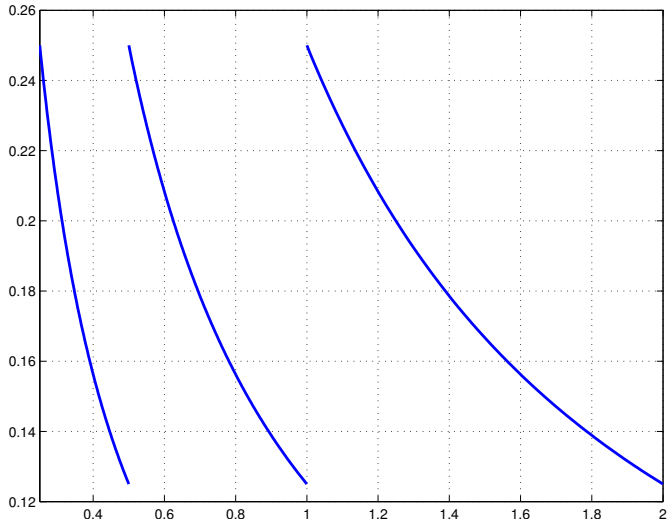
Sia $\alpha \in \mathbb{R}$, allora

$$\left| \frac{\mathrm{fl}(\alpha) - \alpha}{\alpha} \right| \leqslant k\beta^{1-t}$$

con $k = 1$ per il troncamento e $k = 1/2$ per l'arrotondamento.
Inoltre $\mathrm{fl}(\alpha) = \alpha(1 + u)$ con $|u| \leqslant k\beta^{1-t}$.
La quantità $\varepsilon = k\beta^{1-t}$ si chiama precisione di macchina.

La separazione fra i numeri finiti

Se si assume $k = 1$ (troncamento), la distanza tra numeri finiti successivi rappresentabili è uguale tra le potenze della base (poiché in notazione IEEE la mantissa aumenta da 1 a $1 + 1 - \beta^{-t+1}$), ma quando si passa alla potenza successiva, aumenta; se si considera la distanza relativa, essa ha un andamento oscillatorio tra le potenze di β (wobbling precision), decrescendo da un valore massimo a un minimo:



Precisione di macchina

La precisione di macchina è caratteristica del calcolatore che si usa e rappresenta il più piccolo numero sentito dalla macchina relativamente, ossia è caratterizzato dal fatto che

$$\text{fl}(1+u) > 1 \quad \text{mentre} \quad \text{fl}(1+a) = 1 \quad \text{per ogni } a < u$$

Infatti

$$(1 + k\beta^{1-t}) = (.1\beta^1 + k\beta^{1-t}) = \beta(\beta^{-1} + k\beta^{-t}) > 1$$

Precisione di macchina

Esempio
Sia $\beta = 2, t = 3$.
Se $u = \beta^{-2} = 2^{-2} = 0.01$ (TRONCAMENTO), allora:

$$1 + 0.01 = 1.01 > 1$$
$$1 + 0.001 = 1.001, \text{fl}(1.001) = 1$$

Se $u = \frac{1}{2}\beta^{-2} = 2^{-3} = 0.001$ (ARROTONDAMENTO), allora:

$$1 + 0.001 = 1.001, \text{fl}(1.001) = 1.01 > 1$$
$$1 + 0.0001 = 1.0001, \text{fl}(1.0001) = 1$$

Allora ponendo una variabile uguale a 1 e continuando a dimezzarla finchè non viene più sentita nella somma con 1, il ritorno indietro di un passo determina un elemento dell'ordine della precisione di macchina.

```
u = 1;
while (1 + u ~= 1)
   u = u/2;
end
u = u*2;
fprintf('\nOrdine della precisione di macchina = %g \n',u);
```

Precisione di macchina

Per determinare la base della rappresentazione si osserva che:

$$\mathrm{fl}(n+u) > n \;\forall n = 0, 1, \ldots, \beta - 1 \quad \text{ma} \quad \mathrm{fl}(\beta + u) = \beta$$

Infatti:

$$\mathrm{fl}(n+u) = \beta(n\beta^{-1} + k\beta^{-t}) > n$$
$$\mathrm{fl}(\beta+u) = \beta(1 + k\beta^{-t}) = \beta$$

ESEMPIO. Sia $\beta = 10, t = 3, u = 0.5 \cdot 10^{-2}$ (ARROTONDAMENTO).

$$9 + 0.005 = 9.005, \quad \mathrm{fl}(9.005) = 9.01 > 9$$
$$10 + 0.005 = 10.005, \quad \mathrm{fl}(10.005) = 10$$

La base si può dunque determinare facilmente con un semplice frammento di codice Matlab:

```
b = 1;
while (b + u ~= b)
   b = b + 1;
end
fprintf('\nBase = %d \n',b);
```


Precisione di macchina

Infine, per decidere se la macchina lavora per troncamento o per arrotondamento, si osserva che:
dati $\epsilon = \beta^{1-t}, \epsilon_1 = \beta^{-t}$,

$$(1+\epsilon) - \epsilon_1 = \begin{cases} > 1 & \text{arrotondamento} \quad u = \frac{1}{2}\epsilon \\ = 1 & \text{troncamento} \quad u = \epsilon \end{cases}$$

Infatti:

$$\begin{aligned}(1 + \beta^{1-t}) - \beta^{-t} &= \beta(\beta^{-1} + \beta^{-t}) - \beta^{-t} \\ &= \beta(\beta^{-1} + \beta^{-t} - \beta^{-t-1}) \\ &= \beta(\beta^{-1} + \beta^{-t-1}(\beta - 1))\end{aligned}$$

Allora:

$$\begin{array}{lcl} \text{troncamento} & \Rightarrow & \beta\beta^{-1} = 1 \\ \text{arrotondamento} & \Rightarrow & \beta(\beta^{-1} + \beta^{-t}) > 1 \end{array}$$

ESEMPIO. Siano $\beta = 10$, $t = 3$, $\epsilon = 10^{-2}, \epsilon_1 = 10^{-3}$.

$$1 + \epsilon = 1 + 0.01 = 1.01 = r$$
$$r - \epsilon_1 = 1.01 - 0.001 = 1.009$$

$\mathrm{fl}(1.009) = 1$ nel caso di troncamento e $\mathrm{fl}(1.009) = 1.01 > 1$ nel caso di arrotondamento.

Codice Matlab delle caratteristiche dell'aritmetica di macchina

```
u = 1;
while (1 + u ~= 1)
   u = u/2;
end
u = u*2;
fprintf('\nOrdine della precisione di macchina = %g', u);
b = 1;
while (b + u ~= b)
   b = b + 1;
end
fprintf('\nBase = %g ', b);
u = 1;
while (1 + u ~= 1)
   u = u/b;
end
u1 = u;
u = u*b;
if (1 + u) - u1 > 1
   u = u/2;
   fprintf('\nArrotondamento');
   t = 1 - log10(2*u) / log10(b);
else
   fprintf('\nTroncamento');
   t = 1 - log10(u) / log10(b);
end
fprintf('\nPrecisione di macchina = %24.16e', u);
fprintf('\nNumero di bit per la mantissa = %g\n', t);
```


Codice dell'algoritmo della precisione di macchina

In realtà nel caso di codifica IEEE-754, il risultato di una operazione viene troncato secondo la regola di approssimare con *l'intero pari più vicino*.

Esempio. Lavorando con tre cifre binarie, $N = 1.001$ viene approssimato con 1.00 (i candidati a rappresentare N sono gli interi 100 e 110: 1000 è più vicino a 1001 di 1100), mentre $N = 1.011$ viene approssimato con 1.10 (1100 è più vicino a 1011 di 1000).

Quindi la precisione di macchina resta quella del caso del troncamento, ma nel 50% dei casi l'errore è quello del caso dell'arrotondamento.

La singola precisione (binary32) porta a circa 7 cifre decimali di precisione, la doppia (binary64) a circa 15 cifre decimali e la quadrupla (binary128) a circa 34 cifre decimali.

Nota.
Esiste anche una *half precision*, che corrisponde a un numero codificato in 16 bit (2 byte), ove per l'esponente si usano 5 bit e per la mantissa 11 bit.
Il documento dello standard 2008 definisce anche tre formati a base 10, codificati in 32, 64, 128 bit.

OPERAZIONI SUI NUMERI FINITI

Dati $x, y \in \mathcal{F}(\beta, t, L, U)$, non è detto che il risultato di una operazione tra x e y sia un elemento di $\mathcal{F}$. Può essere un numero maggiore del massimo numero rappresentabile in modulo o avere una mantissa con più di t cifre.

Esempio.
Sistema $\mathcal{F}$ dato da $\beta = 2$, $t = 3$, $L = -2$, $U = 1$

$$\mathcal{F} = \left\{ \pm 1.a_1 a_2 \cdot 2^p, \quad p = -2, -1, 0, 1;\ a_1 = 0, 1;\ a_2 = 0, 1 \right\}$$

$m \backslash p$	-2	-1	0	1
1.00	$\pm 1/4$	$\pm 1/2$	± 1	± 2
1.01	$\pm 5/16$	$\pm 5/8$	$\pm 5/4$	$\pm 5/2$
1.10	$\pm 6/16$	$\pm 3/4$	$\pm 3/2$	± 3
1.11	$\pm 7/16$	$\pm 7/8$	$\pm 7/4$	$\pm 7/2$

$\cup \quad \{0\}$

$$2 + \frac{1}{4} = 1.00 \cdot 2^1 + 1.00 \cdot 2^{-2} = 1.001 \cdot 2^1 \notin \mathcal{F}$$

Occorre ridefinire la somma in modo che il risultato sia un elemento di $\mathcal{F}$.

