B-017

無誤差演算のための最小演算精度の提案と評価

Proposal and Evaluation of a Method for Minimum Precision of Numerical Computation with Error Free

井川 尚幸 † 古賀 雅伸 †
Naoyuki Ikawa Masanobu Koga

1. はじめに

一般に数値計算では倍精度の浮動小数点型が用いられるが，数学的に厳密な式が与えられても，丸め誤差などの影響で精度の高い結果を得られるとは限らない．この問題を解決するために、多倍長演算 [1] や精度保証付き数値計算 [2] などが提案されている。Java では，任意精度を扱うクラス BigDecimal[3] が標準実装されているが，必ずしも最小の演算精度で計算を行っているわけではなく，無駄が含まれていることがある．本研究では多倍長演算において，四則演算結果を無誤差で得るために必要な最小演算精度を導出する手法を提案し，その手法を BigDecimal で適用させて評価する．

2. 浮動小数点演算

2.1 浮動小数点数のフォーマット

B 進数 p_a 桁の浮動小数点数 a は

$$\begin{aligned} a &= (-1)^{s_a} \times \left(\frac{d_1}{B} + \cdots + \frac{d_{p_a}}{B^{p_a}} \right) \times B^{e_a} \quad (1) \\ &= (-1)^{s_a} \left(0.d_1 \cdots d_{p_a}\right)_B \times B^{e_a} \\ d_i &\in \{0, 1, \cdots, B-1\} \end{aligned}$$

と表現される．ここで，s_a を a の符号，e_a を a の指数，p_a を a の桁数，$(0.d_1 d_2 \cdots \quad d_p)_B$ を仮数，B を基数と言う．今後，断らない限りすべての浮動小数点数の演算精度は等しく p とする．

浮動小数点演算規格である IEEE754 で定められた倍精度浮動小数点数は 1 ビットの符号部，11 ビットの指数部，52 ビットの仮数部を持っており，演算精度は 10 進数で約 16 桁である．多倍長精度浮動小数点数は，1 ビットの符号部を持ち，指数部と仮数部を任意に指定することで，任意の演算精度を実現する．ここで，仮数部の長さを長くすることによってより高精度な計算が可能となる．

2.2 無誤差変換

浮動小数点数 a, b に対して，Knuth のアルゴリズム [4] と Dekker のアルゴリズム [5] を用いることにより，無誤差変換を行うことが出来る．

†九州工業大学

2.2.1 加減算

加算の無誤差変換を行うための Knuth のアルゴリズムを以下に示す．

function$[x,\ y]$=TwoSum$(a,\ b)$
$x = \mathrm{fl}(a+b)$
$c = \mathrm{fl}(x-a)$
$y = \mathrm{fl}((a-(x-c))+(b-c))$

ここで，$\mathrm{fl}(a+b)$ は，$a+b$ の浮動小数点演算の最近点丸めを表す．上述の TwoSum を用いると

$$a + b = x + y \quad (e_y \leq e_x - p)$$

のように $a+b$ の近似値が x に，丸め誤差が y にそれぞれ浮動小数点数として格納される．別の方法として下述の FastTwoSum 用いると，$|a| > |b|$ という条件において，x と y をより早く求めることができる．

function$[x,\ y]$=FastTwoSum$(a,\ b)$
$x = \mathrm{fl}(a+b)$
$z = \mathrm{fl}(x-a)$
$y = \mathrm{fl}(b-z)$

減算については b の符号を変えて上述のアルゴリズムを用いることにより行える．

2.2.2 乗算

次に乗算の無誤差変換を行うための Dekker のアルゴリズムを以下に示す．

function$[a_H,\ a_L]$=Split(a)
$c = \mathrm{fl}((B^{\frac{1}{2}p}+1)\cdot a)$
$a_H = \mathrm{fl}(c-(c-a))$
$a_L = \mathrm{fl}(a-a_H)$

上述の Split を用いると

$$a = a_H + a_L$$

のように，a_H に a の仮数部上位半分の $\frac{1}{2}p$ 桁，a_L に下位半分の $\frac{1}{2}p$ 桁に対応するものが誤差なしで正確に格納される．

function[x, y]=TwoProdut(a, b)
$x = \mathrm{fl}(a \times b)$
$[a_H, a_L]$=Split(a)
$[b_H, b_L]$=Split(b)
$y = \mathrm{fl}(a_L \times b_L - (((x - a_H \times b_H) - a_L \times b_H) - a_H \times b_L))$

