# I numeri finiti e il calcolo numerico

## Calcolo Numerico

## a.a. 2021-22

Elena Loli Piccolomini

18 settembre 2021

# Outline

# Bibliografia


- Gladwell, Nagy, Ferguson, Introduction to scientific computing using Matlab cap. 2
- M. Heat, Scientific Computing: an introductiry survey, cap.1

# Errori nel calcolo numerico

Nella risoluzione di un problema numerico al calcolatore si possono avere diversi tipi di errore:

- Errore di misura, dovuto alle imperfezioni dello strumento di misura dei dati del problema.
- Errore di troncamento, quando un procedimento infinito viene realizzato come procedimento finito. (esempio: calcolo del valore di una funzione tramite sviluppo in serie).
- Errore algoritmico, dovuto al propagarsi degli errori di arrotondamento sulle singole operazioni in un procedimento complesso.
- Errore inerente, dovuto al fatto che i dati di un problema non sempre appartengono all'insieme $\mathbb{F}$ dei numeri rappresentabili e quindi vengono approssimati.

# Accuratezza di un risultato numerico

Misura dell'accuratezza:

- errore assoluto $E_a = |$ ris. approssimato - ris. esatto $|$
- errore relativo $E_r = \dfrac{\text{errore assoluto}}{|\text{risultato esatto}|}$ (ris. esatto $\neq 0$)
- errore percentuale $E_p = (E_r \times 100)\%$

# Accuratezza di un risultato numerico (cont.)

Sono molto importanti nella decisione se accettare o meno la soluzione di un problema numerico.
Fuori dal contesto, il valore dell'errore assoluto ha poco significato.

| | | | |
|---|---|---|---|
| $\alpha = 0.3\ 10^{1}$ | $\alpha^* = 0.31\ 10^{1}$ | $E_a = 0.1$ | $E_r = 0.3333..\ 10^{-1}$ <br> $E_p = 3.33..\%$ |
| $\alpha = 0.3\ 10^{-3}$ | $\alpha^* = 0.31\ 10^{-3}$ | $E_a = 0.1\ 10^{-4}$ | $E_r = 0.3333..\ 10^{-1}$ <br> $E_p = 3.33..\%$ |
| $\alpha = 0.3\ 10^{4}$ | $\alpha^* = 0.31\ 10^{4}$ | $E_a = 0.1\ 10^{3}$ | $E_r = 0.3333..\ 10^{-1}$ <br> $E_p = 3.33..\%$ |

# Rappresentazione dei numeri in memoria

- I numeri in memoria vengono rappresentati in forma binaria, cioè nella base 2.
- L'unità minima del linguaggio digitale è il BIT (Binary Digit, cifra binaria): spento (0), acceso (1)
- Le cifre utilizzabili nella rappresentazione di un numero in base 2 sono: 0,1.
- Un numero reale viene convertito in base 2.

# Rappresentazione in base $\beta$

Fissato un numero $\beta$ intero maggiore di 1, rappresentiamo nella base $\beta$ un numero reale qualunque $\alpha$.

- rappresentazione di tipo misto

$$\alpha = \pm(\alpha_0\alpha_1\alpha_2\ldots\alpha_p.\alpha_{p+1}\alpha_{p+1}\ldots)_\beta = \sum_{k=0}^{p}\alpha_k\beta^{p-k} + \sum_{k=1}^{\infty}\alpha_{k+p}\beta^{-k} \tag{1}$$

- rappresentazione normalizzata

$$\alpha = \pm(.\alpha_1\alpha_2\ldots)\beta^p$$

# Altre basi

- Base 8, cifre 0, 1, 2, 3, 4, 5, 6, 7

$$(0.36207)_8 = 3 \times 8^{-1} + 6 \times 8^{-2} + \ldots = (0.47286\ldots)_{10}$$

- Base 2, cifre 0, 1, o sui calcolatori "off" e "on" ("bit" = binary digit)

$$(0.111)_2 = 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = (0.875)_{10}$$

- La conversione di un numero in base $\beta$ alla forma decimale può essere ottenuta direttamente dall'espressione (1) operando in aritmetica decimale. La procedura è illustrata dal seguente esempio

$$(257)_8 = 2 \cdot 8^2 + 5 \cdot 8 + 7 = 175_{10}$$

- Consideriamo pertanto il passaggio dalla rappresentazione decimale alla rappresentazione in una generica base $\beta$

# I numeri naturali (cont.)

Conversione del numero naturale da base 10 a base 2 (sistema binario con il metodo delle divisioni successive per la conversione)

Esempio

$(34)_{10}=(100010)_2=0x2^0+1x2^1+0x2^2+0x2^3+0x2^4+1x2^5$

34:2=17 resto 0

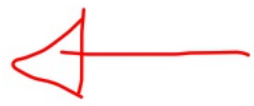
17:2=8 resto 1

8:2=4 resto 0

4:2=2 resto 0

2:2=1 resto 0

1:2=0 resto 1

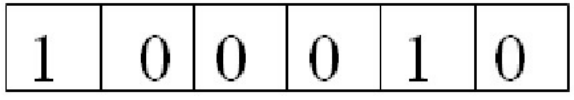


# I numeri naturali (cont.)

- Si ottengono le cifre di rappresentazione di $\alpha$ in base $\beta$ dalla meno significativa alla più significativa.
- Pertanto, scrivendo i resti delle divisioni nell'ordine inverso a quello in cui sono stati ottenuti si ottiene la rappresentazione in forma sintetica di $\alpha$ in base $\beta$.