Operazioni tra numeri finiti

Si devono **ridefinire** le operazioni di macchina nel seguente modo:

$$x \circ y = \mathrm{fl}(x \bullet y) \quad x, y \in \mathcal{F}$$

ove $\bullet$ è $+$, $-$, $*$, $/$ (operazione esatta) e $\circ$ è l'operazione di macchina ridefinita.

- Si tratta di eseguire l'operazione esatta tra x e y che sono in $\mathcal{F}$;
- poi si rappresenta il risultato entro $\mathcal{F}$.

Pertanto, dal Teorema dell'errore di rappresentazione, segue il seguente teorema fondamentale:

TEOREMA

Siano $x, y \in \mathcal{F}(\beta, t, L, U)$. Allora

$$\frac{|(x \circ y) - x \bullet y|}{|x \bullet y|} = \frac{|\mathrm{fl}(x \bullet y) - x \bullet y|}{|x \bullet y|} \leqslant k\beta^{1-t}$$

dove $k = 1$ o $1/2$ a seconda che la rappresentazione sia per troncamento o per arrotondamento. Equivalentemente si può scrivere

$$x \circ y = \mathrm{fl}(x \bullet y) = (x \bullet y)(1 + \epsilon) = \frac{x \bullet y}{1 + \gamma} \qquad |\epsilon| \leqslant k\beta^{1-t},\ |\gamma| \leqslant k\beta^{1-t}$$

Approssimazione del risultato

Il risultato di un'operazione deve essere un numero di macchina.

$$\Downarrow$$

Di conseguenza il risultato *esatto* viene normalizzato e poi arrotondato o troncato. Ogni operazione introduce potenzialmente un errore relativo, il cui valore è maggiorato dalla precisione di macchina.

Nel caso dell'esempio, operando per arrotondamento ($\beta = 2$, $t = 3$),

$$\mathrm{fl}\left(2 + \frac{1}{4}\right) = \mathrm{fl}(1.00 \cdot 2^1 + 1.00 \cdot 2^{-2}) = \mathrm{fl}(1.001 \cdot 2^1) = 1.01 \cdot 2^1 = \frac{5}{2} \in \mathcal{F}$$

$$E_r = \frac{|(x \circ y) - x \bullet y|}{|x \bullet y|} = \frac{|5/2 - 9/4|}{9/4} = \frac{1}{9} \leqslant \frac{1}{2}2^{-2} = \frac{1}{8}$$

Operazioni sui numeri finiti

Siano x e $y \in F(\beta, t, L, U)$, assumendo che la mantissa sia in forma normalizzata:

$$x = x_m \cdot \beta^{x_e}$$
$$y = y_m \cdot \beta^{y_e}$$

SOMMA ALGEBRICA. $z = z_m \cdot \beta^{z_e} = \mathrm{fl}(x \pm y)$.

1. Si confrontano x_e e y_e; se $x_e > y_e$, si divide y_m per $\beta^{x_e - y_e}$ (in altre parole si scala l'addendo più piccolo in modo che i due numeri abbiano lo stesso esponente (quello più alto);
2. si esegue $x_m \pm y_m / \beta^{x_e - y_e}$;
3. si considerano le prime t cifre più significative (con troncamento o arrotondamento), ponendole in z_m. Se il risultato è maggiore o uguale a 1, $h = 1$ altrimenti si pone in h l'opposto del numero degli zeri ottenuti dopo il punto radice;
4. $z_e = x_e + h$.

Esempi

$\beta = 10$, $t = 5$, arrotondamento

- $x = 0.64932 \cdot 10^7$, $y = 0.53726 \cdot 10^4$, $z = \mathrm{fl}(x + y)$
  1) $x_e - y_e = 3$;
  2) $0.64932 + 0.00053726 = 0.64985726$
  3) $z_m = 0.64986$
  4) $z_e = 7$

  $z = 0.64986 \cdot 10^7$.
- $x = 0.64937 \cdot 10^7$, $y = 0.53726 \cdot 10^7$, $z = \mathrm{fl}(x + y)$
  1) $x_e - y_e = 0$;
  2) $0.64932 + 0.53726 = 1.18658$
  3) $z_m = 0.11866$; $h = 1$;
  4) $z_e = 7 + 1$

  $z = 0.11866 \cdot 10^8$
- $x = 0.75869 \cdot 10^2$, $y = 0.75868 \cdot 10^2$, $z = \mathrm{fl}(x - y)$
  1) $x_e - y_e = 0$;
  2) $0.75869 - 0.75868 = 0.00001$
  3) $z_m = 0.1$; $h = -4$;
  4) $z_e = 2 - 4$

  $z = 0.1 \cdot 10^{-2}$.

Esempi

In quest'ultimo caso si ha una **cancellazione di cifre**; vengono introdotte quantità spurie, poiché si esegue una differenza tra due quantità circa uguali, perdendo cifre (effetto smearing).
La situazione non è pericolosa se $E_a = \epsilon_r = 0$; lo diventa quando i dati di partenza sono affetti da errore.
Se infatti

$$x = \mathrm{fl}(0.75868531 \cdot 10^2) = 0.75869 \cdot 10^2$$

$$E_{ax} = 4.69 \cdot 10^{-4} \quad \epsilon_{rx} = 0.6181 \cdot 10^{-5} \leqslant \frac{1}{2} \cdot 10^{-4}$$

$$y = \mathrm{fl}(0.75868100 \cdot 10^2) = 0.75868 \cdot 10^2$$

$$E_{ay} = 1. \cdot 10^{-4} \quad \epsilon_{ry} = 0.1318 \cdot 10^{-5} \leqslant \frac{1}{2} \cdot 10^{-4}$$

Il risultato esatto vale $0.431 \cdot 10^{-3}$, ma

$$E_a = \left|0.10 \cdot 10^{-2} - 0.431 \cdot 10^{-3}\right| = 0.0569 \cdot 10^{-2} \quad \epsilon_r = \frac{0.569 \cdot 10^{-3}}{0.431 \cdot 10^{-3}} \approx 1.320186 = 132\%$$

La cancellazione determina una amplificazione dell'errore iniziale sui dati.

Esempi

- $x = 0.62379 \cdot 10^7$, $y = 0.32881 \cdot 10^1$, $z = \mathrm{fl}(x + y)$
  1) $x_e - y_e = 6$;
  2) $0.62379 + 0.00000032881 = 0.62379032881$
     $z_m = 0.62379$
  3) $z_e = 7$

  $z = 0.62379 \cdot 10^7$ anche se $y \neq 0$.

Quando $x + y = x$ con $y \neq 0$, si verifica un
errore di incolonnamento.
Questo capita ogni volta che $|y| \leqslant \dfrac{u}{\beta}|x|$.

Di conseguenza, non esiste un solo elemento neutro per la somma.

PRODOTTO

$z = z_m \cdot \beta^{z_e} = \mathrm{fl}(xy)$

1. si esegue il prodotto $x_m \cdot y_m$,
2. si tronca o arrotonda il risultato a t cifre; si memorizza in z_m e si pone $h = 1$ se si ottiene uno zero a destra del punto radice, altrimenti $h = 0$;
3. $z_e = x_e + y_e - h$.

Esempio. $\beta = 10$, $t = 5$, arrotondamento, $z = \mathrm{fl}(x \cdot y)$

- $x = 0.11111 \cdot 10^7$, $y = 0.10202 \cdot 10^{-2}$
  1) $0.111111 \cdot 0.10202 = 0.0113354422$
  2) $z_m = 0.11335 \quad h = 1$;
  3) $z_e = 7 - 2 - 1 = 4$

  $z = 0.11335 \cdot 10^4$.

QUOZIENTE

$z = z_m\,\beta^{z_e} = \mathrm{fl}(x/y)$

1. Se $x_m < y_m$ si pone $h = 0$; altrimenti si divide x_m per β e si pone $h = 1$;
2. Si esegue $\left(x_m/\beta^h\right)/y_m$
3. si pongono le t cifre più significative del risultato in z_m;
4. $z_e = x_e - y_e + h$.

Esempio. $\beta = 10$, $t = 5$, arrotondamento, $z = \mathrm{fl}(x/y)$

- $x = 0.62500 \cdot 10^0$; $y = 0.12500 \cdot 10^{-2}$
  1) $0.062500 \quad h = 1$;
  2) $0.06250/0.12500 = 0.5$;
  3) $z_m = 0.5$;
  4) $z_e = 0 + 2 + 1 = 3$

  $z = 0.5 \cdot 10^3$.