上述の TwoProduct を用いると

$$a \times b = x + y \quad (e_y \leq e_x - p)$$

のように，$x \times y$ の近似値が x に，丸め誤差が y にそれぞれ浮動小数点として格納される．

3. 無誤差演算に必要な最小演算精度

第 2 章で紹介した無誤差変換をそのまま適用させると，1 度の計算を行うごとに計算結果を表現するには，計算前の演算精度の 2 倍の演算精度を必要としてしまう．すなわち，無誤差演算を繰り返すことは，演算精度が無限大に発散するため難しい．そこで，無誤差演算に必要な最小演算精度 p^* を与える定理を以下に示す．

定理 1 無誤差変換で得られた x と y が

$$\begin{aligned} x &= (-1)^{s_x} \times \left(\frac{d_1}{B} + \cdots + \frac{d_{p_x}}{B^{p_x}} \right) \times B^{e_x} \\ y &= (-1)^{s_y} \times \left(\frac{d_1}{B} + \cdots + \frac{d_{p_y}}{B^{p_y}} \right) \times B^{e_y} \end{aligned}$$

と表す．無誤差演算に必要な最小演算精度は次式で与えられる．

$$p^* = \begin{cases} \tilde{p}_x & (y = 0) \\ \tilde{p}_y + (e_x - e_y) & (y \neq 0) \end{cases}$$

ただし，e_x，e_y は x，y の指数，$\tilde{p}_x$，$\tilde{p}_y$ は x，y の仮数部において，最上位ビットから 0 でないビットまでの最大ビット長を示す．

定理 1 の証明を以下に示す．$y = 0$ の場合は明らかであるため，$y \neq 0$ の場合について証明する．

証明 1

$$\begin{aligned} & x + y \\ =\ & (-1)^{s_x} \times B^{e_x} \sum_{i=1}^{p_x} \frac{d_{x_i}}{B^i} + (-1)^{s_y} \times B^{e_y} \sum_{i=1}^{p_y} \frac{d_{y_i}}{B^i} \\ =\ & (-1)^{s_x} \times B^{e_x} \sum_{i=1}^{\tilde{p}_x} \frac{d_{x_i}}{B^i} + (-1)^{s_y} \times B^{e_y} \sum_{i=1}^{\tilde{p}_y} \frac{d_{y_i}}{B^i} \\ =\ & B^{e_x} \left((-1)^{s_x} \sum_{i=1}^{\tilde{p}_x} \frac{d_{x_i}}{B^i} + (-1)^{s_y} \times B^{-e_x+e_y} \sum_{i=1}^{\tilde{p}_y} \frac{d_{y_i}}{B^i} \right) \\ =\ & B^{e_x} \left((-1)^{s_x} \sum_{i=1}^{\tilde{p}_x} \frac{d_{x_i}}{B^i} + (-1)^{s_y} \sum_{i=e_x-e_y}^{\tilde{p}_y+e_x-e_y} \frac{d_{y_{i-e_x-e_y}}}{B^i} \right) \\ =\ & (-1)^{s_x} \times \left(\sum_{i=1}^{\tilde{p}_y+e_x-e_y} \frac{d_i}{B^i} \right) \times B^{e_x} \end{aligned}$$

となり，*(1)* 式の形式で表されることが分かる．したがって無誤差演算に必要な *(*結果を表現するために必要な*)* 最小演算精度は $p^* = \tilde{p}_y + (e_x - e_y)$ である．

□

4. 無誤差演算パッケージ

Java で実装されている任意精度の浮動小数を扱うクラス BigDecimal 利用して最小演算精度で無誤差演算を行うクラス BigDecimalEFFloat を作成した．

4.1 BigDecimal

任意精度の浮動小数を扱うクラス BigDecimal では，算術演算子 (+,*,/など) と同等の機能を持つメソッド (add, multiply, divide など) が実装されている．BigDecimal は，任意精度の「スケールなしの整数値」と，32 ビット整数の「スケール」で表現される．ここで，スケールとは小数点以下の桁数を表す．つまり，BigDecimal で表される数値は (unscaledValue $\times 10^{-scale}$) となる．そして精度は，オブジェクトを生成するときに指定することが出来る．指定しなければ，初期値として与えた数字の精度となる．また，BigDecimal クラスには，スケールの値を返すメソッド scale() と使用される演算精度を返すメソッド precision() がそれぞれ用意されている．