# I numeri naturali (cont.)

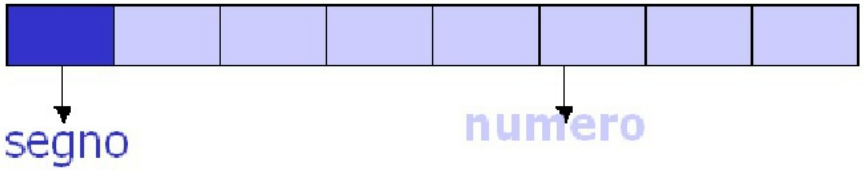
- Rappresentazione delle cifre 0 e 1 in una o più parole di memoria.




- Con $N$ bit sono rappresentabili i numeri naturali da 0 a $2^{N-1}$.

# Numeri interi

Supponiamo: base b=10, spazio di memoria riservato di lunghezza t+1=8




Esempi:

- X=+1320 è rappresentato come **00001320**
- X=+9999999 è rappresentato come **09999999**

Questo è il modo più semplice per rappresentare e distinguere numeri positivi e negativi: al numero binario vero e proprio viene anteposto un bit che, per convenzione, assume il valore 0 se il numero è positivo ed assume il valore 1 se il numero è negativo.

# Numeri interi negativi

- Conversione del valore assoluto del numero intero da base 10 a base 2.
- Rappresentazione in modulo e segno: il primo bit è utilizzato per il segno, gli altri per il modulo del numero.
  Con $N$ bit sono rappresentabili i numeri interi in $[-(2^{N-1}-1), 2^{N-1}-1)]$.
- Il numero 0 non ha rappresentazione unica: 00000000 e 10000000 significano infatti $+0$ e -0

# Numeri reali $<1$

$$(0.1)_{10} = (\quad\cdot\quad\cdot\quad)_2$$

- Si utilizza il metodo delle moltiplicazioni successive

base 2

$$0.1 \times 2 = 0.2 \quad \text{p. intera } 0$$
$$0.2 \times 2 = 0.4 \quad \text{p. intera } 0$$
$$0.4 \times 2 = 0.8 \quad \text{p. intera } 0$$
$$0.8 \times 2 = 1.6 \quad \text{p. intera } 1$$
$$0.6 \times 2 = 1.2 \quad \text{p. intera } 1$$
$$0.2 \times 2 = 0.4 \quad \text{p. intera } 0$$
$$\ldots$$

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

- Ci si arresta o perchè la parte frazionaria diventa nulla o perchè si è raggiunto un numero di cifre sufficienti.
- non è detto che se un numero ha rappresentazione finita in base 10 altrettanto accade in base $\beta$

# Numeri reali

Esempio: $\alpha = -25.357$

- Si converte la parte intera $25 = (11001)_2$
- Si converte la parte frazionaria $.357 = (.011)_2$
- $\alpha == (-11001.011)_2$