Analisi in avanti al prim'ordine degli errori di arrotondamento

Analisi in avanti dell'errore algoritmico:

- si basa sul Teorema dell'errore:

$$\mathrm{fl}(x \bullet y) = (x \bullet y)(1 + \epsilon) \qquad |\epsilon| \leqslant k\beta^{1-t}$$

- si calcola l'errore relativo del risultato finale rispetto agli errori relativi introdotti dalle singole operazioni dell'algoritmo
- dato che ci si limita ad un'analisi del prim'ordine, si trascurano i termini non lineari negli errori
- Fattori di amplificazione: coefficienti moltiplicativi (in valore assoluto) degli errori nelle operazioni di macchina
- Indice algoritmico I_{alg}: somma dei moduli dei fattori di amplificazione
  Poichè per una singola operazione

$$\epsilon_{\mathrm{alg}} = \frac{\mathrm{fl}(x \bullet y) - (x \bullet y)}{x \bullet y} = \frac{(x \bullet y)(1 + \epsilon_1) - (x \bullet y)\big|}{x \bullet y} = 1\epsilon_1$$

  l'indice algoritmico è sempre almeno 1, cioè $I_{\mathrm{alg}} \geqslant 1$

Esempio

Somma di tre numeri reali: caso 1

$\varphi(a,b,c) = a+b+c$ Algoritmo 1: $(a+b)+c$

$$\mathrm{fl}(a+b) = \underbrace{(a+b)(1+\epsilon_1)}_{y} \quad |\epsilon_1| < u$$

$$\mathrm{fl}(y+c) = (y+c)(1+\epsilon_2) \quad |\epsilon_2| < u$$

$$\mathrm{fl}(a+b+c) = \mathrm{fl}(y+c) = \big((a+b)(1+\epsilon_1)+c\big)(1+\epsilon_2)$$

$$\approx a+b+c+(a+b)\epsilon_1+(a+b+c)\epsilon_2$$

$\uparrow$ si trascura il termine in $\epsilon_1\epsilon_2$

da cui

$$\epsilon_{\mathrm{alg1}} = \frac{\mathrm{fl}(a+b+c)-(a+b+c)}{a+b+c} \approx \boxed{\frac{a+b}{a+b+c}}\epsilon_1 + \boxed{1}\epsilon_2$$

Fattori di amplificazione degli errori delle singole operazioni

$$I_{\mathrm{alg1}} = \left|\frac{a+b}{a+b+c}\right| + 1$$

Indice algoritmico

Somma di tre numeri reali: caso 2

$\varphi(a,b,c) = a+b+c$ Algoritmo 2: $a+(b+c)$

$$\mathrm{fl}(b+c) = \underbrace{(b+c)(1+\epsilon_3)}_{w} \quad |\epsilon_3| < u$$

$$\mathrm{fl}(a+w) = (a+w)(1+\epsilon_4) \quad |\epsilon_4| < u$$

$$\mathrm{fl}(a+b+c) = \mathrm{fl}(a+w) = \big(a+(b+c)(1+\epsilon_3)\big)(1+\epsilon_4)$$

$$\approx a+b+c+(b+c)\epsilon_3+(a+b+c)\epsilon_4$$

$\uparrow$ si trascura il termine in $\epsilon_3\epsilon_4$

da cui

$$\epsilon_{\mathrm{alg2}} = \frac{\mathrm{fl}(a+b+c)-(a+b+c)}{a+b+c} \approx \boxed{\frac{b+c}{a+b+c}}\epsilon_3 + \boxed{1}\epsilon_4$$

$$I_{\mathrm{alg2}} = \left|\frac{b+c}{a+b+c}\right| + 1$$

Stabilità

Confronto di algoritmi

- dati due algoritmi, alg1 ed alg2, per il calcolo di una stessa espressione, si dice che alg1 è più stabile di alg2 se $I_{\text{alg1}} < I_{\text{alg2}}$
- attenzione il confronto dipende anche dai dati!

Esempio: somma di tre numeri reali

$$a = 0.2337126 \cdot 10^{-4},\ b = 0.3367843 \cdot 10^{3},\ c = -0.3367781 \cdot 10^{3}$$

$$I_{\text{alg1}} = \left|\frac{a+b}{a+b+c}\right| + 1 \approx 1 + 0.541 \cdot 10^{5}$$
$$I_{\text{alg2}} = \left|\frac{b+c}{a+b+c}\right| + 1 \approx 1 + 0.996 \cdot 10^{0}$$

dunque per i valori assunti da questi dati, alg2 è più stabile di alg1

Esempio

Differenza di quadrati: caso 1

$\varphi(a,b) = a^2 - b^2$ Algoritmo: $a^2 - b^2$

$$\text{fl}(a^2) = \text{fl}(a \cdot a) = \underbrace{a^2(1+\epsilon_1)}_{w} \quad |\epsilon_1| < u$$
$$\text{fl}(b^2) = \text{fl}(b \cdot b) = \underbrace{b^2(1+\epsilon_2)}_{v} \quad |\epsilon_2| < u$$
$$\begin{aligned}\text{fl}(a^2 - b^2) = \text{fl}(w - v) &= (w - v)(1+\epsilon_3) \quad |\epsilon_3| < \varepsilon \\ &= \left(a^2(1+\epsilon_1) - b^2(1+\epsilon_2)\right)(1+\epsilon_3) \\ &\approx a^2 - b^2 + a^2\epsilon_1 - b^2\epsilon_2 + (a^2 - b^2)\epsilon_3\end{aligned}$$

da cui

$$\begin{aligned}\epsilon_{\text{alg1}} &= \frac{\text{fl}\left(a^2 - b^2\right) - \left(a^2 - b^2\right)}{a^2 - b^2} \\ &\approx \frac{a^2}{a^2 - b^2}\epsilon_1 - \frac{b^2}{a^2 - b^2}\epsilon_2 + \epsilon_3\end{aligned}$$

e

$$I_{\text{alg1}} = \left|\frac{a^2}{a^2 - b^2}\right| + \left|\frac{b^2}{a^2 - b^2}\right| + 1$$

Esempio

Differenza di quadrati: caso 2

$\varphi(a,b) = a^2 - b^2$ Algoritmo: $(a-b)(a+b)$

$$\mathrm{fl}(a-b) = \underbrace{(a-b)(1+\epsilon_4)}_{x} \quad |\epsilon_4| < u$$

$$\mathrm{fl}(a+b) = \underbrace{(b+b)(1+\epsilon_5)}_{y} \quad |\epsilon_5| < u$$

$$\begin{aligned}\mathrm{fl}(a^2-b^2) &= \mathrm{fl}(x\cdot y) = (x\cdot y)(1+\epsilon_6) \quad |\epsilon_6| < u \\ &= \big((a-b)(1+\epsilon_4)\cdot(a+b)(1+\epsilon_5)\big)(1+\epsilon_6) \\ &\approx (a^2-b^2)(1+\epsilon_4+\epsilon_5+\epsilon_6)\end{aligned}$$

da cui

$$\epsilon_{\mathrm{alg2}} = \frac{\mathrm{fl}(a^2-b^2)-(a^2-b^2)}{a^2-b^2} \approx \epsilon_4+\epsilon_5+\epsilon_6$$

e $\boxed{I_{\mathrm{alg2}} = 1+1+1 = 3}$

Confronto di algoritmi

Si vuole determinare per quali valori di a e di b l'algoritmo 2 è più stabile dell'algoritmo 1:

$$I_{\mathrm{alg1}} \geqslant I_{\mathrm{alg2}} \quad \Leftrightarrow \quad \frac{a^2}{|a^2-b^2|} + \frac{b^2}{|a^2-b^2|} + 1 \geqslant 3 \quad \Leftrightarrow \quad |a^2-b^2| \leqslant \frac{1}{2}(a^2+b^2)$$

$$\Leftrightarrow \quad \begin{cases} a^2-b^2 \leqslant \dfrac{a^2}{2}+\dfrac{b^2}{2} & \Leftrightarrow \ \dfrac{a^2}{b^2} \leqslant 3 \\ -(a^2+b^2) \leqslant 2(a^2-b^2) & \Leftrightarrow \ \dfrac{a^2}{b^2} \geqslant \dfrac{1}{3} \end{cases} \quad \Leftrightarrow \quad \boxed{\frac{1}{3} \leqslant \frac{a^2}{b^2} \leqslant 3}$$

Esempio

Base 10, $t = 4$, arrotondamento.

$$a = 1.000 \qquad b = 0.9981$$

$a^2/b^2 = 1.003810857501309$. Risultato esatto: 0.00379639.

- $A1 = \mathrm{fl}(a^2) = 1.000$; $B1 = \mathrm{fl}(b^2) = 0.9962$; $\mathrm{fl}(A1 - B1) = 0.0038$
- $A2 = \mathrm{fl}(a+b) = 1.998$; $B2 = \mathrm{fl}(a-b) = 0.0019$; $\mathrm{fl}(A2 \cdot B2) = 0.003796$

Prodotto di n numeri finiti

Siano $x_1, x_2, \ldots, x_n \in F$,

$$\varphi(x_1, x_2, \ldots, x_n) = x_1 x_2 \ldots x_n$$

ALGORITMO

$p = x_1$
for $j = 2$ to n do
  $p = p\, x_j$
$p = \mathrm{fl}(x_1 x_2 \ldots x_n)$

$$\begin{aligned}
\mathrm{fl}(x_1 x_2 \cdots x_n) &= x_1 x_2 (1+\epsilon_2) x_3 (1+\epsilon_3) \cdots x_n (1+\epsilon_n) = x_1 x_2 \cdots x_n (1+\epsilon_2)(1+\epsilon_3)\cdots(1+\epsilon_n) \\
&\approx x_1 x_2 \cdots x_n (1+\epsilon_2+\epsilon_3+\ldots+\epsilon_n) \\
&= x_1 x_2 \cdots x_n + x_1 x_2 \cdots x_n \epsilon_2 + \ldots + x_1 x_2 \cdots x_n \epsilon_n \\
&\qquad \frac{\mathrm{fl}(x_1 x_2 \cdots x_n) - (x_1 x_2 \cdots x_n)}{x_1 \cdots x_n} \approx \epsilon_2 + \ldots + \epsilon_n
\end{aligned}$$

Se $|\epsilon_i| \leqslant u$, $i = 2, \ldots, n$, $|\epsilon_{\mathrm{alg}}| \leqslant (n-1)u$.

$$I_{\mathrm{alg}} = n - 1$$

L'algoritmo è numericamente stabile.