4.2 演算時間

BigDecimalEFFloat では，第 3 章で示したように，無誤差演算を行うための最小演算精度を求め，その演算精度で計算する．つまり，BigDecimalEFFloat では，常に最小の演算精度で値が保持されていることになり，演算に必要な時間が少なくなることが期待される．しかし，最小の演算精度を求めるために，複数回の演算が必要となり，演算時間が必要となる．

4.3 除算

浮動小数点同士の除算については，その結果を浮動小数点数で表せないことがあるため，分子と分母を多倍長浮動小数点数とする有理数で表す．

5. 評価

5.1 最小演算精度

以下の Rump の例題 [6] を使用して，p^* を求める方法の評価を行う．

$$f = (333.75 - a^2)b^6 + a^2(11a^2b^2 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b}$$

$$\text{for} \quad a = 77617, b = 33096$$
$$\text{correct value} = -\frac{54767}{66192} \approx -0.827396\cdots$$

図 1 に 5 の評価 (右辺の計算) における各四則演算の結果を表現するための仮数部の桁数 (p^*) の変化を示す．ただし，横軸は四則演算を行う順番を表す．つまり，横軸の 1 は $(333.75 - a^2)$ を行った後の精度，2 は $(11 \times a^2)$ を行った後の精度,15 は最終的な演算精度を表す．なお，この測定は以下の表 1 の環境で行った．

表 1: 測定環境

OS	windows 8.1
メモリ	16GB
CPU	CORE i7-3770

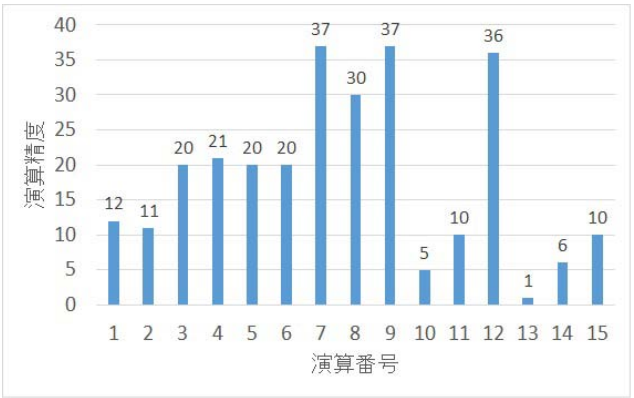

図 1: 演算精度の推移

図 1 から，Rump の例題は無誤差で求めるには，10 進数で 37 桁の精度が必要であることが分かる．

5.2 精度比較

以下に BigDecimal と本論文で提案した定理を用いた BigDecimalEFFloat の精度を比較をするための例題を示す．

$$f = \{(a + b) * (c - d)\}^5$$

for $\quad$ a = 154321, b = 95679, c = 6.54321, d = 2.54321

1 : a, b, c, d の演算精度の合計
2 : $a + b$ を行った後
3 : $c - d$ を行った後
4 : $(a + b) \times (c - d)$ を行った後
5 : 5 乗を行った後