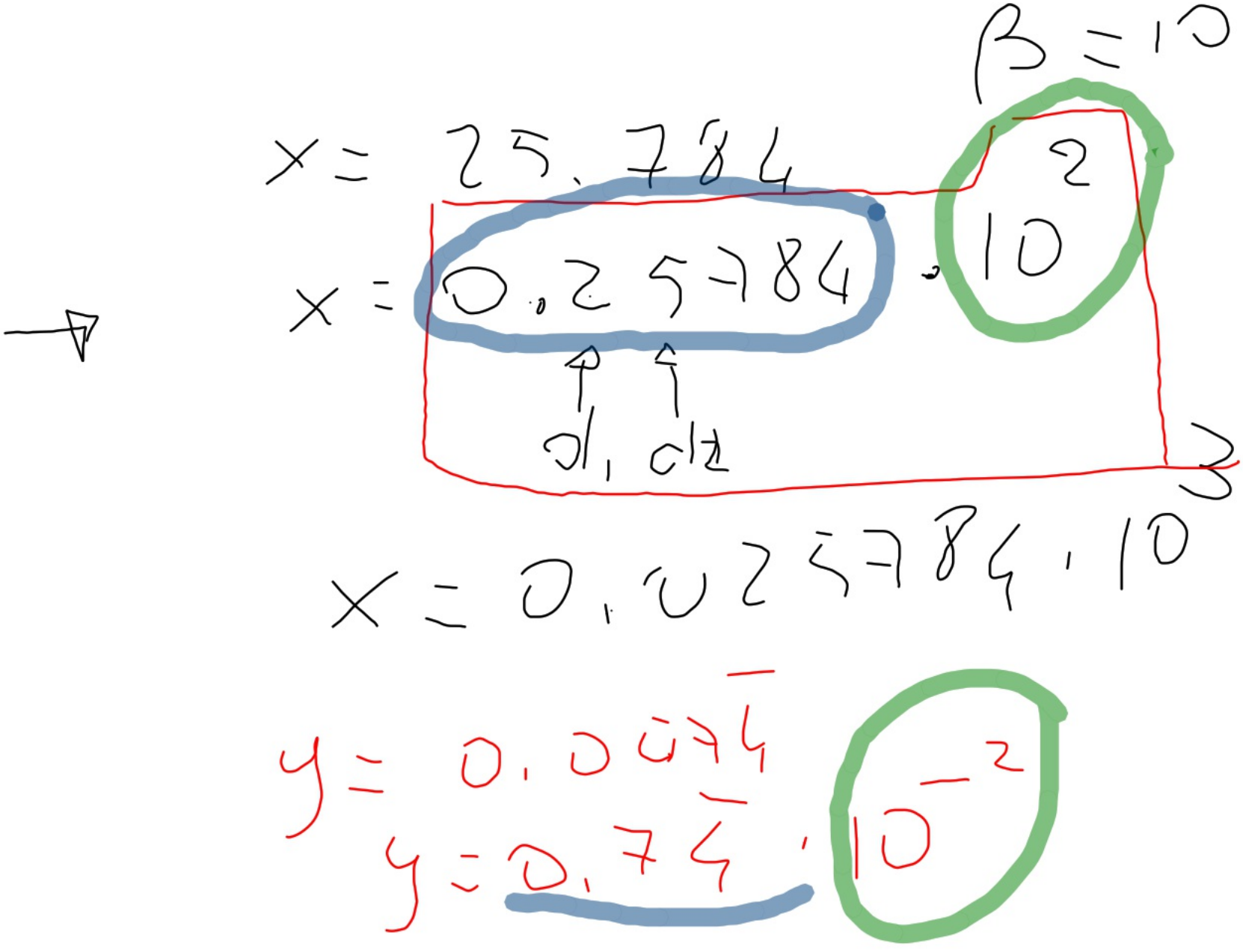
# I numeri reali

- Rappresentazione scientifica normalizzata di un numero reale: ogni numero reale $x \in \mathbb{R}$ può essere rappresentato come

$$x = \pm(0.d_1 d_2 d_3 \ldots)\beta^p = \pm \sum_{i=1}^{\infty} (d_i \beta^{-i})\beta^p \qquad (2)$$

dove

- $p$ è un numero intero
- le cifre $d_i$ verificano le condizioni:
  - i) $0 \leq d_i \leq \beta - 1$
  - ii) $d_1 \neq 0$ e le $d_i$ non sono tutte uguali a $\beta - 1$ a partire da un indice in poi.


β = 10
x = 25,784
x = 0,25784 · 10
2
0,25784
x = 0,0274 · 10⁻²
y = 0,0274
y = 2,74 · 10⁻²

# I numeri reali (cont.)

- La rappresentazione di $x$ nella forma $\pm(0.d_1 d_2 d_3 \ldots)\beta^p$ è chiamata rappresentazione normalizzata
- Il numero $m = 0.d_1 d_2 d_3 \ldots$ viene detto comunemente la mantissa di $x$
- $\beta^p$ la parte esponente
- Il numero $p$ è detto anche caratteristica (o esponente) di $x$
- $\beta$ è detto base
- Le $d_i$ sono dette cifre della rappresentazione

# Sistema floating point

- A causa della sua capacità finita, un calcolatore non è in grado di rappresentare tutto l'insieme dei numeri reali. Si pone pertanto il problema di definire, per ogni $x$ rappresentato nella forma 2, una sua rappresentazione approssimata nel calcolatore.
- Si tratta di un fatto tecnico, ma con importanti implicazioni nel calcolo numerico.
- Un metodo divenuto ormai usuale per il calcolo scientifico e noto come sistema floating-point o virgola mobile.
- Esso permette la rappresentazione di un ampio intervallo della retta reale con una distribuzione uniforme degli errori relativi. L'ampiezza effettiva dell'intervallo dipende dal particolare calcolatore su cui la procedura è implementata.

# Sistema floating point (cont.)

Si definisce insieme dei numeri macchina (floating-point) con $t$ cifre significative, base $\beta$ e range $(L, U)$, l'insieme dei numeri reali definito nel modo seguente

$$\mathbb{F}(\beta, t, L, U) = \{0\} \cup \left\{ x \in \mathbb{R} = \text{sign}(x)\beta^p \sum_{i=1}^{t} d_i \beta^{-i} \right\}$$

ove $t$,$\beta$ sono interi positivi con $\beta \geq 2$. Si ha inoltre

$$0 \leq d_i \leq \beta - 1, \quad i = 1, 2, \ldots$$

$$d_1 \neq 0, \quad L \leq p \leq U$$

Usualmente $U$ è positivo e $L$ negativo.