Analisi degli errori

Somma di n numeri di macchina

$$S = \varphi(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$$

Algoritmo:

$s \leftarrow x_1$
for $i = 2 \ldots, n$
  $\lfloor s \leftarrow s + x_i$

$$\begin{aligned}
\mathrm{fl}(S) &= \Big(\ldots\big(\big((x_1+x_2)(1+\epsilon_2)+x_3\big)(1+\epsilon_3)+x_4\big)(1+\epsilon_4)+\ldots+x_n\Big)(1+\epsilon_n) \\
&= (x_1+x_2)(1+\epsilon_2)(1+\epsilon_3)(1+\epsilon_4)\cdots(1+\epsilon_n) + x_3(1+\epsilon_3)(1+\epsilon_4)\cdots(1+\epsilon_n) \\
&\quad + x_4(1+\epsilon_4)\cdots(1+\epsilon_n) + \ldots + x_n(1+\epsilon_n) \\
&\approx (x_1+x_2)(1+\epsilon_2+\epsilon_3+\epsilon_4+\ldots+\epsilon_n) + x_3(1+\epsilon_3+\epsilon_4+\ldots+\epsilon_n) \\
&\quad + x_4(1+\epsilon_4+\ldots+\epsilon_n) + \ldots + x_n(1+\epsilon_n) \qquad \boxed{\epsilon_k \leqslant u \ \forall k = 2, \ldots, n} \\
&= \sum_{i=1}^{n} x_i + (x_1+x_2)\epsilon_2 + (x_1+x_2+x_3)\epsilon_3 + (x_1+x_2+x_3+x_4)\epsilon_4 + \ldots \\
&\qquad \ldots + (x_1+x_2+\ldots+x_n)\epsilon_n = S + \textstyle\sum_{k=2}^{n}\Big(\sum_{j=1}^{k} x_j\Big)\epsilon_k
\end{aligned}$$

$$\epsilon_{\mathrm{alg}} = \frac{\mathrm{fl}(S) - S}{S} \approx \frac{x_1+x_2}{x_1+\ldots+x_n}\epsilon_2 + \frac{x_1+x_2+x_3}{x_1+\ldots+x_n}\epsilon_3 + \ldots + \frac{x_1+\ldots+x_{n-1}}{x_1+\ldots+x_n}\epsilon_{n-1} + \epsilon_n$$

$$I_{\mathrm{alg}} = \left|\frac{x_1+x_2}{x_1+\ldots+x_n}\right| + \left|\frac{x_1+x_2+x_3}{x_1+\ldots+x_n}\right| + \ldots + \left|\frac{x_1+\ldots+x_{n-1}}{x_1+\ldots+x_n}\right| + 1$$

Analisi degli errori

Somma di n numeri di macchina: stabilità dell'algoritmo

- se gli x_i sono di segno concorde, allora

$$|\epsilon_{\text{alg}}| \leqslant (n-1)u \qquad I_{\text{alg}} \leqslant n-1$$

- conviene sempre sommare dal più piccolo al più grande per evitare errori di incolonnamento

Esempio: $\beta = 10$, $t = 7$, arrotondamento, $x_1 = 0.1 \cdot 10^1$, $x_i = 0.1 \cdot 10^{-6}$ per $i = 2, \ldots, 10$.

$$\text{Risultato esatto :} \qquad \sum_{i=1}^{10} x_i = 1.0000009$$

Con l'algoritmo di accumulazione visto prima:

$$\begin{aligned}
&i = 2 \qquad s = \text{fl}(x_1 + x_2) = \text{fl}\left(0.1 \cdot 10^1 + 0.1 \cdot 10^{-6}\right) = 0.1 \cdot 10^1 = y_1 \\
&i = 3 \qquad s = \text{fl}(y_1 + x_3) = \text{fl}\left(0.1 \cdot 10^1 + 0.1 \cdot 10^{-6}\right) = 0.1 \cdot 10^1 = y_2 \\
&\vdots \qquad\qquad \vdots \\
&i = 10 \qquad s = \text{fl}(y_{n-1} + x_n) = \text{fl}\left(0.1 \cdot 10^1 + 0.1 \cdot 10^{-6}\right) = 0.1 \cdot 10^1
\end{aligned}$$

Risultato calcolato : $S = 0.1 \cdot 10^1 = x_1 \qquad \epsilon_{\text{alg}} \approx 8.999\ldots \cdot 10^{-7} \approx 10^{-6}$

Analisi degli errori

Somma di n numeri di macchina: stabilità dell'algoritmo

Se, nell'esempio precedente, procediamo in ordine inverso

```
s ← x_n
for i = n−1, n−2, ..., 1
    s ← s + x_i
```

l'accumulo degli errori è molto più limitato

Con l'algoritmo di accumulazione in ordine inverso:

$$\begin{aligned}
&i = 9 \qquad s = \text{fl}(x_{10} + x_9) = \text{fl}\left(0.1 \cdot 10^{-6} + 0.1 \cdot 10^{-6}\right) = 0.2 \cdot 10^{-6} = z_1 \\
&i = 8 \qquad s = \text{fl}(z_1 + x_8) = \text{fl}\left(0.2 \cdot 10^{-6} + 0.1 \cdot 10^{-6}\right) = 0.3 \cdot 10^{-6} = z_2 \\
&\vdots \qquad\qquad \vdots \\
&i = 2 \qquad s = \text{fl}(z_7 + x_2) = \text{fl}\left(0.8 \cdot 10^{-6} + 0.1 \cdot 10^{-6}\right) = 0.9 \cdot 10^{-6} = z_8 \\
&i = 1 \qquad s = \text{fl}(z_8 + x_1) = \text{fl}\left(0.9 \cdot 10^{-6} + 0.1 \cdot 10^1\right) = 0.100001 \cdot 10^1
\end{aligned}$$

Risultato calcolato : $S = 0.100001 \cdot 10^1 \qquad \epsilon_{\text{alg}} \approx 9.999\ldots \cdot 10^{-8} \approx 10^{-7}$

L'errore algoritmico relativo è 10 volte più piccolo!

Prodotto scalare tra numeri finiti

Siano $\boldsymbol{x} = (x_1, \ldots, x_n)^T$, $\boldsymbol{y} = (y_1, \ldots, y_n)^T$ due vettori di $\mathbb{R}^n$. Il prodotto scalare canonico è dato da

$$P = \boldsymbol{x}^T \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i$$

ALGORITMO

$p = x_1 \cdot y_1$
for $j = 2$ **to** n **do**
  $p = p + x_j \cdot y_j$
end

$\Rightarrow \quad p = \mathrm{fl}(x_1 \cdot y_1 + x_2 \cdot y_2 + \ldots + x_n \cdot y_n)$

$$\begin{aligned}
\mathrm{fl}(P) = \Big(\cdots \Big(\Big(\big(x_1 \cdot y_1 (1+\epsilon_1) + x_2 \cdot y_2 (1+\epsilon_2) \big)(1+\sigma_2) + x_3 \cdot y_3 (1+\epsilon_3) \Big)(1+\sigma_3) \\
+ x_4 \cdot y_4 (1+\epsilon_4) \Big)(1+\sigma_4) + \ldots + x_n \cdot y_n (1+\epsilon_n) \Big)(1+\sigma_n)
\end{aligned}$$

$$\begin{aligned}
= \quad & x_1 \cdot y_1 (1+\epsilon_1)(1+\sigma_2)(1+\sigma_3)\cdots(1+\sigma_n) \\
& + x_2 \cdot y_2 (1+\epsilon_2)(1+\sigma_2)(1+\sigma_3)\cdots(1+\sigma_n) \\
& + x_3 \cdot y_3 (1+\epsilon_3)(1+\sigma_3)\cdots(1+\sigma_n) \\
& + \ldots \\
& + x_n \cdot y_n (1+\epsilon_n)(1+\sigma_n)
\end{aligned}$$

Prodotto scalare tra numeri finiti

Se $P \approx 0$, l'algoritmo è instabile.

Poichè il prodotto scalare è un'operazione molto usata in algebra lineare, si consiglia di convertire i dati dalla semplice precisione alla doppia precisione e di effettuare l'operazione in doppia precisione. Il risultato viene poi convertito alla semplice precisione. In tal modo l'errore di arrotondamento è dell'ordine della precisione di macchina.

Questo procedimento viene usato nelle routines **BLAS1 (Basic Linear Algebra Subprograms)** che implementano le operazioni di base dell'algebra lineare e che sono contenute nelle principali librerie scientifiche.

Analisi degli errori

Esercizio: $p(x) = x^2 + 3x + 2$

Algoritmo 1: $x^2 + 3x + 2$

$$\begin{aligned}
\mathrm{fl}(x^2) &= x^2(1+\epsilon_1)\\
\mathrm{fl}(3x) &= 3x(1+\epsilon_2)\\
\mathrm{fl}(x^2+3x) &= (x^2+3x+x^2\epsilon_1+3x\epsilon_2)(1+\epsilon_3) \approx x^2+3x+x^2\epsilon_1+3x\epsilon_2+(x^2+3x)\epsilon_3\\
\mathrm{fl}(x^2+3x+2) &= (x^2+3x+x^2\epsilon_1+3x\epsilon_2+(x^2+3x)\epsilon_3+2)(1+\epsilon_4)\\
&\approx x^2+3x+2+x^2\epsilon_1+3x\epsilon_2+(x^2+3x)\epsilon_3+(x^2+3x+2)\epsilon_4\\
\Rightarrow \epsilon_{\mathrm{alg1}} &= \frac{x^2}{p(x)}\epsilon_1+\frac{3x}{p(x)}\epsilon_2+\frac{x^2+3x}{p(x)}\epsilon_3+\epsilon_4
\end{aligned}$$

Algoritmo 2: $(x+3)x+2$

$$\begin{aligned}
\mathrm{fl}(x+3) &= (x+3)(1+\eta_1)\\
\mathrm{fl}\big(x(x+3)\big) &= x(x+3)(1+\eta_1)(1+\eta_2) \approx x(x+3)(1+\eta_1+\eta_2)\\
\mathrm{fl}\big(x(x+3)+2\big) &\approx \big(x(x+3)(1+\eta_1+\eta_2)+2\big)(1+\eta_3)\\
&\approx x(x+3)+2+x(x+3)(\eta_1+\eta_2)+\big(x(x+3)+2\big)\eta_3\\
\Rightarrow \epsilon_{\mathrm{alg2}} &= \frac{x(x+3)}{p(x)}\eta_1+\frac{x(x+3)}{p(x)}\eta_2+\eta_3
\end{aligned}$$

Analisi degli errori

Esercizio (segue): $p(x) = x^2 + 3x + 2$

$$I_{\mathrm{alg1}} = \frac{x^2+3|x|+|x^2+3x|}{|p(x)|}+1, \quad I_{\mathrm{alg2}} = 2\frac{|x(x+3)|}{|p(x)|}+1 \qquad \Rightarrow \qquad I_{\mathrm{alg2}} \leqslant I_{\mathrm{alg1}}$$

perchè per le proprietà dei valori assoluti è $|x^2+3x| \leqslant |x^2|+|3x| = x^2+3|x|$.

