Calcolo Numerico

Aritmetica di macchina

Simone Rebegoldi

Corso di Laurea in Informatica
Dipartimento di Scienze Fisiche, Informatiche e Matematiche

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Optimization Algorithms
and Software for
Inverse problemS

www.oasis.unimore.it

1. La rappresentazione dei numeri

Distinguiamo il concetto di quantità espressa da un numero e la sua rappresentazione.

NUMERO

↙ ENTITA' ASTRATTA

"sette"

univocamente determinata

↘ RAPPRESENTAZIONE

(7, VII, 111_2, ...)

molteplice a seconda dei criteri di rappresentazione adottati.

La convenzione usata per esprimere un numero è detta notazione posizionale.

Cifre	7	5	9	$=$	$7 \cdot 10^2 + 5 \cdot 10^1 + 9 \cdot 10^0$
Posizione	2	1	0		
Base	10				

- La notazione posizionale decimale è dunque caratterizzata da una *base* $\beta = 10$, dalle posizioni delle cifre che costituiscono gli *esponenti* della base e da un insieme di *simboli*, le cifre $0, 1, 2, 3, ...$, ciascuna delle quali indica un valore compreso tra "zero" e "nove".

- Mantenendo la convenzione posizionale, possiamo estendere il discorso per qualsiasi base β. Ad ogni valore di β dovremo definire un insieme ordinato $\mathcal{S}$ di simboli associati ai valori compresi tra 0 e $\beta - 1$.

BASE	SIMBOLI
2	$\mathcal{S} = \{0, 1\}$
8	$\mathcal{S} = \{0, 1, 2, 3, 4, 5, 6, 7\}$
16	$\mathcal{S} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

- L'insieme dei simboli contiene le cifre corrispondenti ai valori $0, 1, \ldots, \beta - 1$.

- Fissata una base β, ogni numero naturale si può quindi scrivere come una stringa di cifre

$$(c_p c_{p-1} \ldots c_1 c_0)_\beta$$

dove $p \in \mathbb{N}$.

- Se con un piccolo abuso di notazione, indichiamo con c_i anche il valore della cifra e identifichiamo il numero con la sua rappresentazione (unica), allora il valore del numero espresso in notazione posizionale è

$$N = (c_p c_{p-1} \ldots c_1 c_0)_\beta = c_p \beta^p + c_{p-1} \beta^{p-1} + \ldots + c_1 \beta + c_0.$$

$$N = (c_p c_{p-1} \ldots c_1 c_0)_\beta = c_p \beta^p + c_{p-1} \beta^{p-1} + \ldots + c_1 \beta + c_0.$$

Supponiamo $c_p \neq 0$.
Dato $t \leq p$,

- le t cifre meno significative della rappresentazione in base β del numero $N \in \mathbb{N}$ sono i pesi delle potenze più piccole della base:

$$(c_p c_{p-1} \ldots c_t \underbrace{c_{t-1} c_1 c_0})_\beta;$$

  tali pesi si contano da destra verso sinistra;

- le t cifre più significative della rappresentazione in base β del numero $N \in \mathbb{N}$ sono i pesi delle potenze più grandi della base:

$$(\underbrace{c_p c_{p-1} \ldots c_{p-t+1}} \ldots c_1 c_0)_\beta;$$

  si contano da sinistra partendo dalla prima cifra non nulla della rappresentazione.

”trecentosettantadue”=

$$\begin{aligned}(372)_{10} &= 3\cdot 10^2 + 7\cdot 10^1 + 2\cdot 10^0\\ (174)_{16} &= 1\cdot 16^2 + 7\cdot 16^1 + 4\cdot 16^0\\ (564)_{8} &= 5\cdot 8^3 + 6\cdot 8^1 + 4\cdot 8^0\\ (101110100)_2 &= 1\cdot 2^8 + 0\cdot 2^7 + 1\cdot 2^6 + 1\cdot 2^5 +\\ &\quad +1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0\end{aligned}$$

”duecentoottantasette”=

$$(287)_{10} = (100011111)_2 = (10133)_4$$
$$(437)_8 = (11F)_{16} = (8V)_{32}.$$

Osservazione

All’aumentare della base, aumenta il numero dei simboli. Al contrario, con una base più grande sono in genere necessarie meno cifre per rappresentare uno stesso valore.

La rappresentazione posizionale di numerali reali < 1 in base β

- Un numero $\alpha \in \mathbb{R}$, $0 \leq \alpha < 1$, si rappresenta in una base fissata $\beta > 1$ come

$$(0.a_1 a_2 ... a_t a_{t+1} ...)_\beta = a_1 \beta^{-1} + a_2 \beta^{-2} + \ldots + a_t \beta^{-t} + a_{t+1} \beta^{t+1} + \ldots$$
$$= \sum_{i=1}^{\infty} a_i \beta^{-i}$$

  dove "." è il punto radice e a_i sono cifre appartenenti all'insieme dei simboli associato alla base β.
- Lo zero a sinistra del punto radice può essere omesso.
- Ogni cifra a destra del punto radice è il peso di una potenza negativa della base.
- Per rappresentare certi valori (numeri periodici, numeri irrazionali) è necessario un numero infinito di cifre.

ESEMPI:

β	rappresentazione	valore
10	0.1	$1/10$
10	0.11	$1/10 + 1/100$
2	0.1	$1/2$
2	0.11	$1/2 + 1/4$
16	0.1	16^{-1}

Un numero reale α si ottiene come la somma della sua parte intera più la sua parte frazionaria. Mettendo insieme i due contributi e tenendo conto del segno si ha

$$\begin{aligned}\alpha &= \text{segno}(\alpha)(a_1 a_2 ... a_p . a_{p+1} a_{p+2} a_{p+3} ...)_\beta \\ &= \text{segno}(\alpha)(a_1 \beta^{p-1} + a_2 \beta^{p-2} + ... + a_p + a_{p+1}\beta^{-1} + a_{p+2}\beta^{-2} + ...).\end{aligned}$$

Raccogliendo un fattore β^p tra tutti i termini, si può scrivere

$$\begin{aligned}\alpha &= \text{segno}(\alpha)(a_1 \beta^{-1} + a_2 \beta^{-2} + a_3 \beta^{-3} + ...)\beta^p \\ &= \text{segno}(\alpha) \sum_{i=1}^{\infty} (a_i \beta^{-i}) \beta^p \\ &= \text{segno}(\alpha) \sum_{i=1}^{\infty} m \beta^p\end{aligned}$$

dove

- segno vale $+1$ o -1 a seconda del segno di α;
- $m = \sum_{i=1}^{\infty} (a_i \beta^{-i}) = (0.a_1 a_2 ...)_\beta$ è un numero reale < 1 chiamato mantissa;
- β^p è un intero chiamato parte esponente di α.

- Le rappresentazioni del tipo segno-mantissa-esponente

$$\alpha = \text{segno}(\alpha)(0.a_1a_2...)\beta^p$$

si dicono in *forma scientifica*; se $a_1 \neq 0$, si parla di forma scientifica *normalizzata*.

ESEMPI:

$(372)_{10}$	mista
$.372\ 10^3$	normalizzata
$.0372\ 10^4$	scientifica
$(3.141592...)_{10}$	mista
$.3141592\ 10^1$	normalizzata
$.3243F...\ 16^1$	normalizzata
$(3.243F...)_{16}$	mista

- Se un numero è rappresentato nella forma scientifica normalizzata, l'unicità della rappresentazione è garantita.

- Serve per convertire un intero positivo α da base 10 ad una diversa base β.
- L'algorimo consiste nell'eseguire la divisione intera (quoziente e resto) del numero α per β finchè non si giunge ad un quoziente nullo.

Esempio

Rappresentazione del numero $(1972)_{10}$ in base 2.

base 2

$$
\begin{array}{ll}
1972 : 2 = 986 & \text{resto } 0 \\
986 : 2 = 493 & \text{resto } 0 \\
493 : 2 = 246 & \text{resto } 1 \\
246 : 2 = 123 & \text{resto } 0 \\
123 : 2 = 61 & \text{resto } 1 \\
61 : 2 = 30 & \text{resto } 1 \\
30 : 2 = 15 & \text{resto } 0 \\
15 : 2 = 7 & \text{resto } 1 \\
7 : 2 = 3 & \text{resto } 1 \\
3 : 2 = 1 & \text{resto } 1 \\
1 : 2 = 0 & \text{resto } 1
\end{array}
$$

$$(1972)_{10} = (11110110100)_2$$

Algoritmo delle moltiplicazioni successive

- Serve per rappresentare un reale decimale $\alpha \in [0, 1)$ in una base $\beta > 1$.
- Si tratta di determinare le cifre della rappresentazione

$$\alpha = (.a_1 a_2 a_3 ...)_\beta = a_1 \beta^{-1} + a_2 \beta^{-2} + a_3 \beta^{-3} + ...$$

moltiplicando il numero α per la base β.

Esempio

Rappresentazione del numero $(0.1)_{10}$ in base 2.

base 2

$$\begin{aligned} 0.1 \times 2 &= 0.2 \quad \text{p. intera } 0 \\ 0.2 \times 2 &= 0.4 \quad \text{p. intera } 0 \\ 0.4 \times 2 &= 0.8 \quad \text{p. intera } 0 \\ 0.8 \times 2 &= 1.6 \quad \text{p. intera } 1 \\ 0.6 \times 2 &= 1.2 \quad \text{p. intera } 1 \\ 0.2 \times 2 &= 0.4 \quad \text{p. intera } 0 \\ &\dots \end{aligned}$$

$$(0.1)_{10} = (0.000\overline{1100})_2$$

Conversione di un reale da base 10 a base β

Siccome ogni numero si esprime come somma della sua parte intera e della sua parte frazionaria, gli algoritmi delle divisioni e delle moltiplicazioni successive possono essere impiegati per calcolare la conversione di base di un qualunque reale.

1. Determinare $|\alpha|$, ricordando il segno.
2. Determinare la parte intera $\lfloor|\alpha|\rfloor$ e eseguire la conversione con l'algoritmo delle divisioni successive.
3. Determinare la parte frazionaria $|\alpha| - \lfloor|\alpha|\rfloor$ e eseguire la conversione con l'algoritmo delle moltiplicazioni successive.
4. Scrivere il segno, la conversione della parte intera, il punto radice, la conversione della parte frazionaria.

Esempio

Convertire $\alpha = (-25.375)_{10}$ in base 2.

1. $|\alpha| = 25.375$; segno='-'.
2. $\lfloor|\alpha|\rfloor = 25$; $(25)_{10} = (11001)_2$.
3. $|\alpha| - \lfloor|\alpha|\rfloor = .375$; $(.375)_{10} = (.011)_2$.
4. $\alpha = (-11001.011)_2$.

