Numeri finiti ed aritmetica con i numeri finiti

Un calcolatore numerico è in grado di **rappresentare soltanto un numero finito di cifre**; ne consegue la possibilità che un numero reale introdotto nel calcolatore venga approssimato; le operazioni elementari eseguite su tali numeri possono, a loro volta, produrre risultati non rappresentabili esattamente nel calcolatore. Pertanto, **quando un algoritmo viene eseguito su un calcolatore, si ha in generale una successiva creazione e propagazione di errori**. Tali errori sono comunemente chiamati **errori di arrotondamento**, dal nome di una tecnica usuale per rappresentare i numeri reali sul calcolatore.

Il risultato prodotto dall'algoritmo **differisce, quindi, in generale dal risultato esatto**, cioè da **quel risultato ideale che si potrebbe ottenere operando con tutte le cifre richieste.** Senza un'idea, più precisamente una maggiorazione della differenza dei due risultati, il risultato numerico può essere del tutto illusorio. Infatti, esso può dipendere dal numero di cifre usate e/o dall'ordine in cui vengono effettuate le operazioni (cioè dal particolare algoritmo utilizzato).

Nel seguito, per esaminare l'origine degli errori si analizzeranno brevemente alcune tecniche usuali di **rappresentazione dei numeri reali su un calcolatore numerico**, in modo da **definire l'insieme dei cosiddetti numeri macchina**. Successivamente si definiranno **per tali numeri le operazioni elementari**, cioè **le operazioni macchina e si studierà il loro comportamento rispetto alla propagazione degli errori**. Si analizzerà quindi, più in generale il comportamento, rispetto agli errori, di una successione di operazioni macchina, cioè di un algoritmo. Importanti, in questo contesto, saranno **i concetti di stabilità di un algoritmo e di condizionamento di un problema**

Teorema: Rappresentazione dei Numeri Reali: Notazione scientifica normalizzata

Ogni numero reale $\alpha \neq 0$ può esprimersi univocamente, in base β, nella seguente forma:

$$\alpha = \pm(a_1\beta^{-1} + a_2\beta^{-2} + \ldots\ldots) \cdot \beta^p$$
$$= sign(\alpha) \cdot (\sum_{i=1}^{+\infty} a_i \beta^{-i}) \cdot \beta^p = sign(\alpha) \cdot m \cdot \beta^p \qquad (1)$$

dove

- $sign(x) = \begin{cases} 1 & se\ x > 0 \\ -1 & se\ x < 0 \end{cases}$ (funzione segno)
- $p \in \mathbb{Z}$,
- le cifre a_i sono numeri interi tali che:

$$0 \leq a_i \leq \beta - 1, \quad i = 1,2,\ldots. \qquad \boldsymbol{a_1 \neq 0}$$

Il numero $m = a_1\beta^{-1} + a_2\beta^{-2} + \cdots \ldots$ si chiama **mantissa** di α e soddisfa la condizione

$$\boldsymbol{\frac{1}{\beta} \leq m < 1}.$$

β^p si definisce **parte esponente** di α e p si chiama **esponente** di α.

Usando la **notazione posizionale** il numero $\alpha \neq 0$ si rappresenta

$$\alpha = \pm 0.a_1a_2a_3\ldots\beta^p$$

Esempio: Il numero reale π può essere espresso in base β=10 nella forma

$\pi = +(3 \cdot 10^{-1} + 1 \cdot 10^{-2} + 4 \cdot 10^{-3} + 1 \cdot 10^{-4} + 5 \cdot 10^{-5} + 9 \cdot 10^{-6} \ldots\ldots)10^1$

e in **notazionale posizionale** diviene

$$\pi = +(0.314159......)10^1 = 3.14159$$

1. I Numeri Finiti: Il Sistema Floating Point

Un **numero reale è caratterizzato da segno, esponente e mantissa**, in cui la mantissa può essere costituita da un numero illimitato di cifre.

Lavorando con il **calcolatore, comunque, ognuna di queste parti deve essere limitata, ovvero rappresentata da un numero finito di bits**. Da ciò nasce l'impossibilità di rappresentare nel calcolatore l'insieme dei numeri reali. Occorre, quindi, definire una rappresentazione approssimata dei numeri reali costituita da numeri la cui mantissa è rappresentata da un numero finito di cifre, che indichiamo con t.

Def: Insieme Floating Point

Ad ogni calcolatore che utilizza la base di rappresentazione β, usa t cifre per rappresentare la mantissa, ed in cui L ed U rappresentano il minimo ed il massimo esponente rappresentabile, (L<0, U>0) si associa l' insieme dei Numeri di Macchina (**Insieme Floating Point**), cioè l'insieme dei numeri reali esattamente rappresentabili con quel calcolatore.

Esso è così definito:

$$F(\beta, t, L, U) = \{0\} \cup \left\{ \alpha \in \Re \setminus \{0\} : \alpha = sign(\alpha) \left(\sum_{i=1}^{t} a_i \beta^{-i} \right) \beta^p \right\}$$

$0 \le a_i \le \beta - 1, \quad i = 1,2,....\qquad a_1 \ne 0$, L≤p≤U, L<0, U>0

Quando inseriamo un numero reale α in un calcolatore, poiché per la rappresentazione dell' esponente vengono riservati solo un numero limitato di bits, si possono verificare i seguenti casi:

1. Se $p \notin [L,U]$. Il numero non può essere rappresentato nel calcolatore.
    - Se p<L si verifca un **underflow** e solitamente si assume lo zero come valore approssimato, e ciò viene solitamente segnalato mediante warning.
    - Se p>U si verifica un **overflow;** non si approssima, ma si segnala l'evento con un arresto del calcolo.