N.B. Entrambi gli algoritmi possono diventare instabili per valori prossimi alle radici di $p(x)$.

Funzioni non razionali

- funzioni trigonometriche, logaritmiche, espinenziali
- vengono approssimate mediante una successione di operazioni algebriche elementari (serie di Taylor troncate)
- diventa importante saper valutare accuratamente i polinomi

Analisi degli errori

Valutazione di un polinomio generico: algoritmo 1

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

```
p ← 1
s ← a_0
for i = 1, 2, ..., n
    p ← p · x
    s ← p · a_i + s
```

Complessità computazionale: $2n$ moltiplicazioni + n addizioni

Valutazione di un polinomio generico: algoritmo 2 (schema di Ruffini-Horner)

$$p_n(x) = \Big(\ldots\big(\big((a_n x + a_{n-1})x + a_{n-2}\big)x + a_{n-3}\big)x + \ldots + a_1\Big)x + a_0$$

Esempio: $p_3(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0 = ((a_3 x + a_2)x + a_1)x + a_0$

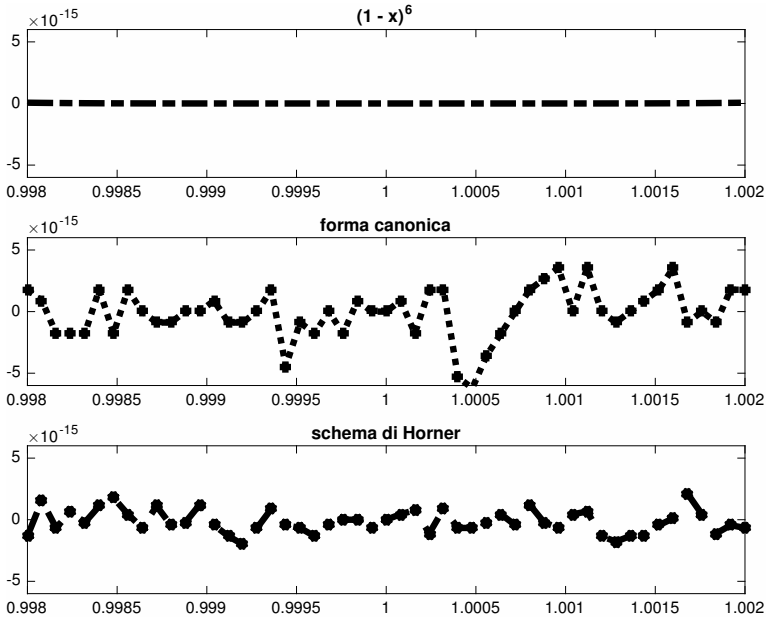
```
s ← a_n
for i = n − 1, n − 2, ..., 0
    s ← s · x + a_i
```

Complessità computazionale: n moltiplicazioni + n addizioni

Analisi degli errori

Comportamento instabile

$$\begin{aligned} p_6(x) &= (1-x)^6 \\ &= x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1 \\ &= \Big(\Big(\big(\big((x-6)x + 15\big)x - 20\big)x + 15\Big)x - 6\Big) + 1 \end{aligned}$$



Calcolo del polinomio $(x-1)^6$

```
clear all;
x = linspace(0.998,1.002,50);
p = [1 -6 15 -20 15 -6 1];
y1 = (x-1).^6;
y2 = polyval(p,x);
y3 = x.^6 - 6*x.^5 + 15*x.^4-20*x.^3+15*x.^2-6*x+1;
plot(x,y1,'-*',x,y2,':+',x,y3,'--x');
legend('(x-1)^6','Horner','sviluppo');
fprintf('x\t (x-1)^6\t  Horner \t sviluppo \n');
for i = 1:3:length(x);
    fprintf('%7.4e\t %6.3e\t %6.3e\t ...
             %6.3e  \n', x(i), y1(i),y2(i),y3(i));
end
```


Analisi degli errori sui dati iniziali

Errore inerente

Esempio: soluzione di equazioni di secondo grado
Dato x, trovare y tale che

$$y^2 - 4y + x = 0$$

Esprimiamo le soluzioni del problema in funzione del dato x:

$$\begin{aligned} & y_{1,2} = 2 \pm \sqrt{4-x} \\ x = 4 \implies & y_{1,2} = 2 \\ x = 4 - 10^{-6} \implies & y_{1,2} = 2 \pm 10^{-3} \end{aligned}$$

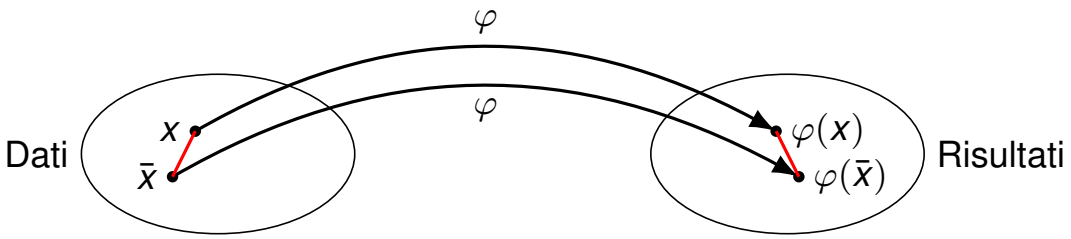
Dunque, a piccole variazioni sui dati (10^{-6}) corrispondono variazioni molto più grandi sui risultati (10^{-3}).

Questo è un esempio di problema mal condizionato.

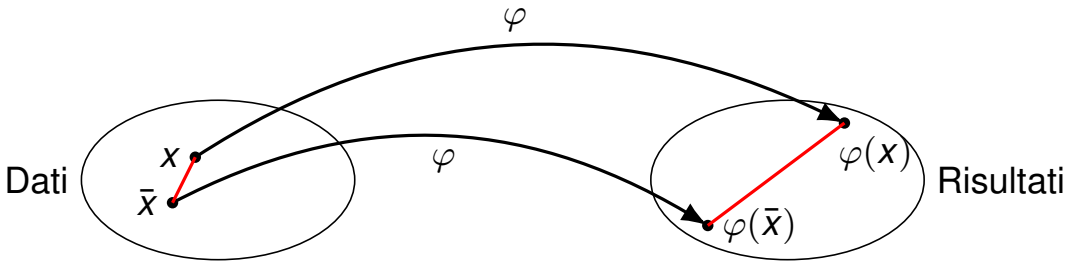
Si ha una enorme amplificazione dell'errore sul dato iniziale. **E' un errore connesso al problema, non all'algoritmo risolutivo.**

Rappresentazione grafica del condizionamento

Problema ben condizionato



Problema mal condizionato



Errore inerente

- Supponiamo che il risultato y che stiamo calcolando sia legato ai dati x dalla funzione φ: $y = \varphi(x)$
- Cerchiamo una stima della variazione che si produce sul risultato a partire da due dati diversi

Indice di condizionamento

Se x e $\bar{x}$ appartengono all'insieme dei dati, allora possiamo scrivere:

$$\bar{x} = x + \delta \;\Rightarrow\; \delta = \bar{x} - x \quad \Longrightarrow \quad \frac{\delta}{x} = \frac{\bar{x} - x}{x} = \epsilon_x$$

$$\epsilon_{\text{dati}} = \frac{\varphi(x+\delta) - \varphi(x)}{\varphi(x)} \approx \frac{\varphi(x) + \varphi'(x)\delta - \varphi(x)}{\varphi(x)}$$

$$= \frac{\varphi'(x)\delta}{\varphi(x)} = \boxed{\frac{\varphi'(x)x}{\varphi(x)}}\,\epsilon_x \qquad I_{\text{cond}} = \left|\frac{\varphi'(x)x}{\varphi(x)}\right|$$

I_{cond} = indice di condizionamento = fattore di amplificazione degli errori sui dati

Se $I_{\text{cond}} \gg 1$ allora il problema è mal condizionato; se invece I_{cond} è vicino a 1, allora il problema è ben condizionato.

Analisi degli errori sui dati iniziali

Esempio (segue): $y^2 - 4y + x = 0 \quad y_1(x) = 2 + \sqrt{4 - x}$

$$\varphi(x) = 2 + \sqrt{4 - x} \qquad \varphi'(x) = -\frac{1}{2\sqrt{4 - x}}$$

$$I_{\text{cond}} = \left| -\frac{x}{2\sqrt{4 - x}\left(2 + \sqrt{4 - x}\right)} \right| \xrightarrow[x \to 4]{} \infty$$

ne segue che il problema è mal condizionato per valori di x prossimi a 4

Esempio

$$\begin{cases} x + \alpha y &= 1 \\ \alpha x + y &= 0 \end{cases}$$

Le soluzioni sono $x(\alpha) = \dfrac{1}{1 - \alpha^2}$ e $y(\alpha) = \dfrac{-\alpha}{1 - \alpha^2}$.

