Relazione esercitazione sugli errori

Andres Coronado MAT 2761046

11 giugno 2019

1 Considerazioni: Il tipo double

In c++ il tipo double è a 64 bit suddivisi come in tabella:

segno	esponente	mantissa	totale
1	11	52	64

Il sistema di memorizzazione dei numeri in virgola mobile nello standard IEEE 754 utilizzato da C++ per rappresentare i numeri float, double, long double ed in generale i numeri in virgola mobile si usano tre campi: segno s(1 bit, 2 possibili valori),mantissa M(52 bit) ed esponente e(11 bit 2048 possibili valori), mediante la relazione :$(-1)^s \cdot 2^E \cdot M$. Dove E rappresenta l'esponente polarizzato con un bias k ($E = \#e - k$) pari a 1023, ed M la mantissa normalizzata(siccome la prima cifra significativa della mantissa normalizzata in base 2 è sempre 1, si guadagna un bit!).Un numero macchina di tipo Double si può quindi pensare come la funzione

$$\mathfrak{F}(2, 52, 1024, 1023) = \{x \in \mathbb{R} : \pm f 2^p, -1024 \leq p \leq 1023, f = \sum_{i=1}^{52} d_i 2^{-i}\}$$

Dove $B = 2$ è la Base, $t = 52$ è il numero di cifre di cui è composta la mantissa, $-m = -1024$ è l'esponente minimo, sotto al quale si ha un Underflow ed $M = 1023$ è invece l'upper bound al di sopra del quale si andrebbe in Overflow.

L'errore relativo assoluto (precisione macchina) dato un numero macchina è minorato da

$$u = \frac{1}{2} \cdot 2^{-52+1} = 2^{-52} \approx 2.2 \cdot 10^{-16}$$
$$|\varepsilon| \leq u$$

Quindi relativamente all'esercitazione posso già intuire che considerando la somma di un numero α ed un numero β piccolo "abbastanza" cioè più piccolo di circa 16 ordini di grandezza rispetto ad α ($\beta < \alpha \cdot 2.2 \cdot 10^{-16}$) la somma così ottenuta non risente del valore β. In particolare nell'esercitazione sono presenti somme tra un numero dell'ordine di grandezza 10^i ($i \in [0, 6]$) a numeri dell'ordine di 10^{20}. Osservo quindi da quanto considerato che finchè i non è abbastanza grande (circa 4) questo tipo di somma non viene "sentita" e viene quindi troncata/approssimata.

2 Risultati

l'algoritmo fornito(ed implementato in C++) si può schematizzare come segue:
$alg0 : a0 := d0 + 1$ $ap0 := 10^i$ $b0 := d1 + 1$ $ap1 := 10^i$ $a := a0 * ap0$ $b := b0 * ap1$ $c := (-1) * b$ $f0 := a + b$ $f1 := b + c$ $ya := f0 + c$ $yb := a + f1$

