桁落ち誤差と情報落ち誤差

この資料では、Maple 上で行う数値計算時に見落としがちな、数値演算誤差について述べています。Maple では任意精度（桁数）の計算が可能ですが、多項式計算、積分、微分方程式などの基本数値演算でも起こり得る誤差、特に桁落ちと情報落ちの誤差について原因と対策を述べています。また、Maple で利用可能な高精度（桁数）計算のメリットについても述べています。

Maple での高精度計算

Maple では、Digits と呼ばれる Maple システムの環境変数の値を用いて数値計算時の精度（桁数）をコントロールしています。標準では、

```
> restart;
> Digits;
```

$$10 \tag{1.1}$$

と表示されるように、10桁の精度です。
Digits 環境変数の値を変えることで、数値計算時の近似値の桁数は変わります。例えば、以下は evalf コマンドを用いて$\sqrt{2}$の値の近似値を求めています。

```
> evalf(sqrt(2));
```

$$1.414213562 \tag{1.2}$$

Digits の環境変数を変えると、その値で設定された桁数で近似値を計算します。ここでは、32桁に設定してみます。

```
> Digits := 32;
```

$$Digits := 32 \tag{1.3}$$

```
> evalf(sqrt(2));
```

$$1.4142135623730950488016887242097 \tag{1.4}$$

精度をコントロールする Digits 環境変数を上手に使うことで、数値計算時の誤差の発生を回避することが可能です。次節以降で例題を使って桁落ち・情報落ちなどの誤差をさらに詳しく見てみます。

桁落ち誤差と情報落ち誤差

桁落ち誤差とは、値の等しい数値同士の減算を行った際に、数値の持つ有効桁数が減少してしまう現象を言います。例えば、

```
> restart;
> 1.23456 - 1.23444;
```

$$0.00012 \tag{2.1}$$

という計算では、0 以外の有効数字は 1, 2 の2つだけです。この計算結果の Maple の内部表現を見てみると、

```
> op((2.1));
```

$$12, -5 \tag{2.2}$$

となっています。1番目の戻り値は10進整数でのデータ、2番目の戻り値が 基数 10 のベキ乗を表しています。
一方、情報落ちとは、絶対値の大きい数と小さい数の加減算を行った際に、小さい数の情報が失われてしまう現象を言います。いま、Maple のデフォルトの Digits の精度（Digits = 10）で以下の計算を行っています。

```
> 1.23456*10^5 + 0.00001;
```

$$123456.0000 \tag{2.3}$$

すると、0.00001 という情報は計算結果からは失われています。
桁落ちと情報落ちが頻繁に起きる現象を見るために、絶対値がほぼ等しい２つの浮動小数を作る次の２つの関数 $m(k), n(k)$ を定義します。ここで、k は正数を取ります。

```
> restart;
> m := k -> sqrt(10^k+1);
```

$$m := k \mapsto \sqrt{10^k + 1} \tag{2.4}$$

```
> n := k -> sqrt(10^k-1);
```

$$n := k \mapsto \sqrt{10^k - 1} \tag{2.5}$$

$m(k), n(k)$ をリストにして、$k = 3\sim50$ までの奇数を動いたときのデータを変数 $vals$ に定義します。

```
> vals := seq([m(k),n(k)],k=3..50,2);
```