2. I numeri di macchina

(Numeri Fixed Point)

Rappresentazione fixed point degli interi (base 2, $t+1$ bit)

- Al momento della assegnazione di un valore intero, questo valore viene trasformato dal calcolatore in una stringa di cifre binarie secondo le regole del formato fixed point (virgola fissa).
- Il formato fixed point dipende da un parametro che in dichiamo con t.
- Viene riservato in memoria uno spazio di $t+1$ bit (formati standard $t+1=16$, $t+1=32$).
- Indichiamo con $fi(N)$ la stringa di cifre binarie ottenuta applicando le regole del formato all'intero N.

$$fi(N) = \boxed{c_t}\boxed{c_{t-1}}\boxed{\cdots}\boxed{\cdots}\boxed{\cdots}\boxed{c_1}\boxed{c_0}$$

Regole della rappresentazione fixed point degli interi (base 2, $t+1$ bit)

- Se $N \geq 0$, si memorizzano le $t+1$ cifre meno significative $d_t, d_{t-1}, \ldots, d_1, d_0$ della sua rappresentazione binaria (eventuali zeri a sinistra compresi).

$$fi(N) = \boxed{d_t}\boxed{d_{t-1}}\boxed{\cdots}\boxed{\cdots}\boxed{\cdots}\boxed{d_1}\boxed{d_0}$$

- Se $N < 0$, si prendono le $t+1$ cifre meno significative $d_t, d_{t?1}, \ldots, d_1, d_0$ della rappresentazione binaria di $|N|$ (eventuali zeri a sinistra compresi) e si calcola la rappresentazione in complemento a 2 nel seguente modo:

$$\tilde{d}_i = \begin{cases} 1, & \text{se } d_i = 0 \\ 0, & \text{se } d_i = 1 \end{cases} \quad \Rightarrow \quad \begin{array}{cccc} \tilde{d}_t & \tilde{d}_{t-1} & \cdots & \tilde{d}_0 \quad + \\ & & & 1 \quad = \\ \hline c_t & c_{t-1} & \cdots & c_0 \end{array}$$

dove la somma è in aritmetica binaria.

$$fi(N) = \boxed{c_t}\boxed{c_{t-1}}\boxed{\cdots}\boxed{\cdots}\boxed{\cdots}\boxed{c_1}\boxed{c_0}$$

Esempi ($t+1=16$)

- $N = 1235 = (10011010011)_2$

$$fi(N) = \boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{1}$$

- $N = -1235 = -(10011010011)_2$
  1. Rappresentazione binaria su 16 bit: $|N| = 0000010011010011$.
  2. Scambio dei bit: 1111101100101100.
  3. Somma dell'unità: 1111101100101101.

$$fi(N) = \boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{1}$$

Esempio ($\beta = 2$, $t + 1 = 4$)

Domanda: quante sono le possibili combinazioni di bit ottenibili su 4 cifre?

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

Risposta: tutti e soli i numeri interi nell'intervallo $[-8, 7]$.

- Per un generico t l'intervallo dei numeri interi rappresentato dai numeri fixed point è $[-2^t, 2^t - 1]$.

- Per i format standard si ha

tipo	$t+1$	-2^t	$2^t - 1$
short int	16	-32768	32767
long int	32	$-2.147483648 \cdot 10^9$	$2.147483647 \cdot 10^9$

- E gli altri interi?

Esempi ($\beta = 2, t + 1 = 4$)

Cosa succede se il numero da memorizzare non appartiene a $[-2^t, 2^t - 1]$?

- $N = -9$
  - $|N| = (1001)_2$ (rappresentazione binaria di $|N|$)
  - 1001 (t cifre meno significative della rappresentazione binaria di $|N|$)
  - $\rightarrow$ 0110 (scambio dei bit)
  - $\rightarrow$ 0111 (somma dell'unità)

  $\Rightarrow fi(N) = 0111.$

- $N = 23$
  - $N = |N| = (10111)_2$

  $\Rightarrow fi(N) = 0111.$

- $N = -25$
  - $|N| = (11001)_2$ (rappresentazione binaria di $|N|$)
  - 1001 (t cifre meno significative della rappresentazione binaria di $|N|$)
  - $\rightarrow$ 0110 (scambio dei bit)
  - $\rightarrow$ 0111 (somma dell'unità)

  $\Rightarrow fi(N) = 0111.$

Esempi ($\beta = 2, t + 1 = 4$)

$\mathbb{Z}$		$fi(N)$	N
		1111	-1
		1110	-2
		1101	-3
		1100	-4
		1011	-5
		1010	-6
		1001	-7
-9	$\searrow$	1000	-8
23	$\rightarrow$	$\boxed{0111}$	7
-25	$\nearrow$	0110	6
		0101	5
		0100	4
		0011	3
		0010	2
		0001	1
		0000	0

- Per questo calcolatore vale l'uguaglianza $-9 = -25 = 23 = 7$.
- Il caso in cui la perdita di informazione si ha nella rappresentazione di un intero troppo piccolo, ossia $N < -2^t$, è chiamato underflow intero (es. -9, -25).
- Il caso in cui la perdita di informazione si ha nella rappresentazione di un intero troppo grande, ossia $N > 2^t - 1$, è chiamato overflow intero (es. -23).
- L'intervallo $[-2^t, 2^t - 1]$ è chiamato intervallo di esatta rappresentazione degli interi.

Esempi ($\beta = 2, t + 1 = 4$)

$\mathbb{Z}$		$fi(N)$	N
		1111	-1
		1110	-2
		1101	-3
		1100	-4
		1011	-5
		1010	-6
		1001	-7
-9	$\searrow$	1000	-8
23	$\rightarrow$	$\boxed{0111}$	7
-25	$\nearrow$	0110	6
		0101	5
		0100	4
		0011	3
		0010	2
		0001	1
		0000	0

- Operando con numeri compresi nell'intervallo $[-2^t, 2^t - 1]$ si può ottenere un risultato che non appartiene all'intervallo.
- Esempio: $7 + 4 = 11 = (1011)_2$
  $\Rightarrow fi(7 + 4) = fi(-5)$
  $\Rightarrow$ per questo calcolatore $7 + 4 = -5$!
- Esempio: $7 \cdot 3 = 21 = (10101)_2$
  $\Rightarrow$ si memorizzano le $t + 1$ cifre meno significative della rappresentazione binaria del risultato
  $\Rightarrow (0101)_2 \Rightarrow fi(7 \cdot 3) = fi(5)$
  $\Rightarrow$ per questo calcolatore $7 \cdot 3 = 5$.

- Il calcolatore rappresenta gli interi in formato fixed point.
- Solo un sottoinsieme limitato di interi è rappresentato esattamente al calcolatore.
- Numeri troppo grandi o troppo piccoli (overflow o underflow) vengono rappresentati con una perdita di informazione, ossia con un errore.
- I risultati delle operazioni aritmetiche tra numeri fixed point vengono rappresentati all'interno dell'intervallo $[-2^t, 2^t - 1]$ tramite le $t + 1$ cifre meno significative.
- In caso di underflow o di overflow, anche il risultato di un'operazione viene rappresentato con un errore.

3. I numeri di macchina

(Numeri Floating Point)

Fissata una base β, ogni numero reale α si può rappresentare in modo univoco nella forma

$$\alpha = \text{segno}(\alpha) m \beta^p$$

dove

- $m \in [0, 1)$ è la mantissa di α:

$$m = (.a_1 a_2 ... a_t a_{t+1} ...)_\beta = \sum_{i=1}^{\infty} a_i \beta^{-i};$$

- a_i, $i = 1, 2, \ldots$ sono le cifre della rappresentazione della mantissa nella base β;
- $a_1 \neq 0$;
- $p \in \mathbb{Z}$ è l'esponente.

$$\alpha \in \mathbb{R} \quad \Leftrightarrow \quad \alpha = \pm(0.a_1 a_2 ... a_t a_{t+1} ...)_\beta \beta^r.$$

Osserviamo che

- le cifre della rappresentazione della mantissa potrebbero essere infinite;
- l'esponente r può essere un qualsiasi numero intero.

L'insieme dei numeri floating point (a virgola mobile) si ottiene ponendo una limitazione al numero di cifre della mantissa e limiti inferiore e superiore per la scelta dell'esponente.

$$F(\beta, t, L, U) =$$
$$\left\{ \pm (0.a_1 a_2 \dots a_t)_\beta \, \beta^r : a_i \in \{0, \dots, \beta - 1\},\ i = 1, \dots, t,\ a_1 \neq 0,\ r \in \mathbb{Z},\ L \leq r \leq U \right\}$$

- E' un insieme discreto e finito di numeri reali.
- E' simmetrico rispetto all'origine.
- La sua cardinalità è data da $2(\beta - 1)\beta^{t-1}(U - L + 1)$, che si ottiene contando tutte le possibili combinazioni di mantissa ed esponente.

La normalizzazione $a_1 \neq 0$ nel caso binario implica necessariamente $a_1 = 1$. In altri termini, ogni numero reale ha un'unica rappresentazione binaria nella forma

$$\alpha = \pm(1.a_2 ... a_t a_{t+1} ...)_2 2^r.$$

In questo caso l'insieme dei numeri floating point si scrive come

$$F(2, t, L, U) = \\ \left\{ \pm (1.a_2 \dots a_t)_2 \, 2^r : a_i \in \{0, 1\},\ i = 2, \dots, t,\ r \in \mathbb{Z},\ L \leq r \leq U \right\}.$$

- Il numero positivo più grande dell'insieme (`realmax` in MATLAB) è

$$(1.1 \dots 1)_2 2^U = (2 - 2^{1-t}) 2^U.$$

- Il numero positivo più piccolo dell'insieme (`realmin` in MATLAB) è

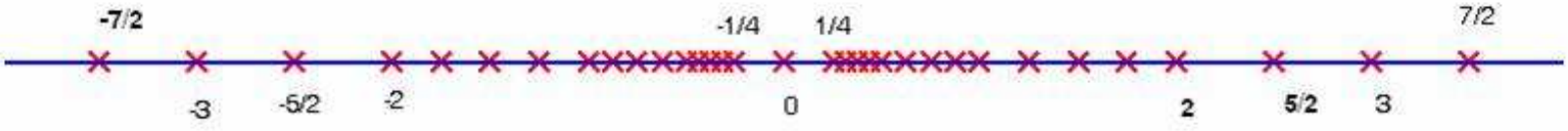
$$(1.0 \dots 0)_2 2^L = 2^L.$$

Esempio ($\beta = 2$, $t = 3$, $L = -2$, $U = 1$)

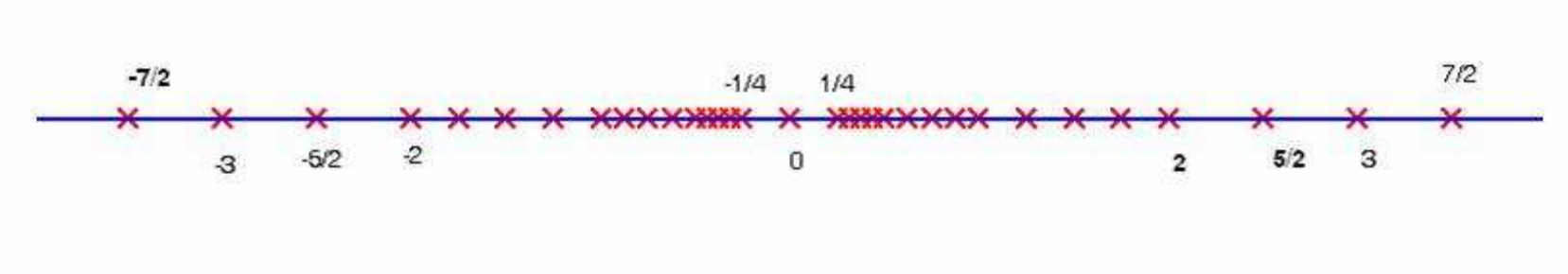
Gli elementi dell'insieme F sono:

$$\pm 1.a_2a_3 \cdot 2^p, \quad p = -2, -1, 0, 1; a_2 = 0, 1; a_3 = 0, 1$$

$m \backslash p$	-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	$\pm 7/16$	$\pm 7/8$	$\pm 7/4$	$\pm 7/2$



$$\pm(1.a_2...a_t)_2 2^r.$$

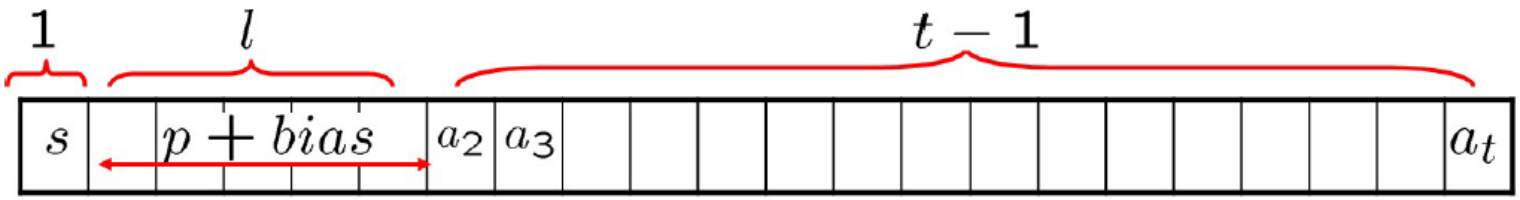


- Attorno allo 0 c'è un intervallo $[-2^L, 2^L]$ di numeri reali che vengono rappresentati come 0 (underflow floating point).
- Per numeri più grandi di $(2 - 2^{1-t})2^U$ o più piccoli di $-(2 - 2^{1-t})2^U$ si incorre in overflow floating point.
- Numeri piccoli sono meglio rappresentati.

