1 Rappresentazione interna dei numeri

MATLAB usa lo standard IEEE in doppia precisione. Il numero di macchina

$$x = \pm\,(1.b_{-1}\dots b_{-52})\cdot 2^p$$

è rappresentato in una parola di memoria di 64 bit come segue.

- 1 bit per il segno + o −: 0 per + e 1 per −.
- 11 bit per la rappresentazione in base 2 di

  $$q = 1023 + p.$$

  Essendo $p \in \{-1022, -1021, \dots, 1022, 1023\}$, si ha $q \in \{1, 2, \dots, 2045, 2046\}$ e i numeri interi tra 0 e $2047 = 2^{11} - 1$ sono rappresentabili in base 2 utilizzando 11 bit. Osservare che le sequenze di undici bit $(00\dots00)_2 = 0$ e $(11\dots11)_2 = 2047$ non sono utilizzate.
- 52 bit per le cifre binarie $b_{-1}, \dots, b_{-52}$.

Per cui, se s è il bit di segno e $q = (q_{10}q_9\dots q_0)_2$ è la rappresentazione in base 2 di q, x è rappresentato dalla parola di memoria

$$\underbrace{s}_{1\text{ bit}}\underbrace{|q_{10}q_9\dots q_1q_0|}_{11\text{ bit}}\underbrace{b_{-1}b_{-2}\dots b_{-51}b_{-52}|}_{52\text{ bit}}.$$

Scriviamo la rappresentazione interna come parola di memoria di 1. Poichè

$$1 = (1.0)_2\cdot 2^0,$$

si ha

$$q = 1023 = 2^{10} - 1 = (\underbrace{01\dots11}_{11\text{ bit}})_2$$

e quindi la rappresentazione interna di 1 è

$$0|01\dots11|00\dots00|$$

Scriviamo la rappresentazione interna di -30.25. Poichè

$$-30.25 = -\,(11110.01)_2 = -\,(1.111001)_2\cdot 2^4$$

si ha

$$q = 1027 = 2^{10} + 3 = (\underbrace{100\dots0011}_{11\text{ bit}})_2$$

e la rappresentazione interna di -30.25 è

$$1|100\dots0011|11100100\dots00|.$$

La rappresentazione interna si vede con il formato di visualizzazione hex. Vengono mostrate 16 cifre esadecimali che quando espanse in 4 bit forniscono i 64 bit della rappresentazione del numero.

$>>$ *help format*
$>>$ *format hex*
$>>$ 1
$>> -30.25$
$>>$ *format*

2 Numeri di macchina

Il numero di macchina più grande è

$$(1.11\ldots11)_2 \cdot 2^{1023}$$

Esso è il valore della costante *realmax*.

$>>$ *help realmax*
$>>$ *realmax*
$>>$ *format hex*
$>>$ *realmax*
$>>$ *format*

La rappresentazione di *realmax* ha bit di segno 0,

$$q = 2046 = 2047 - 1 = (\underbrace{11\ldots110}_{11 \text{ bit}})_2$$

e bit della mantissa tutti uguali a 1:

$$0|11\ldots110|11\ldots11|$$

Un esempio di overflow è

$>> 2 * realmax$
$>>$ *format hex*
$>> 2 * realmax$
$>>$ *format*

La rappresentazione di $2 * realmax$ è la rappresentazione nello standard IEEE di $+\infty$. Si ha bit di segno 0, nella parte dell'esponente tutti bit 1 e nella parte della mantissa tutti bit 0:

$$0|11\ldots11|00\ldots00|.$$

Si osservi che la configurazione di undici bit 1 nella parte dell'esponente è una delle due che non sono utilizzate nella rappresentazione dei numeri normalizzati.

$+\infty$ è il valore della costante *inf*.

```
>> help inf
>> inf
>> 1/0
>> 2 + inf
>> inf + inf
>> 2 * inf
>> -3 * inf
>> format hex
>> inf
>> -inf
>> format
```

Il numero di macchina più piccolo è

$$(1.00\ldots0)_2 \cdot 2^{-1022}.$$

Esso è il valore della costante *realmin*.

```
>> help realmin
>> realmin
>> format hex
>> realmin
>> format
```

La rappresentazione di *realmin* ha bit di segno 0,

$$q = 1 = (\underbrace{00\ldots01}_{11 \text{ bit}})_2$$

e bit della mantissa tutti uguali a 0:

$$0|00\ldots001|00\ldots00|.$$

Esempi di underflow sono i seguenti.

```
>> realmin/2
>> realmin/2^2
>> realmin/2^3
```