$$vals := \left[\sqrt{1001}, 3\sqrt{111}\right], \left[\sqrt{100001}, 3\sqrt{11111}\right], \left[\sqrt{10000001}, 3\sqrt{1111111}\right], \tag{2.6}$$
$$\left[\sqrt{1000000001}, 9\sqrt{12345679}\right], \left[11\sqrt{826446281}, 3\sqrt{11111111111}\right],$$
$$\left[\sqrt{10000000000001}, 3\sqrt{1111111111111}\right], \left[\sqrt{100000000000001},$$
$$3\sqrt{111111111111111}\right], \left[\sqrt{10000000000000001}, 3\sqrt{11111111111111111}\right],$$
$$\left[\sqrt{1000000000000000001}, 3\sqrt{1111111111111111111}\right],$$
$$\left[7\sqrt{20408163265306122449}, 3\sqrt{11111111111111111111}\right],$$
$$\left[\sqrt{100000000000000000001}, 3\sqrt{111111111111111111111}\right],$$
$$\left[\sqrt{1000000000000000000001}, 3\sqrt{1111111111111111111111}\right],$$
$$\left[\sqrt{10000000000000000000001}, 9\sqrt{123456790123456790123456789}\right],$$
$$\left[\sqrt{100000000000000000000001}, 3\sqrt{11111111111111111111111111}\right],$$
$$\left[\sqrt{1000000000000000000000001},\right.$$
$$\left.3\sqrt{111111111111111111111111111}\right],$$
$$\left[11\sqrt{826446280991735537190082644628}1,\right.$$
$$\left.3\sqrt{1111111111111111111111111111}\right],$$
$$\left[\sqrt{10000000000000000000000000001},\right.$$
$$\left.3\sqrt{11111111111111111111111111111}\right],$$
$$\left[\sqrt{100000000000000000000000000001},\right.$$
$$\left.3\sqrt{111111111111111111111111111111}\right],$$
$$\left[13\sqrt{59171597633136094674556213017751479}29,\right.$$
$$\left.3\sqrt{1111111111111111111111111111111}\right],$$
$$\left[\sqrt{1000000000000000000000000000001},\right.$$
$$\left.3\sqrt{11111111111111111111111111111111}\right],$$
$$\left[\sqrt{10000000000000000000000000000001},\right.$$
$$\left.3\sqrt{111111111111111111111111111111111}\right],$$
$$\left[\sqrt{100000000000000000000000000000001},\right.$$
$$\left.9\sqrt{123456790123456790123456790123456789}\right],$$
$$\left[\sqrt{1000000000000000000000000000000001},\right.$$
$$\left.3\sqrt{1111111111111111111111111111111111}\right],$$
$$\left[\sqrt{10000000000000000000000000000000001},\right.$$
$$\left.3\sqrt{11111111111111111111111111111111111}\right]$$

これを数値化（浮動小数化）すると、次のようなデータになります。

```
> evalf(vals);
```

$$[31.63858404, 31.60696125], [316.2293472, 316.2261849], [3162.277818, 3162.277503], [31622.77662, 31622.77659], [316227.7660, 316227.7659], [3.162277660\ 10^{6}, 3.162277659\ 10^{6}], [3.162277660\ 10^{7}, 3.162277659\ 10^{7}], [3.162277660\ 10^{8}, 3.162277659\ 10^{8}], [3.162277660\ 10^{9}, 3.162277659\ 10^{9}], [3.162277660\ 10^{10}, 3.162277659\ 10^{10}], [3.162277660\ 10^{11}, 3.162277659\ 10^{11}], [3.162277660\ 10^{12}, 3.162277659\ 10^{12}], [3.162277660\ 10^{13}, 3.162277660\ 10^{13}], [3.162277660\ 10^{14}, 3.162277659\ 10^{14}], [3.162277660\ 10^{15}, 3.162277659\ 10^{15}], [3.162277660\ 10^{16}, 3.162277659\ 10^{16}], [3.162277660\ 10^{17}, 3.162277659\ 10^{17}], [3.162277660\ 10^{18}, 3.162277659\ 10^{18}], [3.162277660\ 10^{19}, 3.162277659\ 10^{19}], [3.162277660\ 10^{20}, 3.162277659\ 10^{20}], [3.162277660\ 10^{21}, 3.162277659\ 10^{21}], [3.162277660\ 10^{22}, 3.162277660\ 10^{22}], [3.162277660\ 10^{23}, 3.162277659\ 10^{23}], [3.162277660\ 10^{24}, 3.162277659\ 10^{24}] \quad \textbf{(2.7)}$$

k の値が大きくなるにつれてリストの２つの浮動小数の有効数字が等しくなっていっているのがわかります。各リストの浮動小数 $[m(k_i), n(k_i)]$, $i = 1..n$ に対して、