# Sistema floating point (cont.)

- In rappresentazione posizionale un numero macchina $x \neq 0$ viene denotato con $x = \pm .d_1 d_2 \ldots d_t \beta^p$
- La maggior parte dei calcolatori ha la possibilità di operare con lunghezze diverse di $t$, a cui corrispondono, ad esempio, la semplice e la doppia precisione.
- E' importante osservare che l'insieme $\mathbb{F}$ non è un insieme continuo e neppure infinito.

# Sistema floating point (cont.)

$F = -1, 0, 1, 2$

$d_1 \neq 0 \Rightarrow d_1 = 1$

I numeri dell'insieme $\mathbb{F}$ sono ugualmente spaziati tra le successive potenze di $\beta$, ma non su tutto l'intervallo.

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

$$\mathbb{F} = \{0\} \cup \{0.100 \times 2^p, 0.101 \times 2^p, 0.110 \times 2^p, 0.111 \times 2^p,\ p = -1, 0, 1, 2\}$$

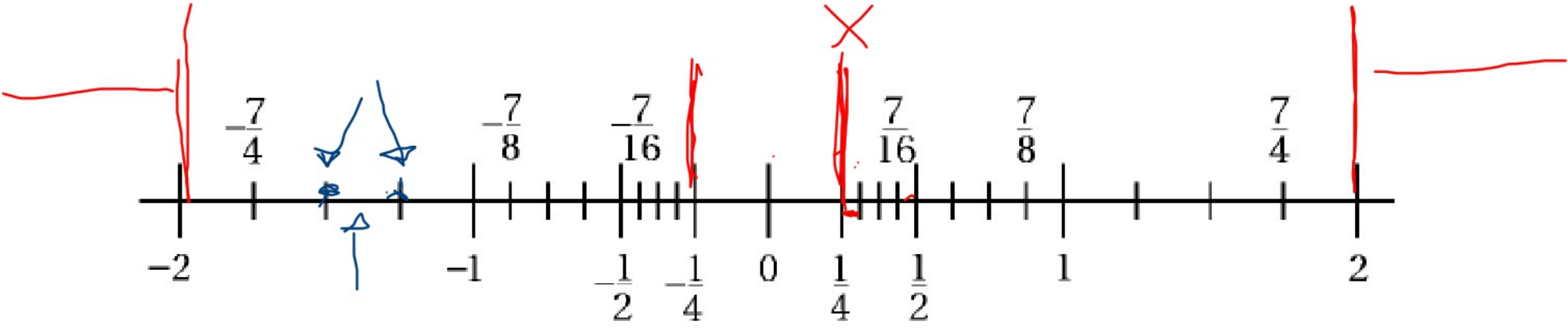
dove 0.100, 0.101, 0.110, 0.111 sono tutte le possibili mantisse e $p$ il valore dell'esponente.

# Sistema floating point (cont.)

Sono 33 numeri compreso lo zero

| | | | |
|---|---|---|---|
| $.100(-1)$ | $.101(-1)$ | $.110(-1)$ | $.111(-1)$ |
| *1/4* | *5/16* | *6/16* | *7/16* |
| $.100(0)$ | $.101(0)$ | $.110(0)$ | $.111(0)$ |
| *1/2* | *5/8* | *6/8* | *7/8* |
| $.100(1)$ | $.101(1)$ | $.110(1)$ | $.111(1)$ |
| *1* | *5/4* | *6/4* | *7/4* |
| $.100(2)$ | $.101(2)$ | $.110(2)$ | $.111(2)$ |
| *2* | *5/2* | *6/2* | *7/2* |

# Sistema floating point (cont.)




Risulta evidente l'aumento dell'ampiezza degli intervalli definiti dal valore $p$, in ognuno dei quali vengono localizzati i 4 valori della mantissa; ne risulta di conseguenza una diradazione delle suddivisioni verso gli estremi sinistro e destro della retta reale.

# Sistema floating point (cont.)

Le grandezze fondamentali che defiscono l'insieme dei numeri finiti, oltre alla base di rappresentazione $\beta$, sono:

- $L$ e $U$ che definiscono l'ordine di grandezza dei numeri rappresentabili e in genere $L \sim U$;
- il numero $t$ di cifre della mantissa che fissa la precisione di rappresentazione di ogni numero; infatti la retta reale viene suddivisa in intervalli $[\beta^p; \beta^{p+1}]$ di ampiezza crescente esponenzialmente con il valore $p$, ed in ognuno di questi intervalli vengono rappresentati lo stesso numero $(\beta - 1)\beta^{t-1}$ di valori. Si ha quindi una suddivisione molto densa per valori vicini allo 0 e più rada per valori grandi in valore assoluto.