2. Nel caso in cui $p \in [L,U]$, cioè il numero α sia rappresentabile nel calcolatore, si possono verificare due casi:
    a. le cifre a_i, per i>t sono tutte nulle, cioè α è rappresentabile esattamente con un numero finito t di cifre.
    b. le cifre a_i, per $i>t$ non sono tutte nulle. In questo caso la rappresentazione di α sul calcolatore è data da $fl(\alpha)$, che si esprime nella forma,

$$fl(\alpha) = \begin{cases} 0 & se\ \alpha = 0 \\ \pm\ m_t \beta^p & se\ \alpha \neq 0 \end{cases}$$

$$m_t = a_1\beta^{-1} + a_2\beta^{-2} + \cdots \ldots + \tilde{a}_t\beta^{-t} \quad con\ a_1 \neq 0.$$

dove

- nel caso di troncamento $\tilde{a}_t = a_t$;
- nel caso di arrotondamento (usato se β è pari) :

  si aggiunge $\frac{1}{2}\beta^{-t}$ alla mantissa e poi si tronca alla t-esima cifra,

In notazione posizionale si può esprimere come

$$fl(\alpha) = \pm 0.a_1a_2\ldots\tilde{a}_t\beta^p.$$

- ES:

  $\alpha = 0.54361$ t = 3

Troncamento $fl(\alpha)$=0.543

Arrotondamento $fl(\alpha)$=Tronc(0.54361+0.0005)=Tronc(0.5441)=0.544

Arrotondamento ai pari (rounding to even)

E' una regola speciale dell'approssimazione per arrotondamento.

Si applica quando un reale α è esattamente equidistante da due numeri finiti consecutivi, ossia quando

$$a_{t+1} = \frac{\beta}{2} \quad \text{e} \quad a_i = 0 \ \forall i > t+1.$$

Il rounding to even impone di prendere come fl(α) il numero di F che ha l'ultima cifra di mantissa pari. In altre parole, il rounding to even comporta l'incremento di a_t di 1 se a_t e dispari, ma di non incrementarlo se è pari.

Esempi:

β = 10, t = 2, α = 0.185, fl(α) = 0.18

β = 10, t = 4, α= 0.37975, fl(α) = 0.3798

β = 2, t = 4, α = 0.101110, fl(α) = 0.1100

Numero di macchina più piccolo rappresentabile in $F(\beta, t, L, U)$

La mantissa più piccola che si può rappresentare in base β e con t cifre è

$$1\cdot\beta^{-1}+0\cdot\beta^{-2}+....0\cdot\beta^{-t}=\beta^{-1}$$

La mantissa più grande è

$$(\beta-1)\cdot\beta^{-1}+(\beta-1)\cdot\beta^{-2}+....(\beta-1)\cdot\beta^{-t}=\beta^{0}-\beta^{-1}+\beta^{-1}-\beta^{-2}+...+\beta^{-t+1}-\beta^{-t}=1-\beta^{-t}$$

Il numero di macchina più piccolo che si può scrivere in $F(\beta,t,L,U)$ avrà quindi la mantissa più piccole tra le possibili e la parte esponente con l' esponente minore possibile, L, cioè

$$\alpha_{\min}=\beta^{-1}\beta^{L}=\beta^{L-1}.$$

Numero di macchina più grande rappresentabile in $F(\beta,t,L,U)$

Il numero di macchina più grande che si può rappresentare in $F(\beta,t,L,U)$ avrà quindi la mantissa più grande tra le possibili e la parte esponente con esponente massimo U, cioè

$$\alpha_{\max}=(1-\beta^{-t})\beta^{U}=\beta^{U}-\beta^{U-t}$$

- Es: base $\beta = 10$, t =3 cifre per la mantissa,

  la mantissa più piccola è 0.1000= 10^{-1}

  la mantissa più grande è 0.999=$1-10^{-3}$

L'insieme $F(\beta,t,L,U)$ è un sottoinsieme di R (insieme di tutti i numeri reali) ed ha cardinalità finita, come ci mostra il seguente teorema.

<u>Teorema</u>: Cardinalità dei numeri floating point

Sia $F(\beta,t,L,U)$ l'insieme dei numeri floating point.

Allora esso possiede *fl*(0) e $(\beta-1)\cdot\beta^{t-1}$ (U-L+1) numeri positivi non uniformemente distribuiti in $[\beta^{L-1},\beta^{U})$ e altrettanti numeri negativi non uniformemente distribuiti in $(-\beta^{U},-\beta^{L-1}]$, ovvero

$$\#F = 2\cdot(\beta-1)\beta^{t-1}(U-L+1)+1$$

Dim:

Ragioniamo per i numeri positivi:

Poiché con *t* cifre in base β si possono formare β^{t} mantisse diverse, e fra questi vanno esclusi quelli la cui prima cifra è nulla, il numero di mantisse totali è dato da $\beta^{t}-\beta^{t-1}$.