Convenzioni del formato IEEE (Institute of Electrical and Electronics Engineering) documento 754 ANSI.

	SEMPLICE PRECISIONE	DOPPIA PRECISIONE
N. totale di bit	32 (4 byte)	64 (8 byte)
segno s	1 bit	1 bit
t-1	23	52
l	8	11
bias	127 (01111111)	1023 (01111111111)
U	127	1023
L	-126	-1022

$$\pm(1.a_2 \ldots a_t)_2 2^p$$



Esempio

Rappresentazione floating point di $\alpha = 12.3125$ in precisione semplice.

$$(12.3125)_{10} = (1100.0101)_2 = (1.1000101)_2 2^3$$
$$p = 3 \quad \Rightarrow \quad p + \text{bias} = 130$$
$$(130)_{10} = (10000010)_2.$$

0 10000010 10001010000000000000000

- Come per gli interi, definiamo un insieme di regole per approssimare un qualsiasi reale α con un elemento dell'insieme $F(\beta, t, L, U)$.
  Assumiamo che α abbia la seguente rappresentazione nella base β

$$\alpha = \pm .a_1 a_2, ..., a_t a_{t+1} ... \beta^p$$

  dove l'esponente $p \in [L, U]$.

- Definiamo il numero $fl(\alpha) \in F(\beta, t, L, U)$ ("floating di α") come segue

$$fl(\alpha) = \begin{cases} \text{troncamento} & \pm .a_1 ... a_t \ \beta^p \\ \text{arrotondamento} & \nearrow \ \pm .a_1 ... a_t \ \beta^p \quad \text{se } 0 \leq a_{t+1} < \beta/2 \\ & \searrow \ \pm .a_1 ... (a_t + 1) \ \beta^p \quad \text{se } \beta/2 \leq a_{t+1} \leq \beta - 1 \end{cases}$$

  In altre parole, un numero reale α che non appartiene all'insieme F viene rappresentato all'interno di esso per arrotondamento oppure per troncamento della sua mantissa alla $t-$esima cifra significativa.

- In generale $\alpha \neq fl(\alpha)$. Si dice che α è un numero di macchina se $\alpha = fl(\alpha)$.

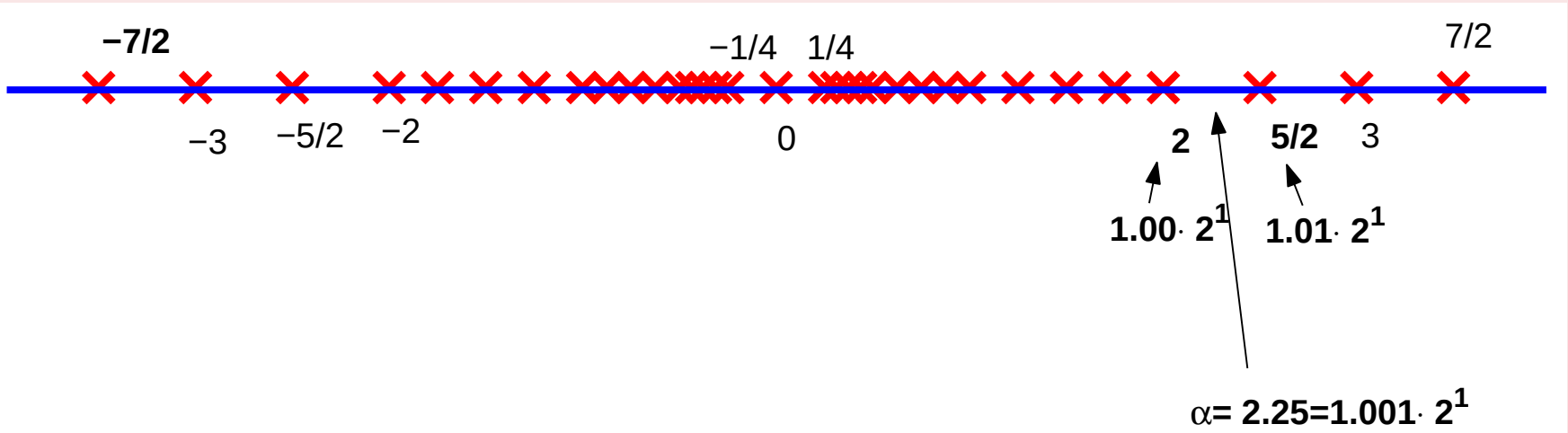
Esempio ($\beta = 2$, $t = 4$, $L = -2$, $U = 1$)

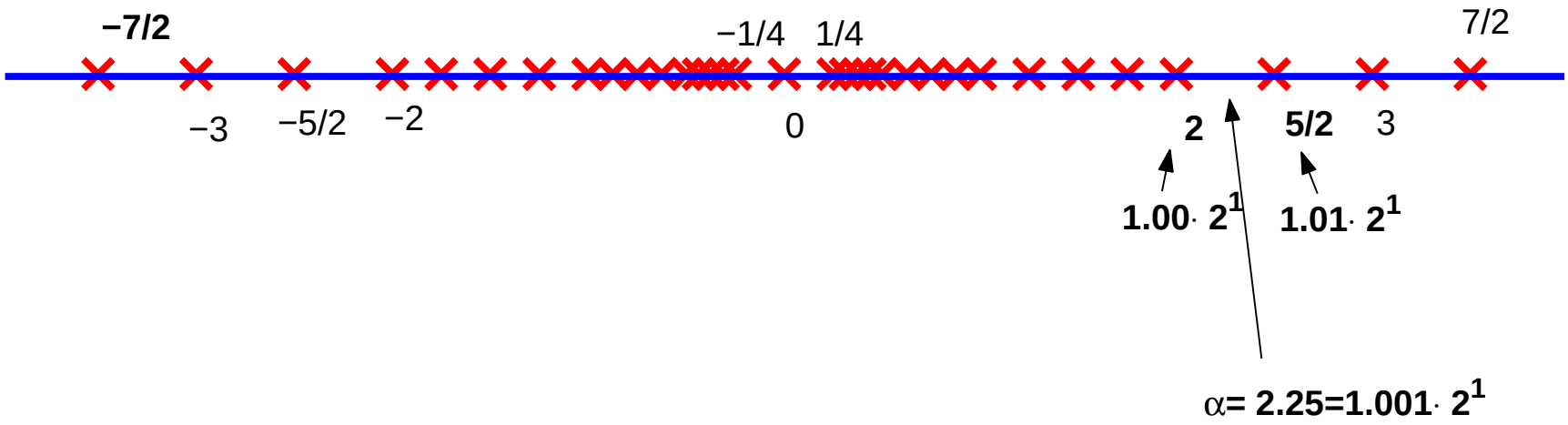
Convertire $\alpha = (2.25)_{10}$ nel corrispondente floating di un calcolatore che opera in base $\beta = 2$, con $t = 4$ cifre riservate alla mantissa e limitazioni sull'esponente date da $L = -2$ e $U = 1$.

$$(2.25)_{10} = (1.001) \cdot 2^1$$

↗ troncamento: $1.00 \cdot 2^1$

↘ arrotondamento: $1.01 \cdot 2^1$





Troncamento

Tutti i numeri dell'intervallo $[2, 5/2)$ vengono rappresentati come 2.

$$\left.\begin{array}{rcl} 2 & = & 1.00\ 2^1 \\ & & 1.001\ 2^1 \\ & & 1.0011\ 2^1 \\ & & \dots \end{array}\right\} \rightarrow 1.00\ 2^1$$

$$\frac{5}{2} = 1.01\ 2^1 \rightarrow 1.01\ 2^1$$

Arrotondamento

Tutti i numeri dell'intervallo $[15/8, 9/4)$ vengono rappresentati come 2.

$$\left.\begin{array}{rcl} 15/8 & = & 1.111\ 2^0 \\ & & 1.1111\ 2^0 \\ & & 1.111101\ 2^0 \\ & & 1.00\ 2^1 \\ & & 1.0001\ 2^1 \\ & & 1.00011\ 2^1 \\ & & \dots \end{array}\right\} \rightarrow 1.00\ 2^1$$

$$17/8 = 1.01\ 2^1 \rightarrow 1.01\ 2^1$$

Esempio

Convertire $\alpha = 0.1$ nel corrispondente floating in precisione semplice.

$$
\begin{aligned}
(0.1)_{10} &= (0.0001100110011001100110011001100...)_2 \\
&= (1.1001100110011001100110011001100...)_2 2^{-4} \\
p = -4 \quad &\Rightarrow \quad p + \text{bias} = 123 \\
(123)_{10} &= (1111011)2
\end{aligned}
$$

Troncamento:

0 01111011 10011001100110011001100

Arrotondamento:

0 01111011 10011001100110011001101

Ogni volta che un numero reale viene rappresentato in virgola mobile con precisione finita (t fissato), se questo non appartiene all'insieme $F(\beta, t, L, U)$ si ha una perdita di informazione, ossia si commette un errore. La motivazione fondante del Calcolo Numerico è la necessità di valutare problemi matematici e relativi algoritmi tenendo conto di questi errori.

Errore assoluto ed errore relativo

Sia α un numero reale. L'errore assoluto commesso approssimando α con un altro numero reale α^* è definito come

$$E_a = |\alpha - \alpha^*|.$$

Se $\alpha \neq 0$, si definisce l'errore relativo come

$$E_r = \frac{E_a}{|\alpha|}.$$

Errore assoluto ed errore relativo

$$E_a = |\alpha - \alpha^*|, \quad E_r = \frac{E_a}{|\alpha|}.$$

α	α^*	E_a	E_r
$0.3\ 10^1$	$0.31\ 10^1$	0.1	$0.3333...\ 10^{-1}$
$0.3\ 10^{-3}$	$0.31\ 10^{-3}$	$0.1\ 10^{-4}$	$0.3333...\ 10^{-1}$
$0.3\ 10^4$	$0.31\ 10^4$	$0.1\ 10^3$	$0.3333...10^{-1}$

Teorema dell'errore di rappresentazione dei numeri reali

Sia $\alpha \in \mathbb{R}, \alpha \neq 0$ e $fl(\alpha)$ la sua rappresentazione in virgola mobile in base β, con t cifre di precisione per la mantissa. L'errore relativo commesso approssimando α con $fl(\alpha)$ soddisfa la seguente disuguaglianza

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

con $k = 1$ nel caso di troncamento e $k = 1/2$ nel caso di arrotondamento.

- La quantità $u = k\beta^{1-t}$ che si trova al lato destro della disuguaglianza si chiama precisione di macchina.
- Un modo equivalente per esprimere la stessa stima è

$$fl(\alpha) = \alpha(1 + \epsilon), \quad |\epsilon| \leq k\beta^{1-t}.$$

La precisione di macchina u può essere definita anche come il più piccolo numero reale positivo tale che $fl(1+u) \neq 1$.