Si osservi che i numeri non sono posti uguale a zero. Ma si ha

```
>> realmin/2^52
>> realmin/2^53
```

I numeri

$$\frac{realmin}{2^k},\ k \in \{1, 2, \ldots, 52\},$$

sono in rappresentazione "denormalizzata"

$$\frac{realmin}{2^k} = (0.\underbrace{0\ldots0\overbrace{1}^{k\text{-esima posizione}}0\ldots0}_{52\text{ bit}})_2 \cdot 2^{-1022} = (1.\underbrace{00\ldots00}_{52-k\text{ bit}})_2 \cdot 2^{-1022-k}.$$

I numeri "denormalizzati" di ordine di grandezza $2^{-1022-k}$ utilizzano una mantissa ridotta con $t = 52-k$ cifre binarie dopo il punto. La precisione di macchina è ridotta a $2^{-t} = 2^k eps$.

Si osservi i 52 bit della mantissa di

```
>> format hex
>> realmin
>> realmin/2
>> realmin/2^2
>> realmin/2^3
>> realmin/2^4
>> realmin/2^5
>> realmin/2^52
>> realmin/2^53
>> format
```

Solo *realmin*/2^53 viene posto uguale a 0. Tutti i numeri denormalizzati hanno una configurazione di undici bit 0 nella parte dell'esponente, che è l'altra non utilizzata nella rappresentazione dei numeri normalizzati.

La rappresentazione nello standard IEEE di 0 è la seguente: bit di segno 0, nella parte dell'esponente tutti bit 0 e nella parte della mantissa tutti bit 0:

$$0|00\ldots00|00\ldots00|$$

Un altro numero di macchina speciale è Not a Number, il valore della costante

nan, e rappresenta una quantità indeterminata.

$$>> help\ nan$$
$$>> nan$$
$$>> 0/0$$
$$>> inf - inf$$
$$>> inf/inf$$
$$>> 2 + nan$$
$$>> -3 * nan$$
$$>> format\ hex$$
$$>> nan$$
$$>> format$$

La rappresentazione nello standard IEEE di Not a Number è la seguente: bit di segno 1, nella parte dell'esponente tutti bit 1 e nella parte della mantissa tutti bit 0 eccetto il primo:

$$1|11\ldots11|100\ldots00|$$

Di nuovo, come con $+\infty$, si ha la configurazione di undici bit 1 nella parte dell'esponente che non è usata dai numeri normalizzati. Si noti il bit 1 di inizio mantissa che serve a distinguere Not a Number da $-\infty$.

La precisione di macchina è il valore della costante *eps*.

$$>> help\ eps$$
$$>> eps, 2\hat{}(-52)$$

$1 + eps$ è il numero di macchina successivo a 1.

$$>> format\ long, 1, 1 + eps$$

Nella visualizzazione decimale non si apprezza la differenza tra 1 e $1 + eps$. La differenza si vede nel formato *hex* della rappresentazione interna.

$$>> format\ hex,\ 1, 1 + eps$$

Si osservi che MATLAB non tronca, ma arrotonda. Questo si può vedere considerando il numero $1 + 3/4 * eps$ che è approssimato da 1 nel troncamento e da $1 + eps$ nell'arrotondamento.

$$>>> format\ hex,\ 1 + 3/4 * eps, 1 + 1/4 * eps,\ format$$

3 Propagazione degli errori di arrotondamento

Confrontiamo i due algoritmi per calcolare

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

Nelle istruzioni MATLAB sottostanti x è un numero di macchina, cosicché l'errore ε_y nel calcolo di $f(x)$ coincide con l'errore algoritmico, e x è grande, cosicché l'Algoritmo 1 è instabile su tale dato x.

Inoltre, denotiamo con y_1 il risultato dell'Algorimo 1 e con y_2 quello dell'Algoritmo 2. Essendo l'Algoritmo 2 stabile, y_2 è considerato il valore corretto. Si approssima l'errore algoritmico con

$$\varepsilon_{\text{alg}} = \frac{y_1 - y_2}{y_2}.$$

Poichè gli indici di stabilità dell'Algoritmo 1 sono circa x, il rapporto

$$\frac{\varepsilon_{\text{alg}}}{\text{eps}}$$

dovrebbe essere dell'ordine di x.