Poiché per ogni mantissa si hanno (U-L+1) esponenti possibili

$\Rightarrow$

$$\#\text{ elementi positivi in } F=(\beta-1)\,\beta^{t-1}\,(U-L+1)$$

$$\#F=2\cdot(\beta-1)\,\beta^{t-1}\,(U-L+1)+1.$$

■

- Es:

$\beta=2$, t=3, L=-1 e U=2

Le possibili mantisse sono:

$$m = .100,\quad m = .101,\quad m = .110,\quad m = .111$$

Ogni mantissa è moltiplicata per $\beta^p \in \{2^{-1}, 2^0, 2^1, 2^2\}$

$\Rightarrow$ 4 mantisse possibili, 4 possibili esponenti

Da cui 16 numeri positivi, 16 negativi e lo zero

Da cui segue che #F=33

Tali numeri sono:

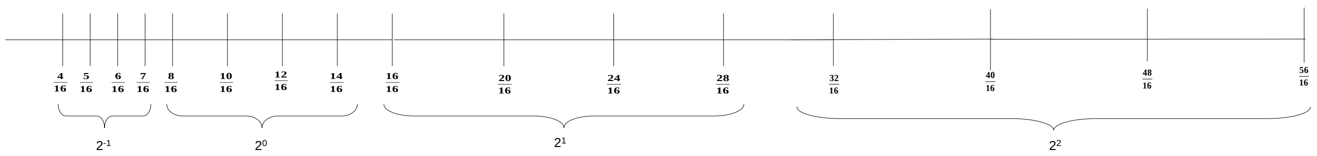
$$\left.\begin{aligned}
.100\ 2^{-1} &= (1*2^{-1})\ 2^{-1} = 1/2\cdot 1/2 = 4/16\\
.101\ 2^{-1} &= (1*2^{-1}+1*2^{-3})\ 2^{-1} = 5/16\\
.110\ 2^{-1} &= (1*2^{-1}+1*2^{-2})\ 2^{-1} = 6/16\\
.111\ 2^{-1} &= (1*2^{-1}+1*2^{-2}+1*2^{-3})\ 2^{-1} = 7/16
\end{aligned}\right\}\ \text{4 numeri relativi a } 2^{-1}$$

$$\left.\begin{aligned}
.100\ 2^{0} &= (1*2^{-1})\,1 = 1/2 = 8/16\\
.101\ 2^{0} &= (1*2^{-1}+1*2^{-3})\,1 = 10/16\\
.110\ 2^{0} &= (1*2^{-1}+1*2^{-2})\,1 = 12/16\\
.111\ 2^{0} &= (1*2^{-1}+1*2^{-2}+1*2^{-3})\,1 = 14/16
\end{aligned}\right\}\ \text{4 numeri relativi a } 2^{0}$$

$$\left.\begin{aligned}
.100\ 2^{1} &= (1*2^{-1})\ 2 = 1/2\cdot 2 = 16/16\\
.101\ 2^{1} &= (1*2^{-1}+1*2^{-3})\ 2 = 20/16\\
.110\ 2^{1} &= (1*2^{-1}+1*2^{-2})\ 2 = 24/16\\
.111\ 2^{1} &= (1*2^{-1}+1*2^{-2}+1*2^{-3})\ 2 = 28/16
\end{aligned}\right\}\ \text{4 numeri relativi a } 2^{1}$$

$$\left.\begin{aligned}
.100\ 2^{2} &= (1*2^{-1})\ 4 = 1/2\cdot 4 = 32/16\\
.101\ 2^{2} &= (1*2^{-1}+1*2^{-3})\ 4 = 40/16\\
.110\ 2^{2} &= (1*2^{-1}+1*2^{-2})\ 4 = 48/16\\
.111\ 2^{2} &= (1*2^{-1}+1*2^{-2}+1*2^{-3})\ 4 = 56/16
\end{aligned}\right\}\ \text{4 numeri relativi a } 2^{2}$$

Tali numeri sono equispaziati relativamente ad ogni singola base ma non globalmente.



Spacing

Calcoliamo la distanza tra due numeri consecutivi di F appartenenti a $[\beta^p, \beta^{p+1}]$.

Ricordiamo che possiamo scrivere $\beta^p = 0.10000 \ldots 00\ \beta^{p+1} = (1 \cdot \beta^{-1})\ \beta^{p+1}$

Il numero finito consecutivo a β^p può essere scritto come

$0.100000\ldots01\ \beta^{p+1} = (1 \cdot \beta^{-1} + 1 \cdot \beta^{-t})\beta^{p+1}$.

Calcoliamo la differenza

s:= $(1 \cdot \beta^{-1} + 1 \cdot \beta^{-t})\beta^{p+1} - (1 \cdot \beta^{-1})\ \beta^{p+1} = \beta^p + \beta^{p+1-t} - \beta^p = \boldsymbol{\beta^{p+1-t}}$

s prende il nome di formula dello **spacing** tra β^p e β^{p+1}.

Esempi: β = 2, t = 53

- I numeri di F in $[2^{52}, 2^{53}]$ sono solo gli interi

(p = 52 ⇒ s = $2^{52+1-53} = 1$)

- I numeri di F in $[2^{53}, 2^{54}]$ sono solo gli interi pari

(p = 53 ⇒ s = $2^{53+1-53} = 2$)

F non è una perfetta simulazione di R

1) I numeri di F non sono uniformemente distribuiti sull'asse reale.
2) La loro distribuzione è uniforme tra due potenze successive di β.
3) La loro densità decresce con l'aumentare del valore assoluto del numero.