# Sistema floating point (cont.)

Tutti i numeri reali all'interno di un sotto-intervallo tra due valori rappresentabili vengono approssimati da uno dei due estremi, quindi maggiore è il numero $t$ di cifre della mantissa, minore sarà l'ampiezza dei sotto-intervalli e quindi migliore l'approssimazione introdotta. La normalizzazione di numeri floating point causa un "salto" intorno allo 0 nell'intervallo dei reali.

# Sistema floating point (cont.)

- La più piccola variazione della mantissa è $u = \beta^{-t}$ (ULP=unit of last position).
- La distanza $\delta x$ fra due numeri di $\mathbb{F}$ in ogni intervallo $[\beta^p, \beta^{p+1}]$ è costante $\Delta x = u\beta^p$.
- Il passaggio da un esponente $p$ ad un esponente $p+1$ aumenta la quantità $\Delta x$ di un fattore $\beta$.
- La precisione del numero floating point dipende dalla lunghezza della mantissa

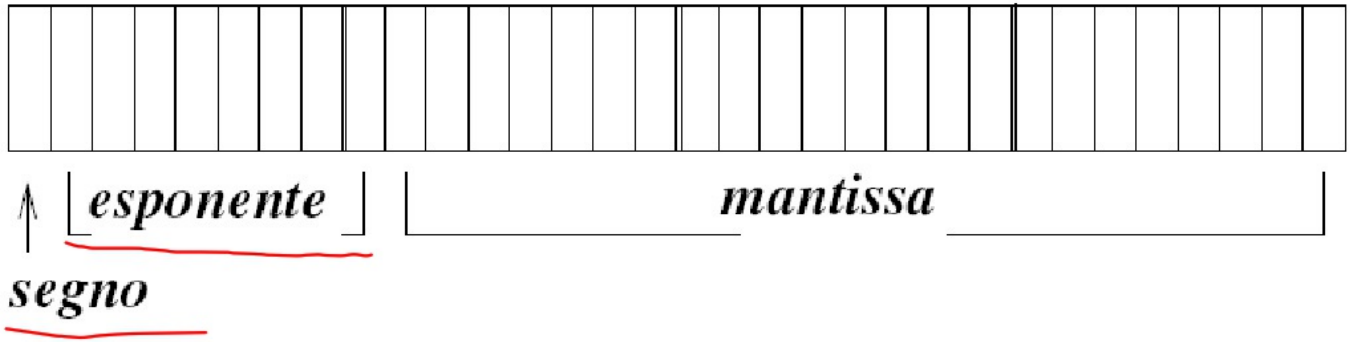
# Sistema floating point (cont.)

- Gli $n$ bit (o posizioni) disponibili per la memorizzazione di un numero finito vengono suddivisi tra le $t$ cifre della mantissa e l'esponente $p$ che può assumere $U - L + 1$ configurazioni diverse, più un bit per il segno del numero.
- Nel caso della rappresentazione binaria, sarà sempre $d_1 = 1$, per cui può essere sottointeso senza mai essere fisicamente rappresentato.

# Sistema floating point (cont.)

Alcune tipiche rappresentazioni sono:

- $\mathbb{F}_{(2,24,-128,127)}$ precisione singola: 32 bit. Vengono destinati 24 bit alla mantissa (in realtà solo 23) e 8 all'esponente ($2^8 = 256$ = U-L+1; con L=-128 e U=127)
- $\mathbb{F}_{(2,53,-1024,1023)}$ precisione doppia: 64 bit. Le cifre della mantissa sono 53 (rappresentati 52 bit) e dell'esponente 11 ($2^{11} = 2048$ =U-L+1; con L=-1024 e U=1023)




Rappresentazione binaria dei numeri finiti

# Sistema floating point (cont.)

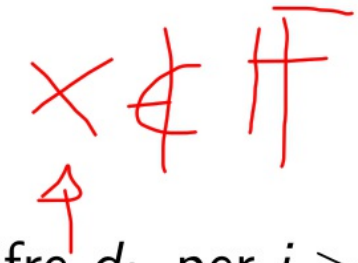
- Per la caratteristica si utilizza, in generale, la rappresentazione in traslazione, in modo che la configurazione nulla corrisponda all'esponente $L$.
- Si noti che la normalizzazione della mantissa permette di sfruttare al meglio le cifre disponibili, evitando di rappresentare esplicitamente gli 0 non significativi. Ad esempio 0.000124 viene rappresentato come $0.124 \times 10^{-3}$.
- La virgola quindi non ha una posizione fissa tra la parte intera e quella non, ma varia; per questo motivo tale rappresentazione è detta floating-point o a virgola mobile.

# Sistema floating point (cont.)

Come rappresentare un numero reale positivo $x$ in un sistema di numeri macchina $\mathbb{F}(\beta, t, L, U)$?