$>>$ *format long e*
$>>$ $x = 10\hat{}3;$
$>>$ $y1 = sqrt(x+1) - sqrt(x), y2 = 1/(sqrt(x+1) + sqrt(x))$
$>>$ $epsilon = (y1 - y2)/y2$
$>>$ $epsilon/eps$
$>>$ $x = 10\hat{}6;$
$>>$ $y1 = sqrt(x+1) - sqrt(x), y2 = 1/(sqrt(x+1) + sqrt(x))$
$>>$ $epsilon = (y1 - y2)/y2$
$>>$ $epsilon/eps$
$>>$ $x = 10\hat{}9;$
$>>$ $y1 = sqrt(x+1) - sqrt(x), y2 = 1/(sqrt(x+1) + sqrt(x))$
$>>$ $epsilon = (y1 - y2)/y2$
$>>$ $epsilon/eps$
$>>$ $x = 10\hat{}12;$
$>>$ $y1 = sqrt(x+1) - sqrt(x), y2 = 1/(sqrt(x+1) + sqrt(x))$
$>>$ $epsilon = (y1 - y2)/y2$
$>>$ $epsilon/eps$
$>>$ $x = 10\hat{}15;$
$>>$ $y1 = sqrt(x+1) - sqrt(x), y2 = 1/(sqrt(x+1) + sqrt(x))$
$>>$ $epsilon = (y1 - y2)/y2$
$>>$ $epsilon/eps$

Confrontiamo ora i due algoritmi per calcolare

$$f(x) = x_1 x_2 + x_1 = x_1 (x_1 + x_2).$$

Nelle istruzioni MATLAB sottostanti x_1 e x_2 sono numeri di macchina, cosicché l'errore ε_y nel calcolo di $f(x)$ coincide con l'errore algoritmico, e x_2 è vicino a -1, cosicché l'Algoritmo 1 è instabile sul dato (x_1, x_2).

Per l'Algoritmo 1 instabile si ha

$$\varepsilon_{\text{alg}} \doteq \frac{x_2}{x_2 + 1}\gamma_1 + \gamma_2$$

e quindi per avere un errore algoritmico con ordine di grandezza superiore a eps è necessario che $\gamma_1 \neq 0$, dove γ_1 è l'errore dell'operazione $x_1 x_2$. Quindi $x_1 x_2$ non deve essere un numero di macchina. Scegliamo allora

$$x_1 = 1 + 2^{-a} \text{ e } x_2 = -\left(1 + 2^{-b}\right)$$

in modo che $a + b = 53$. Allora

$$x_1 x_2 = -\left(1 + 2^{-a} + 2^{-b} + 2^{-53}\right)$$

non è un numero di macchina.

Come sopra, denotiamo con y_1 è il risultato dell'Algorimo 1 e con y_2 quello dell'Algoritmo 2. Essendo l'Algoritmo 2 stabile, y_2 è considerato il valore corretto. Di nuovo come sopra, si approssima l'errore algoritmico con

$$\varepsilon_{\text{alg}} = \frac{y_1 - y_2}{y_2}.$$

Poichè l'indice di stabilità grande dell'Algoritmo 1 è $\frac{x_2}{x_2+1}$, il rapporto

$$\frac{\varepsilon_{\text{alg}}}{\text{eps}}$$

dovrebbe essere dell'ordine di $\frac{x_2}{x_2+1}$.

$$
\begin{aligned}
&>> x1 = 1 + 2\hat{}(-30)\\
&>> x2 = -(1 + 2\hat{}(-23));\\
&>> y1 = x1 * x2 + x1, y2 = x1 * (x2 + 1)\\
&>> epsilon = (y1 - y2)/y2\\
&>> epsilon/eps, x2/(x2 + 1)\\
&>> x1 = 1 + 2\hat{}(-20)\\
&>> x2 = -(1 + 2\hat{}(-33));\\
&>> y1 = x1 * x2 + x1, y2 = x1 * (x2 + 1)\\
&>> epsilon = (y1 - y2)/y2\\
&>> epsilon/eps, x2/(x2 + 1)\\
&>> x1 = 1 + 2\hat{}(-10);\\
&>> x2 = -(1 + 2\hat{}(-43));\\
&>> y1 = x1 * x2 + x1, y2 = x1 * (x2 + 1)\\
&>> epsilon = (y1 - y2)/y2\\
&>> epsilon/eps, x2/(x2 + 1)\\
&>> x1 = 1 + 2\hat{}(-1);\\
&>> x2 = -(1 + 2\hat{}(-52));\\
&>> y1 = x1 * x2 + x1, y2 = x1 * (x2 + 1)\\
&>> epsilon = (y1 - y2)/y2\\
&>> epsilon/eps, x2/(x2 + 1)
\end{aligned}
$$