Precisione di macchina (eps)

eps è lo spacing tra β^0 e β^1 (p = 0) ⇒ eps = β^{1-t}

u (roundoff unit): u := $\frac{1}{2}eps = \frac{1}{2}\ \beta^{1-t}$

Poiché in un calcolatore il numero reale $\alpha \neq 0$ è sostituito con $fl(\alpha)$, è fondamentale **stabilire una stima della differenza tra α e $fl(\alpha)$ relativa al valore di α stesso.**

Definizione: Si definisce **Errore Relativo** di arrotondamento di $\alpha \neq 0$ la quantità

$$\frac{|\alpha - fl(\alpha)|}{|\alpha|}$$

Teorema: Sia $\alpha \neq 0$ un numero reale rappresentato in base β come

$$\alpha = \pm\left(\sum_{i=1}^{+\infty} a_i \beta^{-i}\right) \cdot \beta^p, \quad a_1 \neq 0, \quad p \in [L, U]$$

Allora, se non si verifica un Overflow, si ha:

$$\frac{|\alpha - fl(\alpha)|}{|\alpha|} \leq K \cdot \beta^{1-t} = K \cdot\ eps \qquad (3)$$

Con K=1 nel caso di troncamento

K=1/2 nel caso di arrotondamento

Dimostrazione:

a) Consideriamo il caso del troncamento

$\alpha = \pm 0.a_1 a_2 \ldots . \beta^p \qquad fl(\alpha) = \pm 0.a_1 a_2 \ldots a_t \beta^p$

$|\alpha - fl(\alpha)| = .00000 a_{t+1} a_{t+2} \ldots \beta^p = .a_{t+1} a_{t+2} \ldots \beta^{-t} \beta^p$

$$\frac{|\alpha - fl(\alpha)|}{|\alpha|} = \frac{0.a_{t+1} a_{t+2} \ldots \beta^{-t} \beta^p}{.a_1 a_2 \ldots \ldots . \beta^p}$$

Poiché $0.a_{t+1} a_{t+2} \ldots < 1 \quad e \quad 0.a_1 a_2 \ldots \ldots \geq \beta^{-1}$

$$\Rightarrow \quad \frac{|\alpha - fl(\alpha)|}{|\alpha|} \leq \frac{\beta^{-t}}{\beta^{-1}} = \beta^{1-t}$$

In modo analogo si dimostra per il caso dell'arrotondamento.

■

La quantità $eps = \beta^{1-t}$ è detta **precisione di macchina** nel sistema floating point: **eps è il più piccolo numero positivo di macchina tale che sommato all'unità rende una quantità più grande di 1**:

$$\boldsymbol{fl(1+eps)>1}$$

Se poniamo $\varepsilon = \frac{fl(\alpha) - \alpha}{\alpha}$, dalla (3) si ha

$$|\varepsilon| \leq eps \qquad \text{e} \qquad fl(\alpha) = \alpha(1+\varepsilon)$$

Cioè il numero finito *fl(α)* è una perturbazione del numero reale α corrispondente.

N.B. In doppia precisione t=53 corrisponde ad avere circa 16 cifre decimali significative, infatti u = $\frac{1}{2} 2^{-52} = 2^{-53} \cong 10^{-16}$

Standard IEEE 754

Lo standard IEEE 754, pubblicato nel 1985, (versione attuale IEEE 754-2008) definisce un sistema aritmetico floating point binario ed è adottato dalla maggior parte delle case costruttrici di elaboratori.

Un numero in virgola mobile, secondo lo standard IEEE, su un calcolatore è rappresentato in base β=2, su parole di 32 (singola precisione), 64 (doppia precisione) o 128 bit (precisione quadrupla) divisi in tre parti:

- un bit di **segno** *s*;
- un campo di **esponente** p^*;
- un campo di **mantissa** *m*

in questo ordine.

Di seguito è rappresentato un numero in una stringa di **32 bit:**

```
 1      8                 23                 lunghezza in bit
+-+--------+-----------------------+
|S|  Esp.  |       Mantissa m       |
+-+--------+-----------------------+
```

Il campo ***s*** specifica il segno del numero: **0 per i numeri positivi**, **1 per i numeri negativi**.

Nel campo p^* viene memorizzato l'esponente *p* del numero da rappresentare più un bias, $2^{e-1}-1$, dove ***e*** è il numero di bit riservato all'esponente. Quindi $p^*=p+127$, $0 \le p^* \le 255$.

I valori 0 e 255 vengono utilizzati per rappresentare valori speciali.

I valori da 1 a 254 corrispondono a $-126 \le p \le 127$. Quindi L=-126 e U=127.

Esponente 0, Mantissa 0 → rappresentazione dello zero

Esponente 255, Mantissa 0 → rappresentazione infinito

Esponente 255, Mantissa non zero → rappresentazione NaN (not a number, per rappresentare forme indeterminate del tipo 0/0 oppure inf/inf)

Di seguito è rappresentato un numero in una stringa **di 64 bit:**

```
 1     11                   52                   lunghezza in bit
+-+-----------+-----------------------------------------------------+
|S|  Esp.   |                         Mantissa m                  |
+-+-----------+-----------------------------------------------------+
```

Il campo ***s*** specifica il segno del numero: **<u>0 per i numeri positivi</u>**, **1 per i numeri negativi**.