- Il numero $x$ è tale che $L \leq p \leq U$ e $d_i = 0$ per $i > t$; allora $x$ è un numero macchina ed è rappresentato esattamente.
- $p \notin [L, U]$; il numero non può essere rappresentato esattamente. Se $p < L$, si dice che si verifica un underflow; solitamente si assume come valore approssimato del numero $x$ il numero zero. Se $p > U$ si verifica un overflow e solitamente non si effettua nessuna approssimazione, ma il sistema di calcolo dà un avvertimento più drastico, come ad esempio, l'arresto del calcolo.

# Sistema floating point (cont.)



- La caratteristica $p \in [L, U]$, ma le cifre $d_i$, per $i > t$, non sono tutte nulle. In questo caso si pone il problema di scegliere un suo rappresentante in $\mathbb{F}$. Tale operazione viene indicata comunemente come operazione di arrotondamento (rounding), anche se in realtà possono essere utilizzate tecniche di tipo diverso.
  Se il numero è infinito, al numero viene associato il suo float ottenuto troncando (o approssimando) la mantissa e viene rappresentato in memoria il float ottenuto:

$$x = 1.333333\ldots = 0.133333\ldots \cdot 10^1 = fl(x) = 0.133333 \cdot 10^1$$

$$x = \pm 0.\ d_1 d_2 \cdots d_t \,|\, d_{t+1} \cdots \beta^{\beta}$$

$$fl(x) = 0.\ d_1 d_2 \cdots d_t \ \beta^{p}$$

troncamento

$$fl(x) = 0.\ d_1 d_2 \cdots d_{t-1} D_t \ \beta^{p}$$

arrotondamento

$$D_t = \begin{cases} d_t & \text{se } d_{t+1} \leq \dfrac{\beta}{2} \\ d_t + 1 & \text{se } d_{t+1} > \dfrac{\beta}{2} \end{cases}$$

$x = 0,123|45 \quad t = 3$

$\beta = 10$

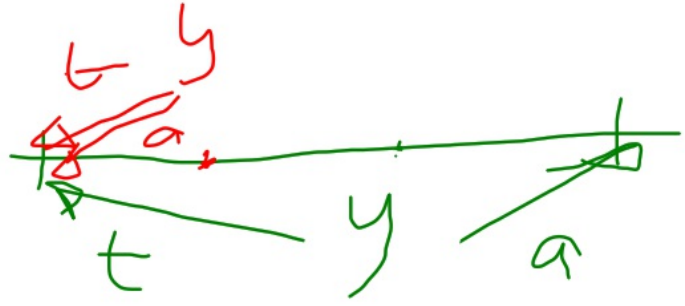
$fl(x) = 0,123 \leftarrow$

$\frac{\beta}{2} = 5$

$y = 0,178|93 \quad t = 3$

$fl(x) = 0,179 \leftarrow$

?



# Errori di rappresentazione

- Errore assoluto di arrotondamento

$$|fl(x) - x| < \beta^{p-t}$$

- Errore relativo di arrotondamento

$$\frac{|fl(x) - x|}{|x|} \leq \frac{1}{2}\beta^{1-t}$$

La quantità $eps = \frac{1}{2}\beta^{1-t}$ è detta precisione macchina nel fissato sistema floating point. La sua importanza numerica è data dalla seguente caratterizzazione: *eps* è il più piccolo numero macchina positivo tale che

$$fl(1 + eps) > 1$$

SINGLE (FLOAT32)

→ $\varepsilon \sim 10^{-7}$

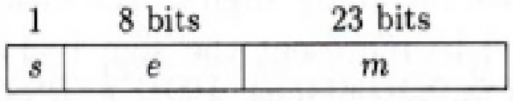
DOUBLE (FLOAT64)

→ $\varepsilon \sim 10^{-16}$

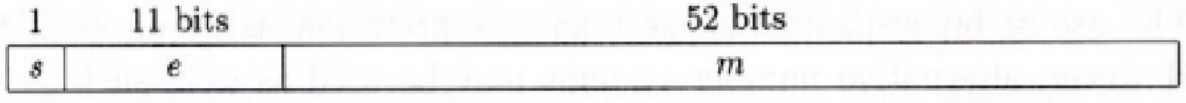
# Standard IEEE

- Importanza di definire uno standard della rappresentazione.
- IEEE computer society (Institute of Electrical and Electronic Engeneers) crea "IEEE standard for binary floating Arithmetic" (spesso riferito come IEEE 754) nel 1985.
- Rappresentazione dei formati IEEE standard in memoria