Per α^2 prossimo a 1, il problema è mal condizionato, mentre è ben condizionato per α^2 distante da 1. Infatti,

$$\frac{x(\alpha + \delta) - x(\alpha)}{x(\alpha)} \approx \frac{x'(\alpha)\alpha}{x(\alpha)} \cdot \frac{\delta}{\alpha} = \frac{2\alpha^2}{1 - \alpha^2} \cdot \frac{\delta}{\alpha}$$

Un altro problema: calcolo di

$$z = x + y = \frac{1}{1 - \alpha^2} - \frac{\alpha}{1 - \alpha^2} = \frac{1}{1 + \alpha}$$

Il calcolo è ben condizionato per $\alpha \approx 1$ e mal condizionato per $\alpha \approx -1$.

$$\frac{z(\alpha + \delta) - z(\alpha)}{z(\alpha)} \approx \frac{-\alpha}{1 + \alpha} \frac{\delta}{\alpha}$$

Tuttavia per $\alpha = .99$, con $\beta = 10$, $t = 4$ e arrotondamento:

$$\text{fl}(z) = \text{fl}\left(\frac{1}{.0199} - \frac{.99}{.0199}\right) = 50.25 - 49.75 = 0.5, \quad \text{mentre} \begin{cases} z = 0.5025125 \\ \epsilon_{\text{tot}} = 4.999 \cdot 10^{-3} \end{cases}$$

Esempio

Ciò è dovuto all'amplificazione dell'errore fatta dalla cancellazione. L'algoritmo è **instabile**.

Invece, con la stessa aritmetica finita:

$$\mathrm{fl}\left(\frac{1}{1+\alpha}\right) = 0.5025$$

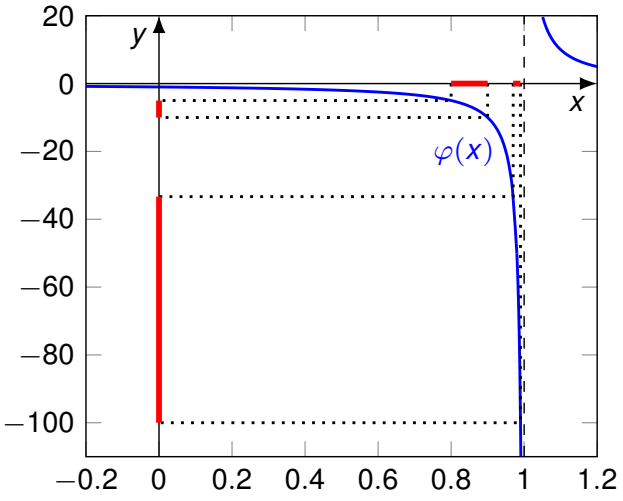
con un errore $\epsilon_{\text{tot}} \approx 2.49 \cdot 10^{-5}$.

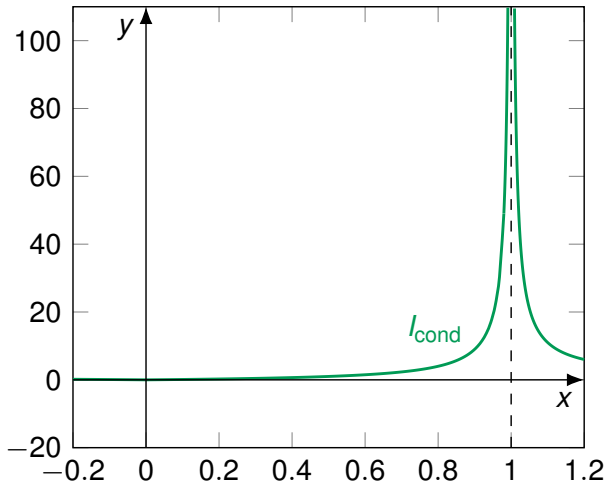
Analisi degli errori sui dati iniziali

Indice di condizionamento

Esempio: si consideri l'espressione $\varphi(x) = \dfrac{1}{x-1}$

$$I_{\text{cond}} = \left|\frac{\varphi'(x)x}{\varphi(x)}\right| = \left|\frac{-x}{x-1}\right|$$





$I_{\text{cond}}(0.90) = 9.0 \cdot 10^{0}$, $I_{\text{cond}}(0.99) = 9.9 \cdot 10^{1}$

vicino ad 1 si ha mal condizionamento, mentre lontano da 1 si ha buon condiz.

Analisi dell'errore sui dati iniziali

Gli esempi mostrano che:

- un problema può essere ben condizionato per certi valori e mal condizionato per altri;
- il condizionamento può essere espresso mediante la derivata della trasformazione;
- un problema ben condizionato può dare risultati non buoni se risolto con algoritmi instabili;
- un algoritmo è stabile o meno a seconda del condizionamento delle trasformazioni che lo compongono.

Analisi dell'errore sui dati iniziali

Si vuole calcolare $y = \varphi(x)$.
Tuttavia x è affetto da un errore Δx, per cui si calcola

$$\varphi(x + \Delta x) = y + \Delta y$$

Se φ è derivabile, facendo un'analisi al I ordine, si ottiene:

$$y + \Delta y = \varphi(x + \Delta x) \approx \varphi(x) + \varphi'(x)\Delta x \quad \Rightarrow \quad E_{\text{dati}} = \Delta y \approx \varphi'(x)\Delta x$$

Se $\varphi(x) \neq 0$, $x \neq 0$, segue

$$\epsilon_{\text{dati}} = \frac{\Delta y}{y} \approx \frac{\varphi'(x)x}{\varphi(x)} \frac{\Delta x}{x} = \frac{\varphi'(x)x}{\varphi(x)} \epsilon_x$$

L'errore $\epsilon_x = \dfrac{\Delta x}{x}$ produce un errore sui risultati finali $\epsilon_{\text{dati}} = \dfrac{\Delta y}{y}$ amplificato di una quantità detta **indice di condizionamento**:

$$I_{\text{cond}} = K(x, \varphi) = \left| \frac{\varphi'(x)}{\varphi(x)} x \right|$$

Se $K(x, \varphi) \gg 1$, il problema è mal condizionato; se $K(x, \varphi)$ è piccolo, il problema è ben condizionato.
Esempio. $y = \varphi(x) = \log(x)$.

$$\epsilon_{\text{dati}} = \frac{1}{\log(x)} \epsilon_x \quad I_{\text{cond}} = \left| \frac{1}{\log(x)} \right| \quad \Rightarrow \quad \text{se } x \approx 1 \text{ problema mal condizionato.}$$

Analisi degli errori

Osservazione: un problema può essere ben condizionato per un insieme di valori e mal condizionato per altri.

Caso generale: dipendenza da n dati $y = \varphi(x_1, \ldots, x_n)$, $x_i \in \mathbb{R}$ $i = 1, \ldots, n$

Avendo n dati, si hanno anche n errori di rappresentazione:

$$x_i - \bar{x}_i = \delta_i \ll x_i \quad \Rightarrow \quad \epsilon_{x_i} = \frac{\delta_i}{x_i}, \quad i = 1, \ldots, n$$

Per l'errore assoluto e quello relativo si ottengono le stime al prim'ordine

$$E_{\text{dati}} = \varphi(\bar{x}_1, \ldots, \bar{x}_n) - \varphi(x_1, \ldots, x_n) = \varphi(x_1 + \delta_1, \ldots, x_n + \delta_n) - \varphi(x_1, \ldots, x_n)$$
$$\approx \sum_{i=1}^{n} \delta_i \frac{\partial \varphi}{\partial x_i}(x_1, \ldots, x_n)$$

$$\epsilon_{\text{dati}} = \frac{\varphi(\bar{x}_1, \ldots, \bar{x}_n) - \varphi(x_1, \ldots, x_n)}{\varphi(x_1, \ldots, x_n)} \approx \sum_{i=1}^{n} \boxed{\frac{\partial\big(\varphi(x_1, \ldots, x_n)\big)}{\partial x_i} \frac{x_i}{\varphi(x_1, \ldots, x_n)}} \epsilon_{x_i}$$

fattori di amplificazione degli errori sui singoli dati

$$I_{\text{cond}} = \boxed{\sum_{i=1}^{n} \left| \frac{\partial\big(\varphi(x_1, \ldots, x_n)\big)}{\partial x_i} \frac{x_i}{\varphi(x_1, \ldots, x_n)} \right|}$$

indice di condizionamento

Analisi degli errori

Condizionamento delle operazioni elementari

$$\epsilon_{x \pm y} = \frac{x}{x \pm y}\epsilon_x \pm \frac{y}{x \pm y}\epsilon_y$$
$$\epsilon_{xy} = \epsilon_x + \epsilon_y$$
$$\epsilon_{x/y} = \epsilon_x - \epsilon_y$$
$$\epsilon_{\sqrt{x}} = \frac{1}{2}\epsilon_x$$
$$\epsilon_{x^\alpha} = \alpha \epsilon_x$$

Sono ben condizionate: moltiplicazione, divisione, radice e potenza con $|\alpha|$ piccolo.