Esempi: $\beta = 2$, $t = 3$, troncamento

$u = 2^{-2} = (0.01)_2$

- $1 + 0.01 = 1.01 \xrightarrow{fl} 1.01$
- $1 + 0.001 = 1.001 \xrightarrow{fl} 1.00$ (perdita di informazione)

Esempi: $\beta = 2$, $t = 3$, arrotondamento

$u = \frac{1}{2}2^{-2} = (0.001)_2$

- $1 + 0.001 = 1.001 \xrightarrow{fl} 1.01$
- $1 + 0.0001 = 1.0001 \xrightarrow{fl} 1.00$ (perdita di informazione)

4. Operazioni con i numeri floating point

- Dati $x, y \in F(\beta, t, L, U)$, non è detto che il risultato di una operazione reale tra x e y sia un elemento di F. In altre parole, l'insieme F non è chiuso rispetto alle operazioni aritmetiche.

- Di conseguenza, il risultato di una operazione floating point si ottiene calcolando il risultato "esatto" dell'usuale operazione reale e rappresentando poi tale risultato entro F, troncandolo o arrotondandolo.

Nel seguito, considereremo le operazioni algebriche sui numeri Floating Point $x, y \in F(\beta, t, L, U)$ con

$$x = xm\beta^{xe}$$
$$y = ym\beta^{ye},$$

dove

- xm e xe sono la mantissa e l'esponente di x;
- ym e ye sono la mantissa e l'esponente di y.

Somma algebrica (somma/differenza aritmetica)

Indichiamo il risultato della somma come il numero floating point z con mantissa zm ed esponente ze:

$$z = x \oplus y = zm\beta^{ze}.$$

Mantissa ed esponente si ottengono eseguendo nell'ordine i seguenti passi.

1. **Denormalizzazione**
   Si portano entrambi gli addendi ad avere lo stesso esponente, quello più grande tra xe ed ye; per esempio se $xe > ye$, si divide ym per β^{xe-ye}. Di conseguenza, si fanno scorrere a destra le cifre della mantissa del numero con esponente più piccolo e contemporaneamente si incrementa l'esponente finché non si raggiunge quello più grande.

2. **Somma delle mantisse**
   Si esegue la somma delle mantisse normalizzate come al passo precedente; per esempio se $xe > ye$ si calcola $R = xm + ym/\beta^{xe-ye}$. La somma viene eseguita in un registro che permetta di memorizzare temporaneamente più delle t cifre ammesse per la rappresentazione finale.

Somma algebrica (somma/differenza aritmetica)

Indichiamo il risultato della somma come il numero floating point z con mantissa zm ed esponente ze:

$$z = x \oplus y = zm\beta^{ze}.$$

Mantissa ed esponente si ottengono eseguendo nell'ordine i seguenti passi.

3. Troncamento o arrotondamento
   Si considerano le prime t cifre più significative (con troncamento o arrotondamento) di R, ponendole in zm.

4. Normalizzazione
   Occorre eventualmente riportare il risultato finale in forma normalizzata, modificando l'esponente di conseguenza.
   Se $R \geq 1$ si pone $h = 1$, altrimenti si pone in h l'opposto del numero degli zeri ottenuti dopo il punto radice. Infine si calcola l'esponente come $ze = xe + h$.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 5$ bit a disposizione per la mantissa, mediante tecnica di arrotondamento.
Dati $x = .64932\ 10^7$ e $y = .53726\ 10^4$, calcolare $x \oplus y$.

1. L'esponente più grande è $xe = 7$; si deve quindi riscrivere y portandolo all'esponente 7. Si tratta di spostare le cifre della mantissa a destra di $xe - ye = 3$ posizioni.
$$y = .00053726\ 10^7.$$
2. Somma delle mantisse normalizzate al punto precedente:
$$R = .64932 + .00053726 = .64985726.$$
3. Le prime t cifre significative con arrotondamento di R danno la mantissa del risultato
$$zm = .64986.$$
4. Il numero è già normalizzato secondo la convenzione che abbiamo scelto, quindi l'esponente è $ze = 7$.

Il risultato della somma è il numero floating point $z = .64986\ 10^7$.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 5$ bit a disposizione per la mantissa, mediante tecnica di arrotondamento.
Dati $x = .64932\ 10^7$ e $y = .53726\ 10^7$, calcolare $x \oplus y$.

1. In questo caso gli addendi hanno già lo stesso esponente ($xe - ye = 0$) e non è necessaria nessuna operazione di denormalizzazione iniziale.
2. Si procede a sommare le mantisse

$$R = .64932 + .53726 = 1.18658$$

3. Si pongono le prime t cifre significative di R in zm

$$zm = .11866$$

4. Si deve tenere conto della normalizzazione finale, poiché $R > 1$. L'esponente è $ze = 7 + 1 = 8$.

Il risultato della somma è il numero floating point $z = .11866\ 10^8$.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 5$ bit a disposizione per la mantissa, mediante tecnica di arrotondamento.
Dati $x = .62379\ 10^7$ e $y = .32881\ 10^1$, calcolare $x \oplus y$.

1. L'esponente più grande è quello di x e $xe - ye = 6$.
2. Somma delle mantisse:

$$R = .62379 + .00000032881 = .62379032881$$

3. Arrotondamento alle prime 5 cifre significative:

$$zm = .62379$$

4. Esponente del risultato: $ze = 7$.

Il risultato della somma è il numero floating point $z = .62379\ 10^7$.

Nell'esempio precedente, si è verificato che

$$x \oplus y = x \text{ anche se } y \neq 0.$$

Questo fenomeno viene detto errore di incolonnamento e capita quando

$$|y| \leq \frac{u}{\beta}|x|,$$

dove $u = k\beta^{1-t}$ è la precisione di macchina.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 5$ bit a disposizione per la mantissa, mediante tecnica di arrotondamento.
Dati $X = .75868531\ 10^2$ e $Y = -.75868100\ 10^2$, calcolare $fl(X)$, $fl(Y)$ e $fl(X) \oplus fl(Y)$.

1. Per primo occorre rappresentare i due addendi all'interno dell'insieme F:

$$x = fl(X) = .75869\ 10^2, \quad y = fl(Y) = -75868\ 10^2.$$

2. Non è necessaria la normalizzazione iniziale ($xe - ye = 0$).
3. La somma delle mantisse fornisce

$$R = .75869 - .75868 = .00001.$$

4. Le prime t cifre significative di R danno luogo a $zm = .1$.
5. Per riportare il risultato alla normalizzazione scientifica si deve tenere conto della divisione per 10^4 del passo precedente. Si ha quindi $ze = 2 - 4 = -2$.

Il risultato della somma è il numero floating point $z = .1\ 10^{-2}$.

L'errore relativo commesso nel sostituire X e Y con $fl(X)$ e $fl(Y)$ è pari a

$$E_{r,X} = \frac{|X - fl(X)|}{|X|} = .6181\ 10^{-5} \leq \frac{1}{2}10^{-4}$$
$$E_{r,Y} = \frac{|Y - fl(Y)|}{|Y|} = .1318\ 10^{-5} \leq \frac{1}{2}10^{-4}$$

mentre l'errore relativo commesso nel sostituire il risultato esatto $X + Y$ con $fl(X) \oplus fl(Y)$ vale

$$E_{r,z} = \frac{|(X + Y) - (fl(X) \oplus fl(Y))|}{|X + Y|} \simeq 1.320186.$$

- L'errore relativo di rappresentazione dei dati è dell'ordine di 10^{-5}, mentre abbiamo un errore dell'ordine dell'unità sul risultato.
- Fenomeno di cancellazione numerica: si verifica quando si sottraggono due numeri quasi uguali in valore assoluto.
- La cancellazione delle cifre al momento della rappresentazione di X e Y come numeri floating point determina un'amplificazione dell'errore sui dati.

Prodotto

Indichiamo il risultato del prodotto come il numero floating point z con mantissa zm ed esponente ze:

$$z = x \otimes y = zm\beta^{ze}.$$

Mantissa ed esponente si ottengono eseguendo nell'ordine i seguenti passi.

1. Si esegue il prodotto tra le mantisse $xm \cdot ym$, troncando o arrotondando il risultato a t cifre, e si memorizza il risultato in zm. Se si ottiene uno zero a destra del punto radice nel prodotto $xm \cdot ym$, si pone $h = 1$, altrimenti $h = 0$.
2. L'esponente è dato da $ze = xe + ye - h$.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 5$ bit a disposizione per la mantissa, mediante tecnica di arrotondamento.
Dati $x = .11111\ 10^7$ e $y = .10202\ 10^{-2}$, calcolare $x \otimes y$.

1. Si esegue il prodotto delle mantisse

$$xm \cdot ym = .11111 \cdot .10202 = .0113354422,$$

si arrotonda alla quinta cifra e si pone in zm il risultato arrotondato e normalizzato:

$$zm = .11335.$$

2. Siccome si ottiene uno zero a destra del punto radice in $xm \cdot ym$, si pone

$$ze = 7 - 2 - 1 = 4.$$

Il risultato del prodotto è $z = .11335\ 10^4$.

Quoziente

Indichiamo il risultato del quoziente come il numero floating point z con mantissa zm ed esponente ze:

$$z = x \oslash y = zm\beta^{ze}.$$

Mantissa ed esponente si ottengono eseguendo nell'ordine i seguenti passi.

1. Se $xm < ym$ si pone $h = 0$, altrimenti si pone $h = 1$.
2. Si esegue la divisione $(xm/\beta^h)/ym$ e si pongono le t cifre più significative in zm.
3. L'esponente è dato da $ze = xe - ye + h$

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 5$ bit a disposizione per la mantissa, mediante tecnica di arrotondamento.
Dati $x = .62500\ 10^0$ e $y = .12500\ 10^{-2}$, calcolare $x \oslash y$.

1. Siccome $xm > ym$, si pone $h = 1$.
2. Si esegue la divisione

$$(xm/\beta^h)/ym = .0625/.12500 = .5$$

   e si pone

$$zm = .5.$$

3. L'esponente è dato da

$$ze = 0 - (-2) + 1 = 3.$$

Il risultato della divisione è $z = .5\ 10^3$.

- Dati $x, y \in F(\beta, t, L, U)$, non è detto che il risultato di una operazione reale tra x e y sia un elemento di F. In altre parole, l'insieme F non è chiuso rispetto alle operazioni aritmetiche.

- Di conseguenza, il risultato di una operazione floating point si ottiene calcolando il risultato "esatto" dell'usuale operazione reale e rappresentando poi tale risultato entro F, troncandolo o arrotondandolo.

- Conseguenza della rappresentazione entro F è l'introduzione di un potenziale errore ogni qual volta il risultato dell'operazione non appartiene ad F e deve quindi essere arrotondato o troncato alla t-esima cifra della mantissa.

Teorema dell'errore di rappresentazione del risultato delle operazioni

Sia $\bullet$ una qualsiasi delle quattro operazioni aritmetiche.
Dati $x, y \in F(\beta, t, L, U)$, l'errore relativo commesso approssimando il risultato esatto dell'operazione $x \bullet y$ con la sua rappresentazione floating point soddisfa la disuguaglianza

$$\frac{|fl(x \bullet y) - (x \bullet y)|}{|x \bullet y|} \leq k\beta^{1-t},$$

dove $k = \frac{1}{2}$ oppure $k = 1$ a seconda che la rappresentazione sia ottenuta per arrotondamento o per troncamento rispettivamente.

- Si tratta del teorema dell'errore di rappresentazione dei numeri reali applicato nel caso particolare $\alpha = x \bullet y$.
- Si può scrivere in forma equivalente come

$$fl(x \bullet y) = (x \bullet y)(1 + \epsilon), \quad |\epsilon| \leq k\beta^{1-t}.$$

Non validità di alcune proprietà delle operazioni aritmetiche

- A causa della memoria finita del calcolatore e delle regole dell'aritmetica di macchina, le operazioni di macchina non godono di tutte le proprietà fondamentali che valgono per le operazioni usuali.

- Seguono alcuni esempi che mostrano la non validità di alcune di queste proprietà.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 5$ bit a disposizione per la mantissa, mediante tecnica di arrotondamento.
Dati $x = .62379\ 10^7$ e $y = .32881\ 10^1$, calcolare $x \oplus y$.