この f の真値は 1.0×10^{30} である．図 2 は横軸が演算の推移，縦軸が必要な精度の合計である．

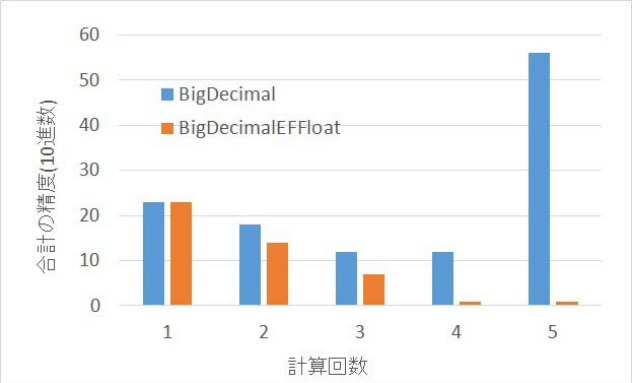

図 2: 演算精度の比較

図 2 から，BigDecimalEFFloat は BigDecimal に比べて少ない精度で計算することが出来ることが分かる．

5.3 演算時間比較

5.3.1 演算精度に差がある場合

5.2 節で使った f を用いて速度比較を行う．以下 a_n を求める時間を評価する．

$$a_{n+1} = a_n \times a_n \qquad (n \geq 1)$$
$$a_0 = f$$

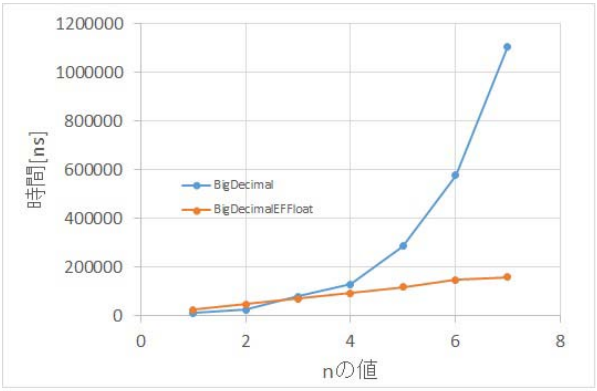

図 3: 演算時間

n の値を増やしていくごとに必要な精度の差が大きくなり，演算時間にも差が出るようになることが図 3 により分かる．

5.3.2 演算精度に差がある場合

以下に BigDecimal と BigDecimalEFFloat で精度が等しい場合の必要な演算時間を比較する．図 4 では，加算時の演算時間を示す．

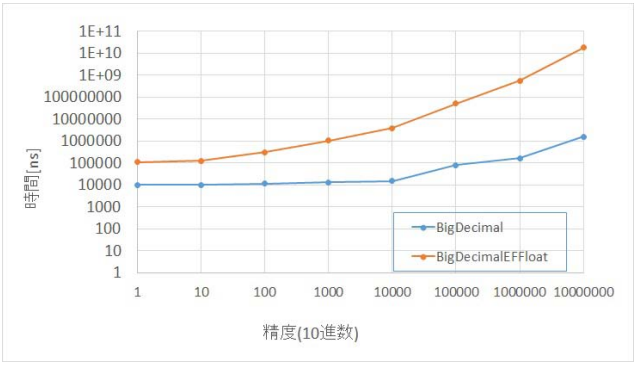

図 4: 加算時の演算時間

図 5 に乗算時における演算時間を示す．

これらから，同じ精度における演算速度は BigDecimal の方が BigDecimalEFFloat よりも早いことが分かる．

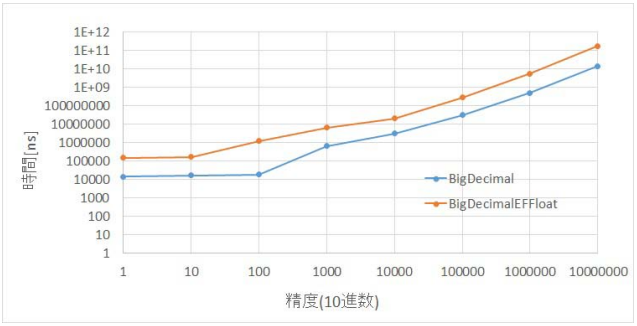

図 5: 乗算時における演算時間

6. おわりに

本研究では多倍長演算において，四則演算を無誤差で行うための最小演算精度を導く方法を提案し，それを例題を用いて評価を行った．現時点では，演算に使う精度の差がかなり多い場合には早く演算できるようになっている．今後は演算に必要な精度が分かっているという利点をさらに生かし，より速く演算が出来るようにしたい．

・謝辞

本研究は JSPS 科研費 15K06147 の助成を受けたものである．

参考文献

[1] 山村英介, 古賀雅伸, 矢野健太郎, 山田賢治. 多倍長計算を用いた制御系設計パッケージ. 第 52 回システム制御情報学会研究発表講演会, 2008.

[2] 大石進一. 精度保証付き数値計算. コロナ社, 2000. ISBN4-339-02605-0.

[3] Bigdecimal web pages. `http://docs.oracle.com/javase/jp/6/api/java/math/BigDecimal.html`.

[4] D.E.Knuth. The Art of Computer Programming（日本語版）. アスキー, 2004.

[5] T.J.Dekker. *A Floating-point Technique for Extending the Available Precision.* Numer. Math, 1971.

[6] Ramon E. Moore. Reliability in computing : the role of interval methods in scientific computing. *Academic Press*, 1988.