Se $x \approx y$ la sottrazione è mal condizionata perché può verificarsi il fenomeno di cancellazione

Operazioni elementari

- Somma di due numeri: $\varphi(x, y) = x + y$

$$\frac{\partial \varphi(x,y)}{\partial x} = 1 \qquad \frac{\partial \varphi(x,y)}{\partial y} = 1$$

$$\epsilon_{x+y} = \frac{\partial \varphi(x,y)}{\partial x}\frac{x}{x+y}\epsilon_x + \frac{\partial \varphi(x,y)}{\partial y}\frac{y}{x+y}\epsilon_y = \frac{x}{x+y}\epsilon_x + \frac{y}{x+y}\epsilon_y$$

- Prodotto di due numeri: $\varphi(x, y) = xy$

$$\frac{\partial \varphi(x,y)}{\partial x} = y \qquad \frac{\partial \varphi(x,y)}{\partial y} = x$$

$$\epsilon_{x+y} = \frac{\partial \varphi(x,y)}{\partial x}\frac{x}{xy}\epsilon_x + \frac{\partial \varphi(x,y)}{\partial y}\frac{y}{xy}\epsilon_y = y\frac{x}{xy}\epsilon_x + x\frac{y}{xy}\epsilon_y = \epsilon_x + \epsilon_y$$

Esempi:[<+->]

- $\varphi(x) = x - \alpha \quad I_{\text{cond}} = \left|\frac{x}{x-\alpha}\right|$
  mal condizionato vicino ad α
- $\varphi(x) = e^x \quad I_{\text{cond}} = |x|$
  mal condizionato per $|x|$ grande, ben condizionato per $|x| < 1$
- $\varphi(x) = \sqrt{x^2+1} \quad I_{\text{cond}} = \left|\frac{x}{x^2+1}\right|$
  sempre ben condizionato
- $\varphi(x) = \log(x) \quad I_{\text{cond}} = \left|\frac{1}{\log(x)}\right| \xrightarrow[x\to 1]{} +\infty$
  mal condizionato per x vicino a 1
- $\varphi(a, b, c) = a + b + c = y \quad \epsilon_y = \frac{a}{a+b+c}\epsilon_a + \frac{b}{a+b+c}\epsilon_b + \frac{c}{a+b+c}\epsilon_c$
  $I_{\text{cond}} = \left|\frac{a}{a+b+c}\right| + \left|\frac{b}{a+b+c}\right| + \left|\frac{c}{a+b+c}\right|$
  ben condizionato se a, b e c sono di segno concorde
- $\varphi(a, b) = a^2 - b^2 = y \quad \epsilon_y = \frac{2a^2}{a^2-b^2}\epsilon_a - \frac{2b^2}{a^2-b^2}\epsilon_b$
  $I_{\text{cond}} = \frac{2a^2}{|a^2-b^2|} + \frac{2b^2}{|a^2-b^2|}$ mal condizionato se $a^2 \approx b^2$

Il metodo del grafo per l'analisi degli errori

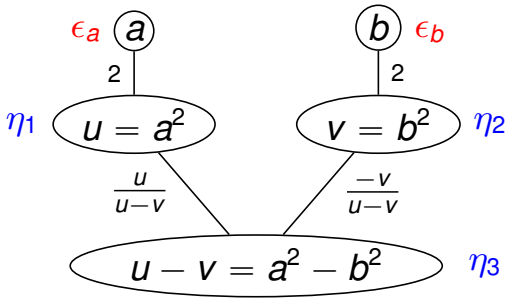
Il grafo si costruisce nel modo seguente:

- ogni nodo contiene un operando: i nodi iniziali contengono dati iniziali (variabili o costanti), i nodi figli contengono un risultato intermedio e indicano l'operazione avvenuta per ottenerlo
- si pone accanto ad ogni nodo l'errore relativo: nei nodi iniziali esso è l'errore inerente ϵ_i, cioè dovuto alla rappresentazione del dato ($\mathrm{fl}(x)$), nei nodi figli esso è l'errore algoritmico η_j, cioè quello dovuto all'operazione in aritmetica finita ($\mathrm{fl}(\varphi(x))$)
- su ogni arco si indica l'indice di condizionamento della corrispondente trasformazione, che costituisce il fattore di amplificazione degli errori accumulati negli operandi fino a quel punto
- il contributo sull'errore relativo totale ϵ_{tot} di ogni errore ϵ_i, η_j si ottiene sommando i prodotti dei fattori di amplificazione su ogni percorso che unisce ciascun nodo a quello del risultato finale

Osservazione: spesso, ma non sempre, il grafo è un albero, che viene costruito e visitato a rivescio, cioè dalle foglie alla radice: le foglie sono i dati iniziali, la radice è il risultato finale.

Esempio: differenza di quadrati

Algoritmo 1: $\varphi(a,b) = a^2 - b^2$

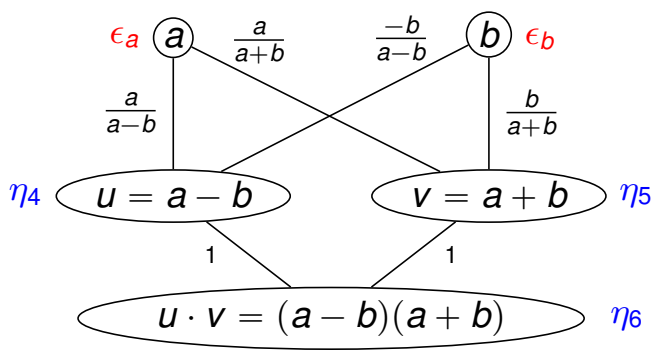


$$\epsilon_{\mathrm{tot}} = 2\frac{u}{u-v}\epsilon_a + 2\frac{-v}{u-v}\epsilon_b + \frac{u}{u-v}\eta_1 + \frac{-v}{u-v}\eta_2 + \eta_3$$

$$= 2\frac{a^2}{a^2-b^2}\epsilon_a + 2\frac{-b^2}{a^2-b^2}\epsilon_b + \frac{a^2}{a^2-b^2}\eta_1 + \frac{-b^2}{a^2-b^2}\eta_2 + \eta_3$$

$$K_{\mathrm{cond}} = 2\frac{a^2+b^2}{|a^2-b^2|} \qquad \boxed{K_{\mathrm{alg1}} = \frac{a^2+b^2}{|a^2-b^2|} + 1}$$

Algoritmo 2: $\varphi(a,b) = (a-b)(a+b)$



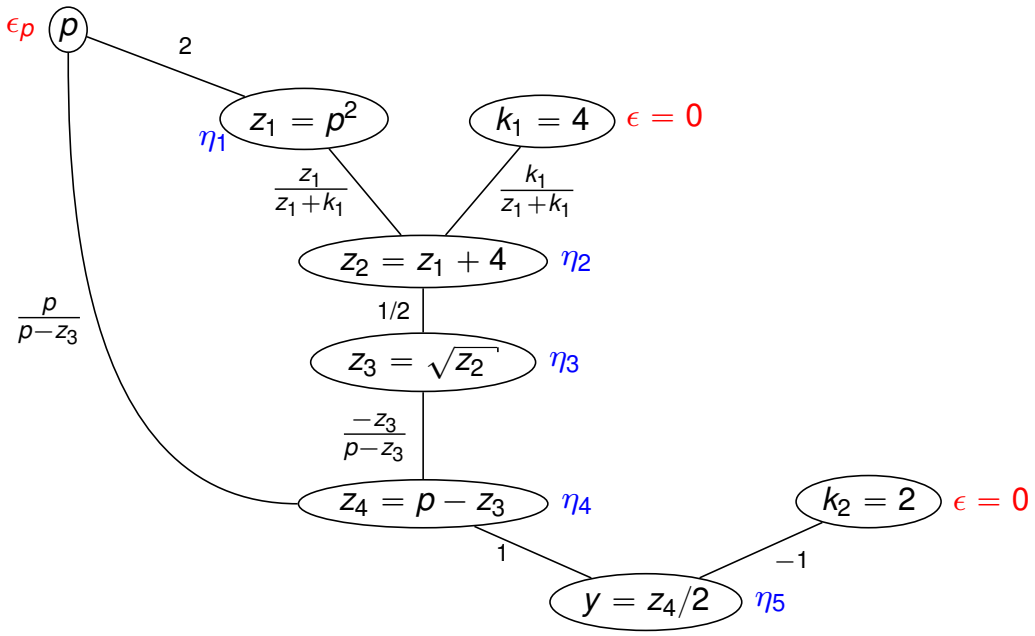
$$\epsilon_{\mathrm{tot}} = \left(\frac{a}{a+b} + \frac{a}{a-b}\right)\epsilon_a + \left(\frac{b}{a+b} - \frac{b}{a-b}\right)\epsilon_b + \eta_4 + \eta_5 + \eta_6$$

$$= 2\frac{a^2}{a^2-b^2}\epsilon_a + 2\frac{-b^2}{a^2-b^2}\epsilon_b + \eta_4 + \eta_5 + \eta_6$$

$$K_{\mathrm{cond}} = 2\frac{a^2+b^2}{|a^2-b^2|} \qquad \boxed{K_{\mathrm{alg2}} = 3}$$

Esercizio: $\varphi(p) = \dfrac{p - \sqrt{p^2+4}}{2}, \quad p \geqslant 0$

Algoritmo 1: si procede valutando la radice, poi la differenza a numeratore e infine dividendo per 2



Esercizio (continua):

$$\epsilon_{\text{tot}}^{\text{alg1}} = \left(\frac{p}{p-z_3} + 2\,\frac{z_1}{z_1+4}\,\frac{1}{2}\,\frac{-z_3}{p-z_3}\right)\epsilon_p + \left(\frac{z_1}{z_1+4}\,\frac{1}{2}\,\frac{-z_3}{p-z_3}\right)\eta_1 + \left(\frac{1}{2}\,\frac{-z_3}{p-z_3}\right)\eta_2 - \frac{z_3}{p-z_3}\eta_3 + \eta_4 + \eta_5$$

$$= \frac{-p}{\sqrt{p^2+4}}\,\epsilon_p + \frac{p^2\left(p+\sqrt{p^2+4}\right)}{8\sqrt{p^2+4}}\,\eta_1 + \frac{1}{8}\left(p\sqrt{p^2+4} + p^2 + 4\right)\eta_2 + \frac{1}{4}\left(p\sqrt{p^2+4} + p^2 + 4\right)\eta_3 + \eta_4 + \eta_5$$

$$I_{\text{cond}} = \frac{p}{\sqrt{p^2+4}} \Rightarrow \text{problema ben condizionato}$$

$$I_{\text{alg1}} = \frac{p^2\left(p+\sqrt{p^2+4}\right)}{8\sqrt{p^2+4}} + \frac{3}{8}\left(p\sqrt{p^2+4} + p^2 + 4\right) + 2$$