$\sum_{i=1}^{n} (m(k_i) - n(k_i))$ を計算してみましょう。

```
> valsf := evalf(vals):
  add(v[1]-v[2],v=valsf);
```

$$1.101111111\ 10^{15} \quad \textbf{(2.8)}$$

上記の計算はデフォルトの精度である Digits = 10 で計算しています。一方、精度を 16 桁にして計算してみると、式 **(2.8)** の結果とはまったく違う値が出てきます。

```
> valsf := evalf[16](vals):
  add(v[1]-v[2],v=valsf);
```

$$0.03513108 \quad \textbf{(2.9)}$$

ここで起きた現象は、絶対値の等しい２つの数同士の計算による誤差（桁落ち）と、絶対値が大きく異なる数同士の加算による誤差（情報落ち）が連続して発生しているものです。桁落ち、情報落ちの各誤差が発生すると、**(2.8)** のようにまったく意味のない結果が得られることがあり得ます。

桁落ち、情報落ちの誤差は、計算順序によっても結果が大きく異なることに注意しなければなりません。いま、Digits = 16 として vals の値 $m(k_i)$, $-n(k_i)$ をランダムに加算してみます。*vals* のデータに対して、$m(k_i)$, $-n(k_i)$ の式列 *avals* を準備します。

```
> Digits := 16:
  avals := map(p->[evalf(p[1]),evalf(-p[2])][],[vals])[];
```

$$avals := 31.63858403911275, -31.60696125855822, 316.2293471517152, -316.2261848740551, 3162.277818282258, -3162.277502054493, 31622.77661749518, -31622.77658587241, 316227.7660184191, -316227.7660152567, 3.162277660168537\ 10^{6}, -3.162277660168221\ 10^{6}, 3.162277660168381\ 10^{7}, -3.162277660168377\ 10^{7}, 3.162277660168379\ 10^{8}, -3.162277660168380\ 10^{8}, 3.162277660168379\ 10^{9}, -3.162277660168380\ 10^{9}, 3.162277660168379\ 10^{10}, -3.162277660168380\ 10^{10}, 3.162277660168379\ 10^{11}, -3.162277660168380\ 10^{11}, 3.162277660168379\ 10^{12}, -3.162277660168380\ 10^{12}, 3.162277660168379\ 10^{13}, -3.162277660168380\ 10^{13}, 3.162277660168379\ 10^{14}, \quad \textbf{(2.10)}$$

$-3.162277660168380\ 10^{14}$, $3.162277660168379\ 10^{15}$, $-3.162277660168380\ 10^{15}$, $3.162277660168380\ 10^{16}$, $-3.162277660168380\ 10^{16}$, $3.162277660168379\ 10^{17}$, $-3.162277660168380\ 10^{17}$, $3.162277660168379\ 10^{18}$, $-3.162277660168380\ 10^{18}$, $3.162277660168379\ 10^{19}$, $-3.162277660168380\ 10^{19}$, $3.162277660168379\ 10^{20}$, $-3.162277660168380\ 10^{20}$, $3.162277660168379\ 10^{21}$, $-3.162277660168380\ 10^{21}$, $3.162277660168379\ 10^{22}$, $-3.162277660168380\ 10^{22}$, $3.162277660168379\ 10^{23}$, $-3.162277660168380\ 10^{23}$, $3.162277660168379\ 10^{24}$, $-3.162277660168380\ 10^{24}$