1. L'esponente più grande è quello di x e $xe - ye = 6$.
2. Somma delle mantisse:

$$R = .62379 + .00000032881 = .62379032881$$

3. Arrotondamento alle prime 5 cifre significative:

$$zm = .62379$$

4. Esponente del risultato: $ze = 7$.

Il risultato della somma è il numero floating point $z = .62379\ 10^7$.

Nell'esempio precedente, a causa di un errore di incolonnamento, si verifica che

$$x \oplus y = x \text{ anche se } y \neq 0.$$

Di conseguenza, per la somma floating point l'elemento neutro non è unico.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 4$ bit a disposizione per la mantissa, mediante tecnica di troncamento.
Dati $x = .2000\ 10^4$, $y = .2500\ 10^1$ e $z = 0.7800\ 10$, calcolare $(x \oplus y) \oplus z$ e $x \oplus (y \oplus z)$.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 4$ bit a disposizione per la mantissa, mediante tecnica di troncamento.
Dati $x = .2000\ 10^4$, $y = .2500\ 10^1$ e $z = 0.7800\ 10$, calcolare $(x \oplus y) \oplus z$ e $x \oplus (y \oplus z)$.

1. Primo algoritmo: $(x \oplus y) \oplus z$

$$\begin{aligned}(x \oplus y) \oplus z &= (0.2000\ 10^4 \oplus .2500\ 10^1) \oplus (0.7800\ 10^1) \\ &= (0.2000\ 10^4 \oplus 0.00025\ 10^4) \oplus (0.7800\ 10^1) \\ &= (0.2002\ 10^4) \oplus (0.7800\ 10^1) \\ &= (0.2002\ 10^4 \oplus 0.00078\ 10^4) \\ &= (0.2009\ 10^4).\end{aligned}$$

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 4$ bit a disposizione per la mantissa, mediante tecnica di troncamento.
Dati $x = .2000\ 10^4$, $y = .2500\ 10^1$ e $z = 0.7800\ 10$, calcolare $(x \oplus y) \oplus z$ e $x \oplus (y \oplus z)$.

1. Secondo algoritmo: $x \oplus (y \oplus z)$

$$\begin{aligned} x \oplus (y \oplus z) &= 0.2000\ 10^4 \oplus (.2500\ 10^1 \oplus 0.7800\ 10^1) \\ &= 0.2000\ 10^4 \oplus (0.1030\ 10^2) \\ &= (0.2000\ 10^4 \oplus 0.001030\ 10^4) \\ &= (0.2010\ 10^4). \end{aligned}$$

2) La somma non gode della proprietà associativa

- In aritmetica esatta:

$$(x + y) + z = x + (y + z) = 2010.3.$$

- In aritmetica di macchina:

$$(x \oplus y) \oplus z \neq x \oplus (y \oplus z).$$

2) La somma non gode della proprietà associativa

- L'esempio precedente può essere visto come un confronto tra due algoritmi diversi per la somma di tre numeri.

ALGORITMO 1	ALGORITMO 2
$s \leftarrow x + y$	$s \leftarrow y + z$
$s \leftarrow s + z$	$s \leftarrow s + x$

- I due algoritmi hanno fornito risultati diversi. Il risultato fornito dal secondo algoritmo è più accurato del primo, infatti calcolando gli errori relativi

$$E_a^1 = 2010.3 - 2009 = 1.3 \quad \Rightarrow \quad E_r^1 = \frac{E_a^1}{2010.3} \simeq 6.47 \cdot 10^{-4}$$

$$E_a^2 = 2010.3 - 2010 = 0.3 \quad \Rightarrow \quad E_r^2 = \frac{E_a^2}{2010.3} \simeq 1.49 \cdot 10^{-4}$$

si osserva che $E_a^2 < E_a^1$.

- Siccome la precisione di macchina è data da $\beta^{1-t} = 10^{1-4} = 10^{-3}$, si osserva che entrambi i risultati sono accettabili nei limiti della precisione di macchina, in quanto $E_r^1 < 10^{-3}$ e $E_r^2 < 10^{-3}$.

2) La somma non gode della proprietà associativa

In generale:

- il risultato del calcolo di un'espressione, intesa come successione di operazioni aritmetiche, dipende dall'algoritmo usato per calcolarla;
- siccome i risultati di operazioni di macchina contengono errori, l'errore nel risultato finale dipende dall'algoritmo utilizzato.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 2$ bit a disposizione per la mantissa, mediante tecnica di troncamento.
Dati $x = .91\ 10^1$, $y = .92\ 10^1$ e $z = .10\ 10^0$, calcolare $x \otimes (y \oplus z)$ e $(x \otimes y) \oplus (x \otimes z)$.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 2$ bit a disposizione per la mantissa, mediante tecnica di troncamento.
Dati $x = .91\ 10^1$, $y = .92\ 10^1$ e $z = .10\ 10^0$, calcolare $x \otimes (y \oplus z)$ e $(x \otimes y) \oplus (x \otimes z)$.

1. Primo algoritmo: $x \otimes (y \oplus z)$

$$\begin{aligned} x \otimes (y \oplus z) &= (.91\ 10^1) \otimes (.92\ 10^1 \oplus .10\ 10^0) \\ &= (.91\ 10^1) \otimes (.92\ 10^1 \oplus .010\ 10^1) \\ &= (.91\ 10^1) \otimes .93\ 10^1 \\ &= (.8463\ 10^2) \\ &= .84\ 10^2. \end{aligned}$$

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 2$ bit a disposizione per la mantissa, mediante tecnica di troncamento.
Dati $x = .91\ 10^1$, $y = .92\ 10^1$ e $z = .10\ 10^0$, calcolare $x \otimes (y \oplus z)$ e $(x \otimes y) \oplus (x \otimes z)$.

1. Secondo algoritmo: $(x \otimes y) \oplus (x \otimes z)$

$$\begin{aligned}(x \otimes y) \oplus (x \otimes z) &= (.91\ 10^1 \otimes .92\ 10^1) \oplus (.91\ 10^1 \otimes .10\ 10^0) \\ &= (.83\ 10^2) \oplus (.91\ 10^0) \\ &= (.83\ 10^2 \oplus 0.0091\ 10^2) \\ &= (.83\ 10^2)\end{aligned}$$

3) La somma non gode della proprietà distributiva rispetto al prodotto

Si è verificato che

$$x \otimes (y \oplus z) \neq (x \otimes y) \oplus (x \otimes z).$$

Di conseguenza, la somma non gode della proprietà distributiva rispetto al prodotto.

Esempio

Supponiamo che un calcolatore operi in base $\beta = 10$, con $t = 7$ bit a disposizione per la mantissa, $L = -50$ e $U = 49$.
Dati $x = .2\ 10^{-27}$ e $y = .1\ 10^{-26}$, calcolare $x \otimes y$.

Si ha che

$$x \otimes y = .2\ 10^{-27} \otimes .1\ 10^{-26} = .2\ 10^{-54} = 0.$$

Siccome l'esponente è < -50, si verifica un UNDERFLOW!

- Si è verificato che non vale la legge dell'annullamento del prodotto: se $x \otimes y = 0$, ciò NON implica che uno dei due fattori sia nullo.

Conclusioni

- Il calcolatore ha a disposizione un insieme limitato e finito di numeri reali.
  $\Rightarrow$ Ogni volta che si assegna un valore reale ad una variabile in semplice o doppia precisione si commette un potenziale errore.

- Il calcolatore utilizza un'aritmetica diversa dalla nostra
  $\Rightarrow$ Ogni volta che si esegue una operazione aritmetica al calcolatore si commette un potenziale errore.

- Entrambi questi errori sono maggiorati dalla precisione di macchina.

- A causa di tali errori, l'aritmetica di macchina non gode di molte delle proprietà delle operazioni usuali
  $\Rightarrow$ Algoritmi diversi possono fornire risultati con errori diversi
  $\Rightarrow$ Vogliamo poter valutare gli algoritmi per poter selezionare quelli che danno gli errori più piccoli.

5. Analisi degli errori

Osservazioni dalle scorse lezioni

1. Gli errori sui dati possono determinare la perdita di cifre significative nel risultato di un algoritmo (cancellazione numerica).

2. Gli errori di arrotondamento/troncamento nelle operazioni fanno sì che formule matematicamente equivalenti non lo siano più quando si opera in aritmetica finita. Di conseguenza, esistono algoritmi che producono risultati "più accurati" di altri, nel senso che rendono più piccolo l'errore relativo sul risultato.

- Gli errori di arrotondamento nelle operazioni e/o gli errori sui dati possono "accumularsi" nella successione delle operazioni, determinando un'amplificazione dell'errore sul risultato finale dell'algoritmo.

- L'accumulo degli errori viene anche detto propagazione degli errori.

- Nell'Analisi Numerica, lo studio della propagazione degli errori viene condotto sotto due punti di vista differenti:

↙ ↘

ANALISI DEL CONDIZIONAMENTO DI UN PROBLEMA
Come varia la soluzione di un problema in presenza di errori sui dati?

ANALISI DELLA STABILITA' DI UN ALGORITMO
Come si propagano gli errori attraverso le operazioni di un algoritmo?

Definizione

Un algoritmo è una successione finita di istruzioni, assegnate in modo non ambiguo, la cui esecuzione consente di passare dai dati in ingresso ai risultati in uscita in un tempo finito.

$$\text{PROBLEMA} \quad \rightarrow \quad \text{ALGORITMO} \quad \rightarrow \quad \text{SOLUZIONE}$$

Definizione

Un algoritmo si dice stabile se non è troppo sensibile agli errori di rappresentazione introdotti dalle operazioni in aritmetica floating point.
Se x è la soluzione del problema e x^* la soluzione fornita dall'algoritmo, la stabilità dell'algoritmo si valuta calcolando l'errore relativo algoritmico

$$E_{alg} = \frac{x^* - x}{x}.$$

- La stabilità dipende dal numero, dal tipo e dall'ordine delle operazioni eseguite dall'algoritmo.
- Un algoritmo A è più stabile dell'algoritmo B se l'errore algoritmico di A è più piccolo dell'errore algoritmico di B.

Nel seguito, verrà considerata una tecnica denominata analisi in avanti del primo ordine per la stima dell'errore relativo algoritmico, nella quale

- si assume che i dati siano esatti, ossia appartenenti ad $F(\beta, t, L, U)$;
- si utilizza il teorema fondamentale sulle operazioni fra numeri di macchina: se $a, b \in F(\beta, t, L, U)$, si ha

$$fl(a \bullet b) = (a \bullet b)(1 + \epsilon), \quad |\epsilon| \leq u,$$

  dove u indica la precisione di macchina;
- si stima l'errore relativo algoritmico, trascurando i termini in cui compaiono prodotti di errori.

Esempio: somma di tre numeri di macchina

- Il problema consiste nel calcolo della funzione

$$\varphi(a, b, c) = a + b + c.$$

- Consideriamo i seguenti due algoritmi numerici

ALGORITMO 1	ALGORITMO 2
$\varphi_1(a, b, c) = (a \oplus b) \oplus c$	$\varphi_2(a, b, c) = a \oplus (b \oplus c),$

  dove $\oplus$ indica l'operazione di somma floating point.

- Per ciascuno dei due algoritmi, ricaviamo l'errore relativo algoritmico applicando il teorema fondamentale sulle operazioni di macchina.

Esempio: somma di tre numeri di macchina

ALGORITMO 1: $(a \oplus b) \oplus c$

$$
\begin{aligned}
y = a \oplus b = fl(a+b) &= (a+b)(1+\epsilon_1), \quad |\epsilon_1| \leq u \\
z = y \oplus c = fl(y+c) &= (y+c)(1+\epsilon_2), \quad |\epsilon_2| \leq u \\
&= ((a+b)(1+\epsilon_1)+c)(1+\epsilon_2) \\
&= (a+b)(1+\epsilon_1)(1+\epsilon_2)+c(1+\epsilon_2) \\
&\simeq (a+b)(1+\epsilon_1+\epsilon_2)+c(1+\epsilon_2) \\
&= a+b+c+(a+b)\epsilon_1+(a+b+c)\epsilon_2,
\end{aligned}
$$

dove si è trascurato il termine $(a+b)\epsilon_1\epsilon_2$.