Nel campo p^* viene memorizzato l'esponente p del numero da rappresentare più un bias, $2^{e-1} - 1$, dove ***e*** è il numero di bit riservato all'esponente.

Quindi $p^*=p+1023$, $0 \le p^* \le 2047$.

I valori 0 e 2047 vengono utilizzati per rappresentare valori speciali.

I valori da 1 a 2046 corrispondono a $-1022 \le p \le 1023$. Quindi L=-1022 e U=1023.

- Poiché la mantissa è normalizzata, e si sa che il primo bit è diverso da zero ed uguale ad 1, non viene memorizzato, ma si considera come se ci fosse. Si usa la tecnica dell'**hidden bit**, e si guadagna un bit e si considera la mantissa con 1 bit in più. Quindi **la mantissa che si deve considerare è *M*=1.*m***

Esempio:

Utilizzando la rappresentazione standard IEEE per numeri floating point su 32 bit, si determini il valore decimale N della sequenza di bit

0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Esponente p*

mantissa **m**

segno

Il segno è +.

$p^* = (01111110)_2 = 126$

$p=126-127=-1$

M=1.m=1.10000000000000000000000

Il valore decimale corrispondente alla sequenza di bit in memoria è:

$segno \cdot M \cdot 2^p = (1.1)_2 \cdot 2^{-1} = (0.11)_2 = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 0.5 + 0.25 = 0.75$

L’insieme F di Python

`sys.float_info`

A named tuple holding information about the float type. It contains low level information about the precision and internal representation. The values correspond to the various floating-point constants defined in the standard header file `float.h` for the ‘C’ programming language; see section 5.2.4.2.2 of the 1999 ISO/IEC C standard [C99], ‘Characteristics of floating types’, for details.

attribute	float.h macro	explanation
`epsilon`	`DBL_EPSILON`	difference between 1.0 and the least value greater than 1.0 that is representable as a float See also `math.ulp()`.
`dig`	`DBL_DIG`	maximum number of decimal digits that can be faithfully represented in a float; see below
`mant_dig`	`DBL_MANT_DIG`	float precision: the number of base-`radix` digits in the significand of a float
`max`	`DBL_MAX`	maximum representable positive finite float
`max_exp`	`DBL_MAX_EXP`	maximum integer *e* such that `radix**(e-1)` is a representable finite float
`max_10_exp`	`DBL_MAX_10_EXP`	maximum integer *e* such that `10**e` is in the range of representable finite floats
`min`	`DBL_MIN`	minimum representable positive *normalized* float Use `math.ulp(0.0)` to get the smallest positive *denormalized* representable float.
`min_exp`	`DBL_MIN_EXP`	minimum integer *e* such that `radix**(e-1)` is a normalized float
`min_10_exp`	`DBL_MIN_10_EXP`	minimum integer *e* such that `10**e` is a normalized float
`radix`	`FLT_RADIX`	radix of exponent representation
`rounds`	`FLT_ROUNDS`	integer representing the rounding mode for floating-point arithmetic. This reflects the value of the system `FLT_ROUNDS` macro at interpreter startup time: `-1` indeterminable, `0` toward zero, `1` to nearest, `2` toward positive infinity, `3` toward negative infinity All other values for `FLT_ROUNDS` characterize implementation-defined rounding behavior.

max=1.7976931348623157e+308,

max_exp=1024, (U=1023)

max_10_exp=308,

min=2.2250738585072014e-308,

min_exp=-1021, (L=-1022)

min_10_exp=-307,

dig=15,

mant_dig=53, (t=53)

epsilon=2.220446049250313e-16, (spacing in [1,2])

radix=2, ($\beta = 2$)

rounds=1)

2. ARITMETICA IN VIRGOLA MOBILE

Dato l'insieme dei numeri di macchina $F(\beta,t,L,U)$, è necessario definire su di esso delle nuove operazioni aritmetiche, cioè le operazioni in aritmetica di macchina, in quanto il risultato di un'operazione aritmetica classica eseguita fra numeri di macchina può non essere un numero di macchina.

Siano quindi $fl(x)$ *e* $fl(y) \in F(\beta,t,L,U)$ e il risultato di un'operazione aritmetica tra *fl*(x) e *fl*(y), sia tale che $p \in [L,U]$

Si definiscono le operazioni di macchina (Operazioni Floating Point) e si indicano con i simboli $\oplus$, $\otimes$, $\ominus$, $\oslash$, come segue:

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

$$fl(x) \otimes fl(y) = fl(fl(x) \times fl(y))$$

$$fl(x) \ominus fl(y)=fl(fl(x)-fl(y))$$

$$fl(x) \oslash fl(y)=fl(fl(x) - fl(y))$$

Si ha che

$$fl(x) \otimes fl(y) = fl(fl(x) \times fl(y)) = (fl(x) \times fl(y))(1+\varepsilon)$$

cioè il risultato di un'operazione Floating Point è una perturbazione di quello della corrispondente operazione reale.

Ovviamente questo vale per tutte le quattro operazioni floating point.