式列の要素 $m(k_i)$, $-n(k_i)$ をランダムに加算するための順序を作ります。

```
> id := combinat[randperm](nops([avals]));
```

$$id := [30, 2, 48, 36, 11, 40, 9, 44, 47, 15, 38, 8, 1, 19, 18, 42, 14, 4, 32, 39, 46, 3, 26, 28, 20, 27, 35, 24, 23, 10, 6, 45, 33, 29, 41, 13, 7, 12, 37, 25, 5, 16, 22, 21, 17, 31, 34, 43] \quad (2.11)$$

生成したランダムな順序で総和を計算してみます。

```
> add(avals[i],i=id);
```

$$-1.64\ 10^{9} \quad (2.12)$$

別の順序を作って計算してみます。

```
> id := combinat[randperm](nops([avals]));
```

$$id := [4, 1, 6, 7, 42, 43, 17, 34, 19, 37, 45, 20, 24, 16, 13, 10, 26, 2, 23, 35, 8, 41, 33, 39, 21, 38, 48, 32, 14, 9, 44, 22, 12, 47, 18, 29, 5, 31, 46, 11, 3, 36, 15, 40, 30, 25, 28, 27] \quad (2.13)$$

```
> add(avals[i],i=id);
```

$$-1.4705666963\ 10^{9} \quad (2.14)$$

再度、別の順序を作って計算してみます。

```
> id := combinat[randperm](nops([avals]));
```

$$id := [37, 38, 13, 44, 7, 29, 23, 42, 12, 18, 39, 5, 16, 19, 27, 22, 28, 6, 26, 11, 32, 14, 24, 46, 41, 21, 20, 10, 40, 25, 1, 47, 30, 43, 31, 8, 48, 15, 17, 34, 36, 2, 35, 9, 45, 33, 3, 4] \quad (2.15)$$

```
> add(avals[i],i=id);
```

$$-5.831620999968377\ 10^{8} \quad (2.16)$$

いずれも同じデータによる加算で総和を計算しているにも関わらず、結果が異なる上に有効桁数も大きく違ってきます。このように、桁落ち・情報落ちの誤差は浮動小数の計算結果を大きく誤らせてしまう現象なのです。

桁落ち・情報落ち誤差への対処方法

桁落ち・情報落ち誤差は、1) 精度を上げることか、または 2) 計算順序を変える方法で回避できる場合があります。

精度変更によるアプローチ

1) 計算精度を変えて総和計算した場合：8, 16, 32, 64 の各桁数で計算してみます。

```
> valsf := evalf[8](vals):
  add(v[1]-v[2],v=[valsf]);
```

$$2.212222222212212\ 10^{17} \quad \textbf{(3.1.1)}$$

```
> valsf := evalf[16](vals):
  add(v[1]-v[2],v=[valsf]);
```

$$-1.111111101075975\ 10^{9} \quad \textbf{(3.1.2)}$$

```
> valsf := evalf[32](vals):
  add(v[1]-v[2],v=[valsf]);
```

$$0.03513641705073 \quad \textbf{(3.1.3)}$$

```
> valsf := evalf[64](vals):
  add(v[1]-v[2],v=[valsf]);
```

$$0.03513641705073 \quad \textbf{(3.1.4)}$$

今回の計算例では、正しい計算結果を得るためには16桁よりも大きい精度が必要になっていることがわかります。Maple の可変精度システムを用いることで、桁落ち・情報落ちの誤差にも対処可能な計算が可能です。

計算方法を変えたアプローチ

2) 計算過程を変更して計算した場合：桁落ちを防いで減算を計算する別のプログラムを定義して総和を計算してみます。この計算方法は、$a-b$ の直接的な減算を回避して、$\frac{(a-b)\cdot(a+b)}{(a+b)}$ により減算を計算しています。

```
> subtract := proc(a,b)
    local tmp;

    tmp := a+b;
    tmp := tmp*a - tmp*b;
    tmp := tmp/(a+b);
    return(tmp);

  end proc;
```

$$subtract := \mathbf{proc}(a, b) \quad \textbf{(3.2.1)}$$

$$\mathbf{local}\ tmp;$$

$$tmp := a + b;\ tmp := tmp * a - tmp * b;\ tmp := tmp / (a + b);\ \mathbf{return}\ tmp$$

$$\mathbf{end\ proc}$$

定義した subtract 関数をリストの各要素に対して適用し、減算を計算してみます。なお、精度（桁数）も変えて subtract による減算を計算することで、計算結果の違いも確認してみます。