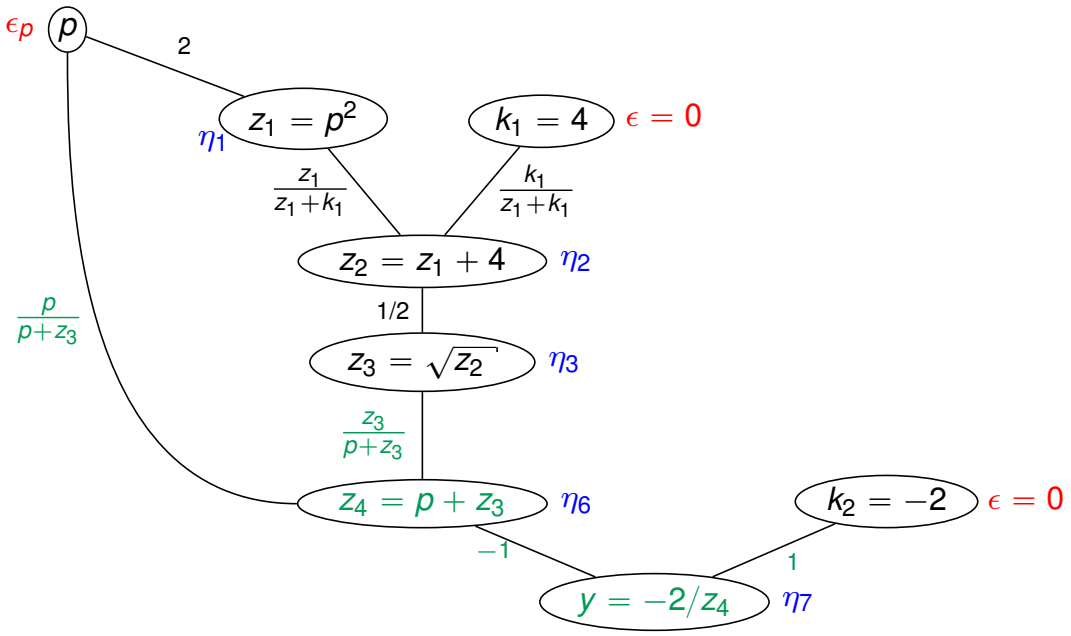
$\Rightarrow$ se $p \gg$, l'algoritmo è instabile

Infatti, se $p \gg$, allora $\sqrt{p^2+4} \propto p$ e dunque $I_{\text{alg1}} \propto p^2$

Per rendere stabile l'algoritmo si riformula il problema mediante la tecnica della razionalizzazione

Esercizio (continua): $\varphi(p) = \dfrac{-2}{p + \sqrt{p^2+4}}, \quad p \geqslant 0$

Algoritmo 2: si procede valutando la radice, poi la somma a denominatore e infine il quoziente



Esercizio (continua):

$$\epsilon_{\text{tot}}^{\text{alg2}} = \frac{-p}{\sqrt{p^2+4}}\,\epsilon_p + \frac{p^2\left(p - \sqrt{p^2+4}\right)}{8\sqrt{p^2+4}}\,\eta_1 + \frac{1}{8}\left(p\sqrt{p^2+4} - (p^2+4)\right)\eta_2$$
$$+ \frac{1}{4}\left(p\sqrt{p^2+4} - (p^2+4)\right)\eta_3 - \eta_6 + \eta_7$$

$$I_{\text{cond}} = \frac{p}{\sqrt{p^2+4}} \Rightarrow \text{l'indice di condizionamento del problema non cambia}$$

$$I_{\text{alg2}} = \frac{p^2\left|p - \sqrt{p^2+4}\right|}{8\sqrt{p^2+4}} + \frac{3}{8}\left|p\sqrt{p^2+4} - (p^2+4)\right| + 2$$

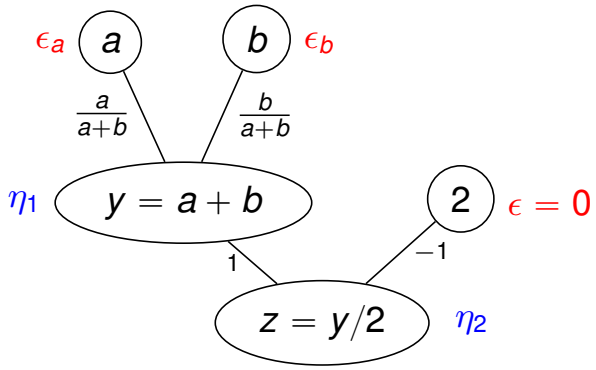
$\Rightarrow$ se $p \gg$, l'algoritmo è più stabile

Infatti, se $p \gg$ si vede che sia il primo che il secondo addendo di I_{alg2} rimangono limitati, invece di crescere come p^2 come accade per I_{alg1}

Esempio

$$\varphi(a,b) = \frac{a+b}{2}.$$

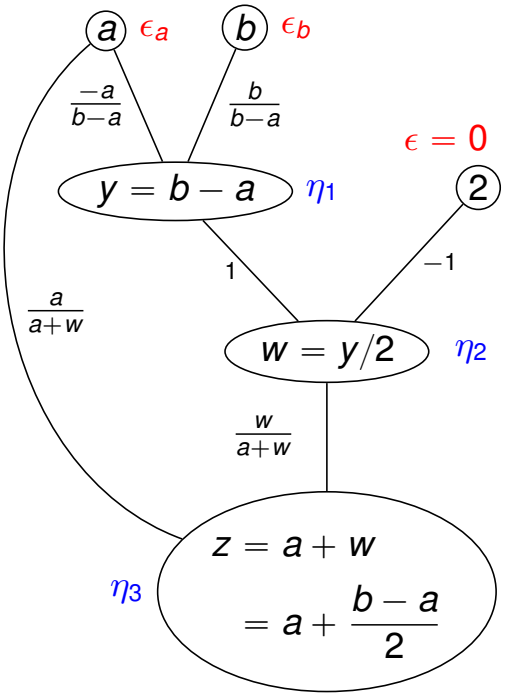
Algoritmo 1



$$\epsilon_{\text{tot}} = \frac{a}{a+b}\epsilon_a + \frac{b}{a+b}\epsilon_b + \eta_1 + \eta_2 \qquad I_{\text{cond}} = \left|\frac{a}{a+b}\right| + \left|\frac{b}{a+b}\right| \qquad I_{\text{alg1}} = 2$$

Esempio

Algoritmo 2



$$\epsilon_{\text{tot}} = \frac{a}{a+b}\epsilon_a + \frac{b}{a+b}\epsilon_b + \frac{b-a}{a+b}\eta_1 + \frac{b-a}{a+b}\eta_2 + \eta_3$$

$$I_{\text{cond}} = \left|\frac{a}{a+b}\right| + \left|\frac{b}{a+b}\right| \qquad I_{\text{alg2}} = 2\left|\frac{b-a}{a+b}\right| + 1$$

Il problema è mal condizionato se $a \approx -b$. Se a e b hanno lo stesso segno, allora è ben condizionato. Inoltre,

$$I_{\text{alg1}} > I_{\text{alg2}} \quad \text{se} \quad 2 > 2\left|\frac{b-a}{b+a}\right| + 1$$

$$\Downarrow$$

$$\frac{1}{2} > \left|\frac{b-a}{b+a}\right|$$

Se a e b sono positivi, allora $a + b > 0$ e si ha:

$$\frac{a+b}{2} > |b-a|$$

$$b - a < \frac{a+b}{2} \Rightarrow \frac{1}{3} < \frac{a}{b}; \quad -\frac{a+b}{2} < b - a \Rightarrow \frac{a}{b} < 3$$

Alg2 è numericamente più stabile di Alg1 se $\frac{1}{3} \leqslant \frac{a}{b} < 3$.

È un piccolo errore?

UN DISASTRO. . . NUMERICO

Il 25 febbraio del 1991, durante la Guerra del Golfo, una batteria di missili Patriot americana fallì l'intercettazione di un missile Scud iracheno. Lo Scud colpì una postazione di soldati americani uccidendone 28.
Un report del General Accounting Office, GAO/IMTEC-92-26 , segnalò il motivo del malfunzionamento: una inaccuratezza nel calcolo del tempo.
All'interno del sistema il tempo era misurato in decimi di secondo, iniziando a contare dalla messa in funzione della batteria. Per convertire il tempo trascorso T misurato in decimi di secondo in secondi e conoscere la posizione del Patriot si fece

$$\mathrm{fl}\left(\frac{1}{10}\cdot T\right)$$

Ma $\frac{1}{10}$ in binario ha rappresentazione periodica e dunque viene troncato; in quel caso l'errore commesso era $9.5\cdot 10^{-8}$. Quando successe il disastro, la batteria di Patriot era in funzione da circa 100 ore, ossia da $T = 100\times 60\times 60\times 10 = 3600000$ decimi di secondo, comportando un errore pari a

$$9.5\cdot 10^{-8}\cdot 3600000 = 0.34 \text{ secondi}$$

Uno Scud viaggia a circa 1676 metri al secondo, circa 6034 Km/h, quindi in 0.34 secondi precorre un tragitto di circa mezzo chilometro (569.8 m). Questo spazio era abbastanza per portare lo Scud fuori dal "range gate" che il Patriot monitorava.