L'errore relativo algoritmico si approssima quindi come

$$
E_{alg1} = \frac{((a \oplus b) \oplus c) - (a+b+c)}{a+b+c} \simeq \frac{a+b}{a+b+c}\epsilon_1 + \epsilon_2.
$$

Esempio: somma di tre numeri di macchina

ALGORITMO 1: $(a \oplus b) \oplus c$

$$E_{alg1} = \frac{((a \oplus b) \oplus c) - (a+b+c)}{a+b+c} \simeq \frac{a+b}{a+b+c}\epsilon_1 + \epsilon_2.$$

In E_{alg1} si distinguono due fattori di amplificazione dell'errore, ossia due coefficienti che moltiplicano gli errori dovuti ad ogni singola operazione:

- $(a+b)/(a+b+c)$ per ϵ_1
- 1 per ϵ_2.

Si definisce indice algoritmico la somma dei valori assoluti dei fattori di amplificazione dei singoli errori introdotti da ciascuna operazione.
Nel caso in questione si ha

$$I_{alg1} = \frac{|a+b|}{|a+b+c|} + 1.$$

Esempio: somma di tre numeri di macchina

ALGORITMO 2: $a \oplus (b \oplus c)$

$$
\begin{aligned}
v = b \oplus c &= fl(b+c) = (b+c)(1+\epsilon_3), \quad |\epsilon_3| \leq u \\
w = a \oplus v &= fl(a+v) = (a+v)(1+\epsilon_4), \quad |\epsilon_4| \leq u \\
&= (a+(b+c)(1+\epsilon_3))(1+\epsilon_4) \\
&= (b+c)(1+\epsilon_3)(1+\epsilon_4) + a(1+\epsilon_4) \\
&\simeq (b+c)(1+\epsilon_3+\epsilon_4) + a(1+\epsilon_4) \\
&= a+b+c+(b+c)\epsilon_3 + (a+b+c)\epsilon_4,
\end{aligned}
$$

dove si è trascurato il termine $(b+c)\epsilon_3\epsilon_4$.

L'errore relativo algoritmico si approssima quindi come

$$
E_{alg2} = \frac{(a \oplus (b \oplus c)) - (a+b+c)}{a+b+c} \simeq \frac{b+c}{a+b+c}\epsilon_3 + \epsilon_4.
$$

Esempio: somma di tre numeri di macchina

ALGORITMO 1: $(a \oplus b) \oplus c$

$$E_{alg2} = \frac{(a \oplus (b \oplus c)) - (a+b+c)}{a+b+c} \simeq \frac{b+c}{a+b+c}\epsilon_3 + \epsilon_4.$$

In E_{alg2} i fattori di amplificazione dell'errore sono

- $(b+c)/(a+b+c)$ per ϵ_3
- 1 per ϵ_4.

L'indice algoritmico è dato da

$$I_{alg2} = \frac{|b+c|}{|a+b+c|} + 1.$$

Esempio: somma di tre numeri di macchina

Quale dei due algoritmi è più stabile? In altre parole:

$$I_{alg1} \gtrless I_{alg2}?$$

La risposta dipende dai dati a, b, c che vengono sommati.
Infatti entrambi gli indici dipendono dai valori assegnati ai dati a, b, c.

Esempio: somma di tre numeri di macchina

Nell'esempio precedente, in cui il calcolatore opera in base $\beta = 10$ con $t = 4$ bit a disposizione per la mantissa approssimando mediante tecnica di troncamento e i dati sono

$$a = .2000\ 10^4, \quad b = .2500\ 10^1, \quad c = .7800\ 10^1,$$

risulta che

$$I_{alg1} = \frac{.2000\ 10^4 + .2500\ 10^1}{.2000\ 10^4 + .2500\ 10^1 + .7800\ 10^1} + 1 = \frac{.20025\ 10^4}{.20103\ 10^4} + 1 \simeq 2$$

$$I_{alg2} = \frac{.2500\ 10^1 + .7800\ 10^1}{.2000\ 10^4 + .2500\ 10^1 + .7800\ 10^1} + 1 = \frac{.1030\ 10^2}{.20103\ 10^4} + 1 \simeq 1.01$$

Si ha $I_{alg2} < I_{alg1}$ e dunque il secondo algoritmo è più stabile del primo per i valori assunti dai dati.

Esempio: somma di n numeri di macchina

- Dati $x_1, x_2, \ldots, x_n \in F(\beta, t, L, U)$, il problema consiste nel calcolo della funzione

$$\varphi(x_1, x_2, \ldots, x_n) = x_1 + x_2 + \ldots + x_n = s.$$

- Consideriamo il seguente algoritmo numerico:

$$\begin{array}{l} s \leftarrow x_1 \\ \text{FOR } i = 2, \ldots, n \\ \quad \lfloor \; s \leftarrow s \oplus x_i \end{array}$$

  Sono $n - 1$ somme.

Esempio: somma di n numeri di macchina

Generalizzando i passaggi eseguiti per il caso della somma di tre numeri, se indichiamo con ϵ_i l'errore dovuto alla somma eseguita all' $i-$esimo passo dell'algoritmo troviamo la seguente espressione per l'errore relativo algoritmico:

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

L'indice algoritmico è dato da

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

Esempio: somma di n numeri di macchina

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

- Se gli x_i sono di segno discorde, può essere $|x_1 + \ldots + x_n| << 1$ e dunque l'indice algoritmico diventa grande, per cui si ha amplificazione degli errori $\Rightarrow$ l'algoritmo diventa instabile.
- Se gli x_i sono di segno concorde, dato che $|\epsilon_i| \leq u$, $i = 2, \ldots, n$, segue che $|E_{alg}| \leq (n-1)u$. Tuttavia l'algoritmo è più stabile numericamente se si sommano i numeri dal più piccolo al più grande.

Esempio: somma di n numeri di macchina

Supponiamo che $\beta = 10$, $t = 7$, tecnica di arrotondamento e

$$x_1 = .1\ 10^1,$$
$$x_i = .1\ 10^{-6},\ i = 2, \ldots, 10.$$

La somma esatta è data da

$$x_1 + \ldots + x_n = 1 + 9 \cdot 10^{-6} = 1.0000009.$$

Consideriamo i due algoritmi numerici:

ALGORITMO 1	ALGORITMO 2
$s \leftarrow x_1$	$s \leftarrow x_n$
FOR $i = 2 : n$	FOR $i = n - 1 : -1 : 1$
$\quad s \leftarrow s \oplus x_i$	$\quad s \leftarrow s \oplus x_i$

Esempio: somma di n numeri di macchina

ALGORITMO 1

- Sommando i numeri a partire dal più grande si ottiene come risultato finale il valore $s = 1$, poichè ad ogni passo dell'algoritmo si verifica un errore di incolonnamento:

$$
\begin{aligned}
i = 2 \quad & s \leftarrow s \oplus x_2 = fl(.1\ 10^1 + .1\ 10^{-6}) = fl(.10000001\ 10^1) = 1 \\
i = 3 \quad & s \leftarrow s \oplus x_3 = fl(.1\ 10^1 + .1\ 10^{-6}) = fl(.10000001\ 10^1) = 1 \\
& \vdots \\
i = 10 \quad & s \leftarrow s \oplus x_{10} = fl(.1\ 10^1 + .1\ 10^{-6}) = fl(.10000001\ 10^1) = 1
\end{aligned}
$$

$$
E_{alg} = \frac{1.0000009 - 1}{1.0000009} = 8.999\ldots \cdot 10^{-7} \simeq 10^{-6}.
$$

Esempio: somma di n numeri di macchina

ALGORITMO 2

- Se invece si sommano prima i termini più piccoli, si ottiene il risultato finale $s = 1.000001$, poiché

$$
\begin{aligned}
i = 2 \quad & s \leftarrow s \oplus x_9 = fl(.1\ 10^{-6} + .1\ 10^{-6}) = .2\ 10^{-6} \\
i = 3 \quad & s \leftarrow s \oplus x_8 = fl(.2\ 10^{-6} + .1\ 10^{-6}) = .3\ 10^{-6} \\
& \vdots \\
i = 10 \quad & s \leftarrow s \oplus x_1 = fl(.9\ 10^{-6} + 1) = fl(.10000009\ 10^{1}) = 1.000001.
\end{aligned}
$$

$$
E_{alg} = \frac{1.000001 - 1.0000009}{1.0000009} = 9.999\ldots \cdot 10^{-8} \simeq 10^{-7}.
$$

L'errore relativo algoritmico risulta 10 volte più piccolo rispetto al primo algoritmo.

Esempio: prodotto di n numeri di macchina

- Dati $x_1, x_2, \ldots, x_n \in F(\beta, t, L, U)$, il problema consiste nel calcolo della funzione
$$\varphi(x_1, x_2, \ldots, x_n) = x_1 \cdot x_2 \cdot \ldots \cdot x_n.$$

- Consideriamo il seguente algoritmo numerico:

$$\begin{array}{l} p \leftarrow x_1 \\ \text{FOR } i = 2, \ldots, n \\ \quad \lfloor \; p \leftarrow p \otimes x_i \end{array}$$

  Sono $n - 1$ prodotti.

Esempio: prodotto di n numeri di macchina

L'analisi in avanti del primo ordine è la seguente:

$$\begin{aligned}
i = 2 \quad & p = p \otimes x_2 = fl(x_1 \cdot x_2) = x_1 x_2 (1 + \epsilon_2) \\
i = 3 \quad & p = p \otimes x_3 = fl(p \cdot x_3) = x_1 x_2 x_3 (1 + \epsilon_2)(1 + \epsilon_3) \\
& \vdots \\
i = n \quad & p = p \otimes x_n = fl(p \cdot x_n) = x_1 x_2 x_3 \ldots x_n (1 + \epsilon_2)(1 + \epsilon_3) \ldots (1 + \epsilon_n)
\end{aligned}$$

dove gli ϵ_i, $|\epsilon_i| \leq u$, $i = 2, \ldots, n$, sono gli errori dovuti alle singole operazioni. Trascurando i termini del secondo ordine, otteniamo che

$$\begin{aligned}
p = fl(x_1 \cdot x_2 \cdot \ldots \cdot x_n) &\simeq x_1 x_2 x_3 \ldots x_n (1 + \epsilon_2 + \epsilon_3 + \ldots + \epsilon_n) \\
&= x_1 x_2 x_3 \ldots x_n + x_1 x_2 x_3 \ldots x_n \epsilon_2 \\
&\quad + \ldots + x_1 x_2 x_3 \ldots x_n \epsilon_n.
\end{aligned}$$

L'errore relativo algoritmico si approssima quindi come

$$E_{alg} = \frac{fl(x_1 \cdot x_2 \cdot \ldots \cdot x_n) - x_1 \cdot x_2 \cdot \ldots x_n}{x_1 \cdot x_2 \cdot \ldots x_n} = \epsilon_2 + \ldots + \epsilon_n.$$

Esempio: prodotto di n numeri di macchina

$$E_{alg} = \epsilon_2 + \ldots + \epsilon_n.$$

Siccome $|\epsilon_i| \leq u$, $i = 2 \ldots, n$, segue che $|E_{alg}| \leq (n-1)u$ e $I_{alg} = n - 1$
$\Rightarrow$ l'algoritmo è stabile.

Più in generale, i criteri con cui si valutano gli algoritmi sono due:

1. La complessità computazionale: il numero di operazioni aritmetiche necessarie per portare a termine un algoritmo, espresso in funzione della dimensione del problema.
   - Fornisce una valutazione dell'efficienza dell'algoritmo indipendente dal processore.
   - Il tempo di calcolo è direttamente proporzionale alla complessità computazionale.