**Single Precision Format**: format width $N = 32$ bits



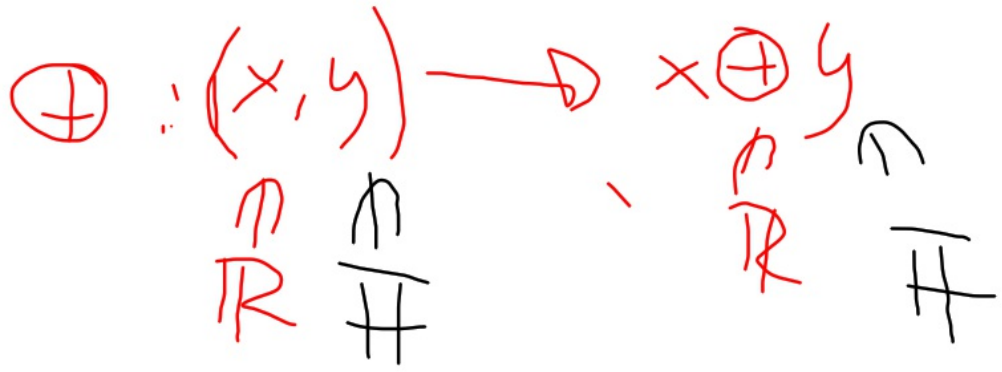

**Double Precision Format**: format width $N = 64$ bits



# Standard IEEE (cont.)

- Formato singola precisione $\mathbb{F}_{(2,24,-128,127)}$, 32 bit.
- Formato doppia precisione $\mathbb{F}_{(2,53,-1024,1023)}$, 64 bit.
- Nella codifica IEEE standard esistono anche sequenze di bit per rappresentare i valori $+0$, -0, $+\infty$, $-\infty$, e quantità simboliche chiamate NaN (Not a Number). Queste quantità sono utilizzate per gestire "speciali" operazioni, tipo la radice quadrata di un numero negativo, altrimenti non rappresentabili

# Aritmetica floating point



- Le operazioni eseguite sul calcolatore (calcolo numerico) possono essere eseguite solo con numeri rappresentabili sul calcolatore stesso.
- Poichè $\mathbb{F}$ è un sottoinsieme di $\mathbb{R}$, le usuali operazioni aritmetiche sono definite anche per operandi in $\mathbb{F}$, ma il loro risultato, in generale, non sta in $\mathbb{F}$.
- Il risultato esatto è arrotondato ad un numero che stia in $\mathbb{F}$.

# Aritmetica floating point (cont.)

- Operazione aritmetica reale $\cdot : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
- operazione floating-point (o di macchina): $\odot : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$

$$x \odot y = fl(x \cdot y)$$

- Ogni operazione provoca in generale un errore, detto errore di arrotondamento, molto piccolo

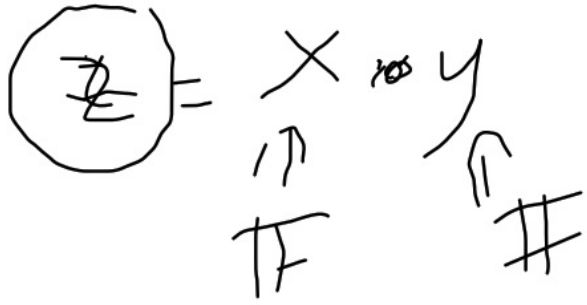
$$\left| \frac{(x \odot y) - (x \cdot y)}{x \cdot y} \right| < eps$$

# Aritmetica floating point (cont.)

```
>> u=29/13
u =
    2.23076923076923e+000
>> v=29-13*u
v =
0
>> u=29/1300
u =
    2.23076923076923e-002
>> v=29-1300*u
v =
    3.55271367880050le-015
```

Nel primo caso gli errori di arrotondamento si sono compensati nel secondo no

# Aritmetica floating point (cont.)



Realizzazione hardware di un'operazione floating point (con registro con precisione estesa) $x \oplus y$

- Eseguo l'operazione esatta $z = x + y$
- Effettuo il troncamento del risultato $x \oplus y = fl(z)$

Non valgono le proprietà commutativa e associativa nell'aritmetica floating point.

- Sulla maggior parte dei computer le operazioni aritmetiche (passo 1) sono fatte utilizzando più bit di quelli utilizzati per memorizzare il risultato.
- Il formato intermedio esteso non è accessibile all'utente, ma viene memorizzato in un registro interno alla CPU.
- Una volta che il calcolo è stato fatto in precisione estesa, il risultato viene arrotondato alla precisione del risultato (passo 2)

# Aritmetica floating point

Il risultato di unóperazione in aritmetica floating point puo' essere differente rispetto al risultato della stessa operazion e in aritmetica esatta.