<u>Osservazione:</u>

Per le operazioni floating point non valgono le proprietà dell'aritmetica reale: in particolare **non vale la proprietà associativa** come dimostra il seguente esempio:

- **Es:** Sia β=10, e t=8. Dati i numeri di macchina

  a=0.23371258*10^{-4}

b=0.33678429*10^2

c=-0.33677811*10^2

Si valuti

1) (a ⊕ b) ⊕ c =

$$
\begin{array}{r}
0.000000233\,71258 \cdot 10^2 \\
0.33678429 \qquad\quad \cdot 10^2 \\
\hline
0.33678452371258 \cdot 10^2
\end{array}
$$

fl(a+b)=0.33678452·10^2

$$
\begin{array}{lr}
fl(a+b) & 0.33678452 \cdot 10^2 \\
c & -0.33677811 \cdot 10^2 \\
\hline
 & 0 \cdot 00000641 10^2
\end{array}
$$

fl(*fl*(a+b)+c)= 0.64100000·10^{-3}

2) a ⊕ (b ⊕ c)

$$
\begin{array}{r}
0.33678429 \cdot 10^2 \\
-\,0.33677811 \cdot 10^2 \\
\hline
0.00000618 \cdot 10^2
\end{array}
$$

fl(b+ c)=0.61800000·10^{-3}

$$
\begin{array}{lr}
a & 0.023371258 \cdot 10^{-3} \\
fl(b+c) & 0.61800000 \quad \cdot 10^{-3} \\
\hline
 & 0.641371258 \cdot 10^{-3}
\end{array}
$$

fl(a+*fl*(b+ c))= 0.64137126·10^{-3}

Il risultato esatto è a+b+c = 0.641371258·10^{-3}

Perciò si osserva che il risultato ottimale si ottiene con la 2), mentre con la 1) si ottiene un risultato con solo 3 cifre esatte.

Poiché tutte le volte che si esegue una operazione in aritmetica floating point si ottiene un risultato approssimato, uno dei principali compiti del calcolo numerico è quello di **studiare come questi singoli errori si propagano e quale effetto danno sul risultato finale**. Consideriamo il seguente esempio di radici di equazioni di II° grado.

Si vogliono calcolare le radici di equazione del tipo: $ax^2+bx+c=0$.

La formula risolvente è:

$$x_1=\frac{-b-\sqrt{b^2-4ac}}{2a} \quad e \quad x_2=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

La formula risolutiva ha un grandissimo vantaggio: suggerisce una precisa successione di passi , i quali, partendo dai coefficienti a,b,c conducono alle radici x_1 e x_2.

In matematica pura è sottointeso che nell'algoritmo descritto si faccia uso dei numeri reali e si eseguano operazioni aritmetiche esatte, inclusa l'estrazione della radice. Come abbiamo visto però un calcolatore non è in grado di fare queste operazioni e nemmeno di memorizzare i numeri reali a,b,c in modo esatto ma deve sostituire i numeri reali con numeri in virgola mobile e le operazioni aritmetiche esatte con le operazioni floating point.

Si consideri l'equazione

$$x^2-6.433x+0.009474=0$$

e si consideri il sistema decimale **con t=4 cifre.**

Applicando la formula risolutiva si ha

$$\alpha=(6.433)^2=0.6433\cdot 10^1\times 0.6433\cdot 10^1=0.41383489\cdot 10^1$$

$$\beta=4\times 0.009474=0.4\cdot 10^1\times 0.9474\cdot 10^{-2}=0.37896\cdot 10^{-1}$$

$$\gamma=fl(\alpha)=0.4138\cdot 10^2$$

$$\delta=fl(\beta)=0.3789\cdot 10^{-1}$$

$$\gamma-\delta=0.4138\cdot 10^2-0.0003789\cdot 10^2=0.4134211\cdot 10^2$$

$$\eta=fl(\gamma-\delta)=0.4134\cdot 10^2$$

$$fl(\sqrt{\eta}) = fl(0.64296.. \cdot 10^1) = 0.6429 \cdot 10^1$$

Da cui si ottiene la radice $x_1 = \frac{-b-\sqrt{b^2-4ac}}{2a}$ pari a

$$x_1 = fl(\frac{0.6433 \cdot 10^1 - 0.6429 \cdot 10^1}{2}) = fl(\frac{0.0004 \cdot 10^1}{2})$$
$$= 0.2000 \cdot 10^{-2}$$

La sottrazione di due numeri quasi uguali ha portato alla cancellazione di cifre significative.

In aritmetica reale si sarebbe ottenuto:

$$x_1 = \frac{6.433 - \sqrt{41.383489 - 0.037896}}{2} = \frac{6.433 - \sqrt{41.345593}}{2}$$
$$= \frac{6.433 - 6.4300538}{2} = 0.0014731...$$

Il risultato ottenuto in aritmetica floating point è molto diverso da quello vero (l'errore relativo sulla soluzione è del 36%) . Ciò è dovuto in particolare all'ultimo passaggio, dove facciamo la differenza fra numeri molto simili in modulo. Questo produce il fenomeno della "**cancellazione di cifre significative**", che è un'operazione molto pericolosa, che se è possibile va evitata.