2. La stabilità: la capacità dell'algoritmo di non amplificare troppo gli errori dovuti alle operazioni di macchina.

Esempio: somma di n numeri, confronto fra Algoritmo 1 e Algoritmo 2

- La complessità computazionale è pari a $n-1$ somme per entrambi gli algoritmi.
- L'algoritmo 2 è più stabile dell'algoritmo 1.

Definizione

Un problema si dice ben condizionato se a piccole perturbazioni dei dati corrispondono altrettanto piccole variazioni delle soluzioni. Viceversa, un problema si dice mal condizionato se a piccole perturbazioni dei dati corripondono (relativamente) grandi variazioni delle soluzioni.

Se α è la soluzione del problema con dati esatti e α^* la soluzione del problema con dati perturbati, il condizionamento del problema si valuta calcolando l'errore relativo inerente

$$E_{ine} = \frac{\alpha^* - \alpha}{\alpha}.$$

- Il condizionamento è una proprietà della funzione che lega i dati alla soluzione di un problema, non dipende da come queste vengano calcolate.
- Vedremo una definizione più rigorosa di condizionamento per i sistemi lineari.

Esempio: calcolo delle radici di un polinomio di 2° grado

Supponiamo di voler calcolare le soluzioni dell'equazione

$$x^2 - 4x + c = 0.$$

Dati: $a_2 = 1$, $a_1 = 4$, $a_0 = c$.

Soluzioni:

$$x_{1,2} = \frac{-\frac{a_1}{2} \pm \sqrt{\left(\frac{a_1}{2}\right)^2 - a_2 a_0}}{a_2} = 2 \pm \sqrt{4 - c}.$$

Esempio: calcolo delle radici di un polinomio di 2° grado

Supponiamo di voler calcolare le soluzioni dell'equazione

$$x^2 - 4x + c = 0.$$

Dati: $a_2 = 1$, $a_1 = 4$, $a_0 = c$.

Soluzioni:

$$x_{1,2} = \frac{-\frac{a_1}{2} \pm \sqrt{\left(\frac{a_1}{2}\right)^2 - a_2 a_0}}{a_2} = 2 \pm \sqrt{4 - c}.$$

Per semplicità studiamo come variano le soluzioni al variare del solo termine noto c e prendiamo la soluzione con il segno negativo.

$$\begin{aligned} c = 4 \quad &\rightarrow \quad \alpha = 2 \\ c^* = 4 - 10^{-6} \quad &\rightarrow \quad \alpha^* = 2 - 10^{-3}. \end{aligned}$$

Variazione sul dato	Variazione sulla soluzione
$\frac{c^* - c}{c} = \frac{1}{4} 10^{-6}.$	$\underbrace{E_{ine} = \frac{\alpha^* - \alpha}{\alpha} = -\frac{1}{2} 10^{-3}.}_{= \text{errore relativo inerente}}$

Esempio: calcolo delle radici di un polinomio di $2°$ grado

Supponiamo di voler calcolare le soluzioni dell'equazione

$$x^2 - 4x + c = 0.$$

Dati: $a_2 = 1$, $a_1 = 4$, $a_0 = c$.

Soluzioni:

$$x_{1,2} = \frac{-\frac{a_1}{2} \pm \sqrt{\left(\frac{a_1}{2}\right)^2 - a_2 a_0}}{a_2} = 2 \pm \sqrt{4 - c}.$$

Il problema è mal condizionato per valori del dato c vicini a 4

La variazione sulla soluzione è di 3 ordini di grandezza superiore alla perturbazione sul dato: 10^{-3} vs. 10^{-6}.

Esempio: somma di due numeri

- Dati: $a, b \in \mathbb{R}$.
- Soluzione: $\varphi(a, b) = a + b$.
- Supponiamo che i dati vengano approssimati in macchina dai floating

$$fl(a) = a(1 + \epsilon_a), \quad fl(b) = b(1 + \epsilon_b),$$

  dove $|\epsilon_a| \leq u$ e $|\epsilon_b| \leq u$.
- Supponiamo che la somma venga eseguita esattamente (no errori di arrotondamento/troncamento) sui dati perturbati.
  Come cambia il risultato della somma in presenza di dati perturbati?

$$fl(a) + fl(b) = a(1 + \epsilon_a) + b(1 + \epsilon_b) = a + b + (a\epsilon_a + b\epsilon_b)$$

$$E_{ine} = \frac{(fl(a) + fl(b)) - (a + b)}{a + b} = \frac{a}{a + b}\epsilon_a + \frac{b}{a + b}\epsilon_b.$$

- N.B. Se a e b hanno segno opposto e $|a| \simeq |b|$, si può verificare la cancellazione numerica con amplificazione dell'errore sulla soluzione
  $\Rightarrow$ Mal condizionamento per a e b con segno opposto e $|a| \simeq |b|$

Esempio: prodotto di due numeri

- Dati: $a, b \in \mathbb{R}$.
- Soluzione: $\varphi(a, b) = a \cdot b$.
- Supponiamo che i dati vengano approssimati in macchina dai floating

$$fl(a) = a(1 + \epsilon_a), \quad fl(b) = b(1 + \epsilon_b),$$

  dove $|\epsilon_a| \leq u$ e $|\epsilon_b| \leq u$.
- Supponiamo che il prodotto venga eseguito esattamente (no errori di arrotondamento/troncamento) sui dati perturbati.
  Come cambia il risultato del prodotto in presenza di dati perturbati?

$$\begin{aligned} fl(a) \cdot fl(b) &= a(1 + \epsilon_a) \cdot b(1 + \epsilon_b) = (a + a\epsilon_a) \cdot (b + b\epsilon_b) \\ &= ab + ab\epsilon_b + ab\epsilon_a + ab\epsilon_a\epsilon_b \simeq ab(1 + \epsilon_a + \epsilon_b). \end{aligned}$$

$$E_{ine} = \frac{(fl(a) \cdot fl(b)) - (a \cdot b)}{a \cdot b} \simeq \epsilon_a + \epsilon_b.$$

- Il prodotto è un'operazione ben condizionata per qualunque valore dei dati.

- Supponiamo di voler valutare una formula matematica $\varphi(x)$ mediante un algoritmo numerico con dati perturbati. Andranno considerati sia gli errori sui dati che gli errori sulle operazioni.

- Sia $\tilde{x}$ il dato perturbato. Allora l'errore relativo totale è dato da

$$
\begin{aligned}
E_{tot} &= \frac{fl(\varphi(\tilde{x})) - \varphi(x)}{\varphi(x)} \\
&= \frac{fl(\varphi(\tilde{x})) - \varphi(\tilde{x}) + \varphi(\tilde{x}) - \varphi(x)}{\varphi(x)} \\
&= \frac{\varphi(\tilde{x}) - \varphi(x)}{\varphi(x)} + \frac{fl(\varphi(\tilde{x})) - \varphi(\tilde{x})}{\varphi(\tilde{x})} \left(\frac{\varphi(\tilde{x}) - \varphi(x) + \varphi(x)}{\varphi(x)} \right) \\
&= E_{ine} + E_{alg}(1 + E_{ine}) \simeq E_{ine} + E_{dati}.
\end{aligned}
$$

Nell'esempio della somma dei tre numeri $\alpha = x + y + z$, si ha che

$$\begin{aligned}\alpha^* &= (x \oplus y) \oplus z \\ &= ((x(1+\epsilon_x) + y(1+\epsilon_y))(1+\epsilon_1) + z(1+\epsilon_z))(1+\epsilon_2) \\ &\simeq x + y + z + x\epsilon_x + y\epsilon_y + z\epsilon_z + (x+y)\epsilon_1 + (x+y+z)\epsilon_2,\end{aligned}$$

dunque

$$\begin{aligned}E_{tot} &= \frac{\alpha^* - \alpha}{\alpha} \\ &\simeq \underbrace{\frac{x}{x+y+z}\epsilon_x + \frac{y}{x+y+z}\epsilon_y + \frac{z}{x+y+z}\epsilon_z}_{\text{errore inerente}} + \underbrace{\frac{x+y}{x+y+z}\epsilon_1 + \epsilon_2}_{\text{errore algoritmico}} \\ &= E_{ine} + E_{alg}.\end{aligned}$$

- La valutazione di un algoritmo numerico deve tenere conto non solo dell'efficienza (complessità computazionale), ma anche della stabilità.
- Particolari cautele vanno adottate nell'interpretare i risultati di un algoritmo numerico applicato ad un problema mal condizionato.

6. Errori numerici: un tragico caso di cronaca bellica

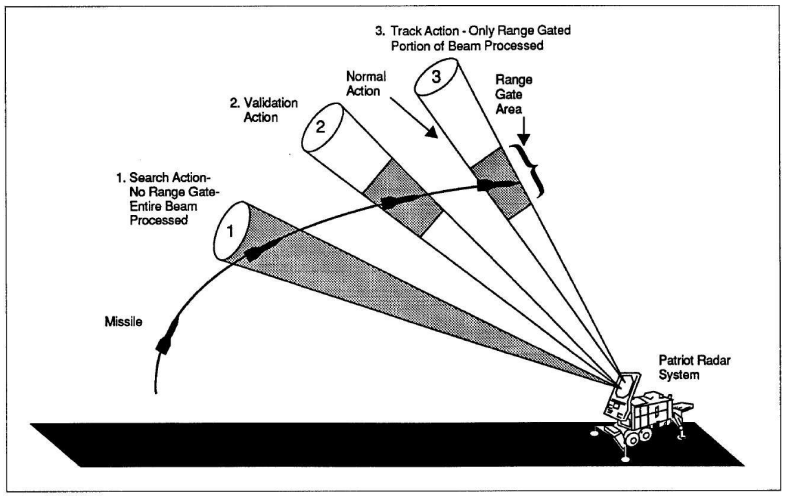
Il sistema MIM-104 Patriot

- Il sistema MIM-104 Patriot è un sistema missilistico terra-aria mobile per la difesa aerea. Fu progettato nel 1969, prodotto per la prima volta nel 1976 dall'azienda statunitense Raytheon e usato dalle forze armate USA durante la Guerra del Golfo (1991).
- Ogni batteria Patriot consta di sei lanciamissili, un'unità radar, una stazione di controllo ed un gruppo di comunicazione di supporto.
- Il fulcro del sistema Patriot risiede nel suo computer centrale per il controllo delle armi, il quale esegue le funzioni principali per il monitoraggio e l'intercettazione dei missili nemici. Tale computer è basato su un'architettura degli anni '70, con una capacità relativamente limitata di eseguire calcoli ad alta precisione.

Funzionamento del sistema di intercettazione

Il computer centrale fa uso del radar in dotazione al sistema Patriot per identificare, intercettare ed eventualmente abbattere i missili della fazione contrapposta.

1. Il radar invia impulsi elettronici per sondare lo spazio aereo soprastante, alla ricerca di obiettivi sensibili.
2. Se viene identificato un oggetto che ha le caratteristiche del missile nemico, il computer calcola un'area nella quale l'eventuale missile dovrebbe trovarsi nella rilevazione radar successiva, sulla base della sua velocità (nota a priori) e dell'istante di tempo in cui è avvenuta l'individuazione dell'oggetto.
3. Infine, il radar imposta l'area calcolata dal computer ed invia un impulso in quella direzione: se l'oggetto si trova nell'area, ciò conferma che si tratta di un missile nemico; in tal caso, il sistema Patriot dispone il lancio di un missile contro l'oggetto nemico.



Il fallimento del sistema Patriot nella Guerra del Golfo

- Il 25 Febbraio 1991 una batteria contraerea Patriot situata a Dharam, in Arabia Saudita, non fu in grado di intercettare e abbattere un missile SCUD lanciato dagli iracheni. Il missile non intercettato colpì una caserma statunitense, uccidendo 28 soldati e ferendone più di 100.

- Cosa non funzionò?