- Addizione o sottrazione: lo spostamento della mantissa puo causare perdita di cifre o di accuratezza.
- Moltiplicazione: Il prodotto di due numeri con $t$-cifre di mantissa ha al piu' $2t$ cifre, quindi il risultato puo'non essere rappresentabile.
- Division :Il quoziente di due numeri con $t$cifre di mantissa puo'contenere piu'di $t$ cifre come per esempio la rappresentazione binaria di $1/10$

Example: $\beta = 10,\ t = 6\ x = 192.403, y = 0.635782,\quad fl(x) = 0.192403 \cdot 10^3,\ fl(y) = 0.635782 \cdot 10^0$

- $z = fl(x) + fl(y) = (0.192403 + 0.000635782) \cdot 10^3 = 0.193038782 \cdot 10^3$ $fl(z) = 0.193039 \cdot 10^3$ The last two digits of $y$ do not affect the result, and with even smaller exponent, $y$ could have had no effect on the result.
- ~~$w = fl(x) * fl(y) = (0.635782 \cdot 0.192403) \cdot 10^3 =$~~

# Aritmetica floating point

Il risultato in aritmetica reale puo'non essere rappfresentabile an ch perche'l'esponente $p$ e' esterno al suo range di rappfresentazione:

- $p > U$ Overflow
- $p < L$ Underflow

# Aritmetica floating point (cont.)

fl(x) = 0.715 · 10^15  fl(y) = 0,125 · 10^-4

Esempio: $x = 2.15 \times 10^{12}, y = 1.25 \times 10^{-5}$

$z = x - y$

Viene calcolata come:

$x = 2.15 \times 10^{12}$

$y = 0.00000000000000125 \times 10^{12}$

$x - y = 2.14999999999999999 \times 10^{12}$ → 0.21...

*arrotondato* a $2.15 \times 10^{12}$ = X

# Aritmetica floating point (cont.)

propagazione errori

$n = 1, n^{0}$

$10^{2}$

$10^{16}$

## Esempio di **errore catastrofico**

```
function epprox
% valuta:    e = exp(1) = lim_{n->infinity} (1 + 1/n)^n
e = exp(1);
fprintf('\n     n            f(n)                  error\n');
for n=logspace(0,16,9)
   f = (1+1/n)^n;
   fprintf('%9.1e  %14.10f  %14.10f\n',n,f,abs(f-e));
end
```

Calcolo del numero: $e = \lim_{n \to \infty} (1 + \frac{1}{n})^{n}$

$err(n) = |e - f(n)|$

# Aritmetica floating point (cont.)

```
>> epprox

      n              f(n)              error
 1.0e+000     2.0000000000     0.7182818285
 1.0e+002     2.7048138294     0.0134679990
 1.0e+004     2.7181459268     0.0001359016
 1.0e+006     2.7182804691     0.0000013594
 1.0e+008     2.7182817983     0.0000000301
 1.0e+010     2.7182820532     0.0000002248
 1.0e+012     2.7185234960     0.0002416676
 1.0e+014     2.7161100341     0.0021717944
 1.0e+016     1.0000000000     1.7182818285
```

$1 \cdot 10^{18}$ 1 ...

$10^{20}$ 1

# Aritmetica floating point (cont.)

Errori di arrotondamento:

1. Calcolo di $1/n$: errore di arrotondamento. Non provoca un errore grande rispetto al valore di $1/n$.
2. Calcolo di $1 + 1/n$: provoca errore di arrotondamento grande quando $n$ è grande. L'errore assoluto è piccolo rispetto a 1, ma grande rispetto a $1/n$. Per $n > 1016$, $1 + 1/n = 1$ in aritmetica doppia precisione.
3. $(1 + 1/n)^n$: amplifica l'errore commesso al punto 2

Il problema in questo caso è quello di sommare due numeri di grandezza molto diversa.

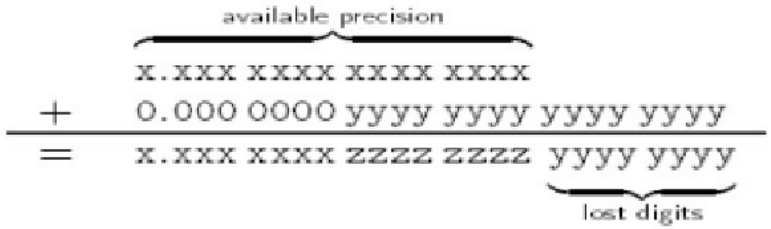
# Aritmetica floating point (cont.)

$$c = a + b, a >> b$$
$$c = a - b, a << b (a < 0)$$

**Provocano grandi errori di arrotondamento**

**Esempio:** $c = a + b, a = x.xxxx\ldots \times 10^{0}, b = y.yyy\ldots \times 10^{-8}$

La somma nell'aritmetica floating point viene fatta
Utilizzando una precisione estesa per la mantissa
e riportando i due numeri allo stesso esponente
(il maggiore):




Le cifre più significative di $a$ si mantengono, quelle di $b$ no.

# Conclusioni

- Tutta l'informazione viene codificata in binario, cioè con una sequenza finita di 0 e 1.
- I dati numerici non sempre sono esattamente rappresentabili in una simile codifica e questo provoca delle approssimazioni e degli errori di cui si deve tenere conto nel calcolo numerico. Tali errori di solito non sono esattamente quantificabili.