Un algoritmo più efficiente per il calcolo delle radici di un'equazione di II° grado è il seguente:

calcolo la radice che, in base al segno di b, non porta problemi con la formula risolvente,

$$x_1 = \frac{-b + sign(b)\sqrt{b^2 - 4ac}}{2a}$$

mentre l'altra radice viene calcolata tenendo presente che: $x_1 x_2 = \frac{c}{a}$ $\quad o \quad$ $x_1 + x_2 = -\frac{b}{a}$.

In generale, lo studio della propagazione degli errori è un argomento matematicamente difficile, per cui ci limitiamo a considerare solo i due casi che seguono.

Propagazione degli errori nella moltiplicazione

Assumiamo $x, y \in R$, ma $x, y \notin F$.
Pertanto, x e y devono innanzitutto essere arrotondati in F:

$x \rightarrow fl(x) = x(1 + \epsilon_x) \in F,\ \ y \rightarrow fl(y) = y\left(1 + \epsilon_y\right) \in F,$

I due numeri di F vengono quindi moltiplicati, ottenendo così:

$$fl((fl(x) \otimes fl(y)) = [x(1 + \epsilon_x) \cdot y(1 + \epsilon_y)](1 + \epsilon_p)$$

Quindi il risultato finale sarà $x \cdot y(1 + \epsilon_x)(1 + \epsilon_y)(1 + \epsilon_p)$ da confrontare con il prodotto vero xy.

$$\boldsymbol{Errore\_relativo\_prodotto} = \frac{|x \cdot y(1+\epsilon_x)(1+\epsilon_y)(1+\epsilon_p) - xy|}{|xy|}$$

semplificando per xy si ha

$$\boldsymbol{Errore\_relativo\_prodotto} = |(1 + \epsilon_x)\left(1 + \epsilon_y\right)\left(1 + \epsilon_p\right) - 1|$$

da cui si ottiene = $|\epsilon_x + \epsilon_y + \epsilon_p + \epsilon_x\epsilon_y + \epsilon_x\epsilon_p + \epsilon_y\epsilon_p + \epsilon_x\epsilon_y\epsilon_p|$

con $|\epsilon_x|, \left|\epsilon_y\right|, \left|\epsilon_p\right| \leq u$

Assumiamo adesso che la roundoff unit u sia molto più piccola di 1, in modo da poter trascurare u^2 rispetto a u eccetera (ovviamente questo è il caso interessante).

Trascurando i termini del tipo $\epsilon_x\epsilon_y + \epsilon_x\epsilon_p + \epsilon_y\epsilon_p + \epsilon_x\epsilon_y\epsilon_p$ abbiamo

$$\boldsymbol{Errore\_relativo\_prodotto} \approx |\epsilon_x + \epsilon_p + \epsilon_p|$$

Quindi qualunque siano i numeri $x, y \in R$, da cui si parte, l'errore relativo sul prodotto è sempre minore o uguale a $3u$.

Di conseguenza il prodotto è sempre una operazione sicura (stabile).

Propagazione degli errori nella somma

Assumiamo $x, y \in R$, ma $x, y \notin F$.
Pertanto, x e y devono innanzitutto essere arrotondati in F:

$x \to fl(x) = x(1+\epsilon_x) \in F,\ \ y \to fl(y) = y\left(1+\epsilon_y\right) \in F,$

I due numeri di F vengono quindi sommati, ottenendo così:

$$fl((fl(x) \oplus fl(y)) = [x(1+\epsilon_x) + y(1+\epsilon_y)](1+\epsilon_s)$$

Quindi il risultato finale sarà $[x(1+\epsilon_x) + y(1+\epsilon_y)](1+\epsilon_s)$ da confrontare con il prodotto vero $x+y$.

Errore_relativo_somma=$\frac{|[x(1+\epsilon_x)+y(1+\epsilon_y)](1+\epsilon_s)-(x+y)|}{|x+y|}$

$$\frac{|[x + x\epsilon_x + y \ + y\epsilon_y)](1+\epsilon_s) - (x+y)|}{|x+y|} =$$

$$\frac{|x + x\,\epsilon_s + x\epsilon_x + \ \ x\epsilon_x\epsilon_s + y + y\,\epsilon_s + y\,\epsilon_y \ + y\epsilon_y\epsilon_s - (x+y)|}{|x+y|} =$$

\

$$\frac{|x\epsilon_x + y\epsilon_y + (x+y)\epsilon_s + x\epsilon_x\epsilon_s + \ y\epsilon_y\epsilon_s|}{|x+y|} =$$

Trascurando i termini in $\epsilon_x\epsilon_s,\ \ \epsilon_y\,\epsilon_s$

Errore relativo somma $\approx \left|\frac{x}{x+y}\,\epsilon_x + \frac{y}{x+y}\,\epsilon_y + \epsilon_s\right| \le \left|\frac{x}{x+y}\right| u + \left|\frac{y}{x+y}\right| u + u$

Se x e y hanno lo stesso segno allora $\left|\frac{x}{x+y}\right| \le 1$ e $\left|\frac{y}{x+y}\right| \le 1$ e di conseguenza

errore relativo somma $\le 3\,u$

se x e y hanno segno diverso le quantità $\left|\frac{x}{x+y}\right|$ e $\left|\frac{y}{x+y}\right|$ possono assumere qualunque grandezza e quindi **non possiamo controllare l'errore a priori**.