- Un'inchiesta del 1992 della Camera dei Rappresentanti degli USA appurò che la causa del fallimento fu dovuta agli errori dovuti all'aritmetica finita utilizzata dal computer del sistema Patriot.

Schema della procedura di intercettazione (semplificato)

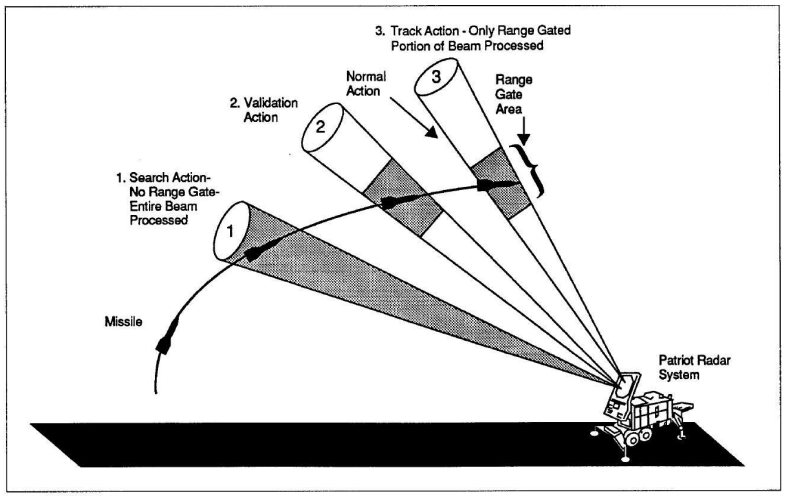
Dati

- t: istante in cui l'oggetto viene individuato la prima volta tramite un impulso radar;
- Δt : intervallo di tempo tra due impulsi radar (noto e costante);
- v: velocità di un missile nemico (nota e costante)

la posizione s dell'oggetto al tempo $t + \Delta t$ viene calcolata come

$$s = v \cdot (t + \Delta t).$$

Se all'istante $t + \Delta t$ l'impulso radar rileva ancora l'oggetto in un'"area" $[s - \epsilon, s + \epsilon]$, allora il sistema si prepara al lancio del missile di difesa.



Schema dell'algoritmo

- Il computer centrale tiene in memoria un numero intero $T \in \mathbb{N}$ rappresentante il tempo, in decimi di secondo, trascorso dalla messa in funzione della batteria.

- Per poter calcolare la posizione successiva dell'oggetto aereo individuato, il computer necessita del tempo espresso in secondi $t \in \mathbb{R}$.

- Il valore t in secondi si ottiene moltiplicando T per il valore $\frac{1}{10}$.

$$\begin{aligned} d &\leftarrow \frac{1}{10} \\ t &\leftarrow d \cdot T \\ s &\leftarrow v \cdot (t + \Delta t) \end{aligned}$$

Un errore nella memorizzazione del tempo

Il valore $\frac{1}{10}$ viene rappresentato in precisione finita, prendendo le prime 23 cifre della sua rappresentazione binaria *non normalizzata*

$$(0.1)_{10} = (0.000\overline{1100})_2 \simeq \underbrace{(0.00011001100110011001100)_2}_{=d}.$$

L'errore assoluto commesso nell'approssimare $\frac{1}{10}$ con d è dato da

$$E_a = \left| d - \frac{1}{10} \right| \simeq 9.54 \cdot 10^{-8}.$$

Analisi dell'algoritmo

$$\begin{aligned} d &\leftarrow \frac{1}{10} \\ t &\leftarrow d \cdot T \\ s &\leftarrow v \cdot (t + \Delta t) \end{aligned}$$

- 1° passo: errore di d rispetto a $\frac{1}{10}$

$$\left| d - \frac{1}{10} \right| \simeq 9.54 \cdot 10^{-8}.$$

Per semplicità assumiamo che non vi siano altri errori sui dati.

- 2° passo: errore di t (val. calcolato) rispetto a $t^* = \frac{T}{10}$ (val. esatto)

$$|t - t^*| = \left| \frac{1}{10} T - dT \right| = \left| \frac{1}{10} - d \right| \cdot T.$$

- 3° passo: errore di s (val. calcolato) rispetto a $s^* = v(t^* + \Delta t)$ (val. esatto)

$$|s - s^*| = |v(t^* + \Delta t) - v(t + \Delta t)| = v|t^* - t| = v \left| \frac{1}{10} - d \right| \cdot T \simeq vT \cdot 9.54 \cdot 10^{-8}.$$

$$|s - s^*| \simeq vT \cdot 9.54 \cdot 10^{-8}.$$

- L'errore iniziale viene amplificato dal fattore vT.
- Se vT è grande, l'errore finale è grande.

Il caso reale

La velocità dei missili SCUD è

$$v = 1676\ m/s.$$

Al momento dell'episodio, la batteria era in funzione da oltre 100 ore:

$$T = 3600000.$$

L'errore sul calcolo della posizione fu quindi di

$$|s - s^*| \simeq 575.6054\ m.$$

Di conseguenza, l'impulso radar all'istante $t + \Delta t$ fu indirizzato in un'area nella quale il missile nemico non era presente.

7. Errori numerici: la derivazione numerica in MATLAB

- **Problema**
  Calcolare il valore approssimato della derivata di una funzione f in un punto.

- **Approccio numerico**
  Eseguire su calcolatore un algoritmo che, preso in ingresso un punto a e l'espressione analitica della funzione f, restituisce in uscita il valore di f' in a, ossia $f'(a)$.

- **Algoritmo**
  Approssimare la derivata con un rapporto incrementale

$$f'(a) := \lim_{h \to 0} \frac{f(a+h) - f(a)}{h} \quad \Rightarrow \quad f'(a) \simeq \frac{f(a+h) - f(a)}{h}$$

  con h "sufficientemente piccolo".

- **Verifica sperimentale**
  Verifichiamo la bontà dell'idea scrivendo un programma MATLAB per un caso in cui conosciamo la soluzione esatta del problema, in modo da poter valutare l'accuratezza della soluzione calcolata (problema test).

Problema test

- Scegliamo la funzione

$$f(x) = \sin(x)$$

la cui derivata prima è data da

$$f'(x) = \cos(x).$$

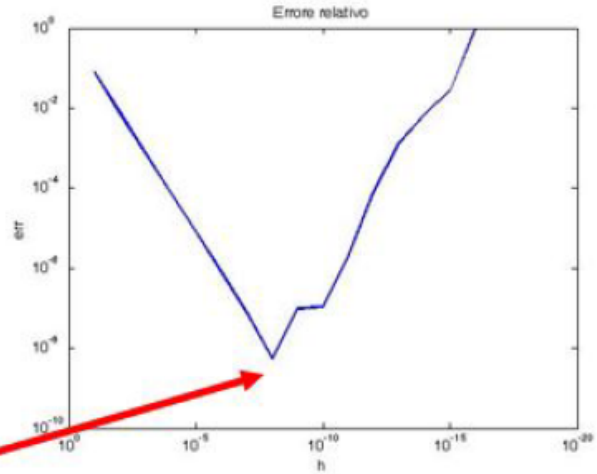
Il problema consiste nel calcolare numericamente f' in $a = 1$.

Soluzione esatta	Soluzione calcolata
$\alpha = \cos(\alpha)$	$\alpha* = \frac{\sin(a+h)-\sin(a)}{h}$

- Scriviamo un programma MATLAB che calcola diversi valori di α^* corrispondenti a diverse scelte di h:

$$h \in \{10^{-1}, 10^{-2}, \ldots, 10^{-16}\}.$$

h	sol	der	err
1e-01	0.54030	0.49736	7.9471e-02
1e-02	0.54030	0.53609	7.8036e-03
1e-03	0.54030	0.53988	7.7887e-04
1e-04	0.54030	0.54026	7.7872e-05
1e-05	0.54030	0.54030	7.7871e-06
1e-06	0.54030	0.54030	7.7872e-07
1e-07	0.54030	0.54030	7.7415e-08
1e-08	0.54030	0.54030	5.4967e-09
1e-09	0.54030	0.54030	9.7244e-08
1e-10	0.54030	0.54030	1.0824e-07
1e-11	0.54030	0.54030	2.1631e-06
1e-12	0.54030	0.54035	8.0030e-05
1e-13	0.54030	0.53957	1.3583e-03
1e-14	0.54030	0.54401	6.8609e-03
1e-15	0.54030	0.55511	2.7409e-02
1e-16	0.54030	0.00000	1.0000e+00



- Ci saremmo aspettati che quanto più h fosse piccolo, tanto più accurata sarebbe stata la soluzione calcolata. Infatti, se $f \in C^2$ e $|f''(y)| \leq M$ per ogni $y \in \mathbb{R}$, abbiamo per il Teorema di Taylor che

$$\left| f'(a) - \frac{f(a+h) - f(a)}{h} \right| \leq \frac{M}{2} h \overset{h \to 0}{\longrightarrow} 0.$$

- Contrariamente alla nostra intuizione e alla teoria, si ha un valore minimo dell'errore in corrispondenza di $h = 10^{-8}$, mentre per valori di h più piccoli l'errore è più grande.

- Perché?

- Assumiamo per semplicità di avere soltanto gli errori di rappresentazione dei valori $f(a)$ ed $f(a+h)$ (no errori nelle operazioni) e che questi errori siano maggiorati entrambi da una costante δ:

$$|fl(y) - y| \leq \delta.$$

- In realtà l'algoritmo sta calcolando il valore approssimato

$$\alpha^* = \frac{fl(f(a+h)) - fl(f(a))}{h}.$$

- L'errore assoluto che commettiamo è dunque

$$E_a = |\alpha - \alpha^*| = \left| f'(a) - \frac{fl(f(a+h)) - fl(f(a))}{h} \right|.$$

Supponendo $|fl(y) - y| \leq \delta$ e ricordando che $\left| f'(a) - \frac{f(a+h)-f(a)}{h} \right| \leq \frac{M}{2}h$, si ottiene la seguente analisi dell'errore:

$$
\begin{aligned}
E_a &= \left| f'(a) - \frac{fl(f(a+h)) - fl(f(a))}{h} \right| \\
&= \left| f'(a) - \frac{f(a+h)-f(a)}{h} + \frac{f(a+h)-f(a)}{h} - \frac{fl(f(a+h)) - fl(f(a))}{h} \right| \\
&\leq \left| f'(a) - \frac{f(a+h)-f(a)}{h} \right| + \left| \frac{f(a+h)-f(a)}{h} - \frac{fl(f(a+h)) - fl(f(a))}{h} \right| \\
&\leq \frac{M}{2}h + \left| \frac{fl(f(a+h)) - f(a+h)}{h} - \frac{fl(f(a)) - f(a)}{h} \right| \\
&\leq \frac{M}{2}h + \left| \frac{fl(f(a+h)) - f(a+h)}{h} \right| + \left| \frac{fl(f(a)) - f(a)}{h} \right| \leq \frac{M}{2}h + \frac{2\delta}{h}.
\end{aligned}
$$

Si conclude quindi che

$$E_a \simeq \frac{M}{2}h + \frac{2\delta}{h} \xrightarrow{h \to 0} \infty \text{ (!!!)}$$

- Chiamiamo ψ la funzione che maggiora (e approssima bene) l'errore

$$\psi(h) = \frac{M}{2}h + \frac{2\delta}{h}.$$

- Il minimo di ψ (dell'errore) si trova per il valore di h che risolve l'equazione

$$\psi'(h) = 0 \quad \Leftrightarrow \quad \frac{M}{2} - \frac{2\delta}{h^2} = 0 \quad \Rightarrow \quad \bar{h} = 2\sqrt{\frac{\delta}{M}}.$$

- Nel nostro problema test abbiamo $M = 1$ e $\delta \simeq u = 10^{-16}$, dunque

$$\bar{h} \simeq 10^{-8}$$

Ciò spiega il motivo per cui l'errore relativo minimo si ottiene proprio in corrispondenza del valore 10^{-8}.