il seguente output conferma quanto discusso, infatti quando i arriva a 4 la somma $f0 = a + b$ vale $5.00000000000000007e + 20$ e quindi y_a comincia ad avvicinarsi a y_b al crescere di i.

```
double ranges in [1.7976931348623157e+308,1.7976931348623157e+308]
________________________________________________________________
               i = 0
________________________________________________________________
 a0 := d0 + 1 = 7
ap0 := a0^i = 1
```

```
 b0  :=  d1  +  1  =  5
ap1  :=  10^20  =  1e+20
  a  :=  a0*ap0  =  7
  b  :=  b0*ap1  =  5e+20
  c  :=  (-1)*b  =  -5e+20
 f0  :=  a+b  =  5e+20  <---  TRONCATURA,  somma  non  sentita
 f1  :=  b+c  =  0
 ya  :=  f0+c  =  (a+b)+c  =  0  <---  c  =  -(a+b)
 yb  :=  a+f1  =  a+(b+c)  =  7
```

```
TEST:  [  (a+b)+c  !=  a+(b+c)  Approssimazione  /  troncatura]  -  |ya-yb|  =   7
```

```
                 i  =  1
```

```
 a0  :=  d0  +  1  =  7
ap0  :=  a0^i  =  10
 b0  :=  d1  +  1  =  5
ap1  :=  10^20  =  1e+20
  a  :=  a0*ap0  =  70
  b  :=  b0*ap1  =  5e+20
  c  :=  (-1)*b  =  -5e+20
 f0  :=  a+b  =  5e+20  <---  TRONCATURA,  somma  non  sentita
 f1  :=  b+c  =  0
 ya  :=  f0+c  =  (a+b)+c  =  0<---  c  =  -(a+b)
 yb  :=  a+f1  =  a+(b+c)  =  70
```

```
TEST:  [  (a+b)+c  !=  a+(b+c)  Approssimazione  /  troncatura]  -  |ya-yb|  =   70
```

```
                 i  =  2
```

```
 a0  :=  d0  +  1  =  7
ap0  :=  a0^i  =  100
 b0  :=  d1  +  1  =  5
ap1  :=  10^20  =  1e+20
  a  :=  a0*ap0  =  700
  b  :=  b0*ap1  =  5e+20
  c  :=  (-1)*b  =  -5e+20
 f0  :=  a+b  =  5e+20  <---  TRONCATURA,  somma  non  sentita
 f1  :=  b+c  =  0
 ya  :=  f0+c  =  (a+b)+c  =  0<---  c  =  -(a+b)
 yb  :=  a+f1  =  a+(b+c)  =  700
```

```
TEST:  [  (a+b)+c  !=  a+(b+c)  Approssimazione  /  troncatura]  -  |ya-yb|  =   700
```

```
                 i  =  3
```

```
 a0  :=  d0  +  1  =  7
ap0  :=  a0^i  =  1000
 b0  :=  d1  +  1  =  5
ap1  :=  10^20  =  1e+20
  a  :=  a0*ap0  =  7000
  b  :=  b0*ap1  =  5e+20
  c  :=  (-1)*b  =  -5e+20
 f0  :=  a+b  =  5e+20  <---  TRONCATURA,  somma  non  sentita
 f1  :=  b+c  =  0
 ya  :=  f0+c  =  (a+b)+c  =  0<---  c  =  -(a+b)
 yb  :=  a+f1  =  a+(b+c)  =  7000
```

```
TEST: [ (a+b)+c != a+(b+c) Approssimazione / troncatura] - |ya-yb| = 7000
---------------------------------------------------------------------------
---------------------------------------------------------------------------
              i = 4
---------------------------------------------------------------------------
 a0 := d0 + 1 = 7
ap0 := a0^i = 10000
 b0 := d1 + 1 = 5
ap1 := 10^20 = 1e+20
  a := a0*ap0 = 70000
  b := b0*ap1 = 5e+20
  c := (-1)*b = -5e+20
 f0 := a+b = 5.0000000000000007e+20 <--- SOMMA "SENTITA"
 approssimata al primo numero macchina disponibile
 f1 := b+c = 0
 ya := f0+c = (a+b)+c = 65536 <--- |f0-c|>10^16 <--- Approssimato
 yb := a+f1 = a+(b+c) = 70000
---------------------------------------------------------------------------
TEST: [ (a+b)+c != a+(b+c) Approssimazione / troncatura] - |ya-yb| = 4464
---------------------------------------------------------------------------
---------------------------------------------------------------------------
              i = 5
---------------------------------------------------------------------------
 a0 := d0 + 1 = 7
ap0 := a0^i = 100000
 b0 := d1 + 1 = 5
ap1 := 10^20 = 1e+20
  a := a0*ap0 = 700000
  b := b0*ap1 = 5e+20
  c := (-1)*b = -5e+20
 f0 := a+b = 5.0000000000000072e+20 <--- SOMMA "SENTITA"
 approssimata al primo numero macchina disponibile
 f1 := b+c =
 ya := f0+c = (a+b)+c = 720896 <--- |f0-c|>10^16 <--- Approssimato
 yb := a+f1 = a+(b+c) = 700000
---------------------------------------------------------------------------
TEST: [ (a+b)+c != a+(b+c) Approssimazione / troncatura] - |ya-yb| = 20896
---------------------------------------------------------------------------
---------------------------------------------------------------------------
              i = 6
---------------------------------------------------------------------------
 a0 := d0 + 1 = 7
ap0 := a0^i = 1000000
 b0 := d1 + 1 = 5
ap1 := 10^20 = 1e+20
  a := a0*ap0 = 7000000
  b := b0*ap1 = 5e+20
  c := (-1)*b = -5e+20
 f0 := a+b = 5.0000000000000701e+20 <--- Approssimato
 f1 := b+c = 0
 ya := f0+c = (a+b)+c = 7012352 <--- |f0-c|>10^16 <--  Approssimato
 yb := a+f1 = a+(b+c) = 7000000
---------------------------------------------------------------------------
TEST: [ (a+b)+c != a+(b+c) Approssimazione / troncatura] - |ya-yb| = 12352
---------------------------------------------------------------------------
```

3 Conclusione

Si dimostra come la rappresentazione dei numeri macchina possa alterare risultati di somme di numeri distanti tra loro.