Conclusione: il caso pericoloso si ha quando sommo **due quantità x e y di segno diverso, ma molto vicine in modulo**, di modo che il denominatore |x+y| è piccolo; in questo caso i fattori di amplificazione $\left|\frac{x}{x+y}\right|$ e $\left|\frac{y}{x+y}\right|$ possono essere molto grandi e il risultato totalmente inaccurato (fenomeno di cancellazione numerica).

N.B. I fattori di amplificazione $\left|\frac{x}{x+y}\right|$ e $\left|\frac{y}{x+y}\right|$ moltiplicano rispettivamente ϵ_x ed ϵ_y, che sono gli errori di arrotondamento di x e y. Quindi se moltiplico due numeri che stanno già in F per i quali $\epsilon_x = 0$, $\epsilon_y = 0$ allora questo problema non si pone.

- **Esempio 1:**

Sia β=10, t=5.

Calcolare e^x con lo sviluppo in serie $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... = \sum_{k=0}^{\infty} \frac{x^k}{k!}$ in x = -5.5.

$$e^{-5.5} = 1.0000 - 5.5000 + 15.1250 - 27.730 + 38.1290 - 41.9420 + 38.4460 - 30.208 + 20.768 - 12.692 + 6.9803 - 3.4902..........$$

Se si effettua le somma dei primi 25 termini si ottiene $e^{-5.5} \approx 0.26363 \cdot 10^{-2}$

Il valore corretto è $e^{-5.5} = 0.408677 \cdot 10^{-2}$

Quando x è negativo lo sviluppo in serie è un procedimento che numericamente porta problemi.

Poiché

$$e^{-5.5} = \frac{1}{e^{5.5}} = \frac{1}{1+5.5+15.125+27.730+38.129+.......} = 0.40865 \cdot 10^{-2}$$

Possiamo applicare la formula usando tutti termini positivi. Il risultato che otteniamo è molto più preciso.

Un’altra possibilità è quella di sommare tutti i termini positivi e tutti i termini negativi e fare un’unica differenza finale.

Due disastri causati dall'aritmetica a precisione finita del calcolatore

Di seguito vengono riportati due episodi realmente accaduti.

Lo scopo è quello di dimostrare che lavorando in un sistema aritmetico a precisione finita la facilità di commettere errori, sia pur piccoli, può compromettere un risultato le cui conseguenze non si limitano purtroppo al solo errore numerico.

Un Missile Patriot manca il bersaglio

Nel **1991** durante la guerra del golfo un **missile Patriot** cadde su una caserma americana a causa di un errore di arrotondamento nel calcolo del tempo dal momento del lancio. Il tempo in decimi di secondi come viene misurato dal clock interno fu moltiplicato per 1/10 per avere il tempo in secondi. Questo calcolo fu eseguito usando un registro a 24 bit fisso.

Il valore 1/10 ha una espansione binaria non finita

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

e fu troncato a 24 bit dopo il punto decimale

Ora il registro a 24 bit del patriot memorizzò

$$\tilde{x} = 0.00011001100110011001100$$

introducendo un errore assoluto di

$$Err = |x - \tilde{x}| = (0.00000000000000000000000110011001100)2$$

che equivale circa a $0.95 \cdot 10^{-7}$ in decimale.

La batteria dei Patriot era stata attiva circa 100 ore pari a

100×60×60×10= 3600000 decimi di secondo,

numero di decimi di secondo in un secondo

L'errore di $0.95 \cdot 10^{-7}$, dovuto alla memorizzazione di 1/10, (che è stato poi moltiplicato per il tempo risultante in decimi di secondi per riportarlo in secondi), produce un errore sul tempo risultante di circa 0.34 secondi:

$$0.000000095 \times 100 \times 60 \times 60 \times 10 = 0.342$$

Moltiplicando questo tempo per la velocità con cui viaggia uno Scud, e cioè circa 1676 metri al secondo, otteniamo:

$$\text{spazio} = \text{velocità} \times \text{tempo} = 1676 \text{ m/s} \times 0.34 \text{ s} = 569.8 \text{ m}$$

Un tempo di 0.34 secondi fu ovviamente più che sufficiente per mancare il bersaglio, in effetti uno Scud che viaggia a 1676 metri al secondo percorre in 0.34 secondi uno spazio di circa mezzo chilometro in avanti nella sua direzione, e questo spazio percorso fu sufficiente per uscire fuori dalla zona individuata dal Patriot.
(cfr http://www-users.math.umn.edu/~arnold/disasters/patriot.html)

Esplosione dell'Ariane V

Il 04.06.1996 un missile della classe Ariane V (un progetto costato, durante una decina d'anni di sviluppo, circa 7 bilioni di dollari) della European Space Agency (ESA) precipitó 37 secondi dopo il lancio dal poligono di Kourou nella Guiana francese.

La distruzione del missile e del suo carico provocó una perdita di circa 500 milioni di dollari.

Il guasto alla base del fallimento, accertato dopo due settimane di indagini, fu dovuto ad una erronea conversione di un numero floating-point a 64 bit (correlato alla velocitá orizzontale del razzo rispetto alla piattaforma) in un intero a 16 bit: il numero risultante, maggiore del massimo intero rappresentabile pari a 32768, provocó un overflow nella misura della velocitá orizzontale con il conseguente spegnimento dei razzi ed esplosione del missile.

(cfr http://www-users.math.umn.edu/~arnold/disasters/ariane.html)