10桁精度：

```
> valsf := evalf[10](vals):
  map(v->subtract(v[1],v[2]),[valsf]);
```

$$[0.03162278999999748, 0.003162299999905400, 0.0003150000006142724, \quad \textbf{(3.2.2)}$$
$$0.00003000000638305746, 0.00009999991589922561, 0.00009999991589922561,$$
$$0.00999991589922561, 0.09999991589922561, 0.9999991589922561,$$

9.999991589922561, 99.99991589922561, 999.9991589922561, 0.,
99999.91589922561, 999999.1589922561, $9.999991589922561\ 10^{6}$,
$9.999991589922561\ 10^{7}$, $9.999991589922561\ 10^{8}$, $9.999991589922561\ 10^{9}$,
$9.999991589922561\ 10^{10}$, $9.999991589922561\ 10^{11}$, 0.,
$9.999991589922561\ 10^{13}$, $9.999991589922561\ 10^{14}$]

16桁精度：

```
> valsf := evalf[16](vals):
  map(v->subtract(v[1],v[2]),[valsf]);
```

[0.03162278055453260, 0.003162277660049793, 0.0003162277660168383, **(3.2.3)**
0.00003162277660168379, $3.162435774051388\ 10^{-6}$, $3.162277660168380\ 10^{-7}$,
$4.743416490252569\ 10^{-8}$, $-1.581138830084190\ 10^{-7}$,
$-1.581138830084190\ 10^{-6}$, −0.00001581138830084190,
−0.0001581138830084190, −0.001581138830084190,
−0.01581138830084190, −0.1581138830084190, −1.581138830084190, 0.,
−158.1138830084190, −1581.138830084190, −15811.38830084190,
−158113.8830084190, $-1.581138830084190\ 10^{6}$, $-1.581138830084190\ 10^{7}$,
$-1.581138830084190\ 10^{8}$, $-1.581138830084190\ 10^{9}$]

24桁精度：

```
> valsf := evalf[24](vals):
  map(v->subtract(v[1],v[2]),[valsf]);
```

[0.03162278055453260, 0.003162277660207908, 0.0003162277660168384, **(3.2.4)**
0.00003162277660168379, $3.162277660168379\ 10^{-6}$, $3.162277660168380\ 10^{-7}$,
$3.162277660168379\ 10^{-8}$, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]

32桁精度：

```
> valsf := evalf[32](vals):
  map(v->subtract(v[1],v[2]),[valsf]);
```

[0.03162278055453260, 0.003162277660207908, 0.0003162277660168384, **(3.2.5)**
0.00003162277660168379, $3.162277660168379\ 10^{-6}$, $3.162277660168380\ 10^{-7}$,
$3.162277660168379\ 10^{-8}$, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]

特に絶対値が大きい浮動小数に対して、高精度かつ subtract による減算を適用することで有効桁数の少ないデータが 0 となっている点に注意してください。
ここで、得られた減算結果から加算を計算します。
16桁精度の場合：

```
> add(v,v=(3.2.3));
```

$$-1.756820906469242\ 10^{9} \qquad \textbf{(3.2.6)}$$

24桁精度の場合：

```
> add(v,v=(3.2.4));
```

$$0.03513641888556182 \qquad \textbf{(3.2.7)}$$

16桁精度の場合、減算による有効桁数の損失は一部回避できているものの、絶対値の大きい数（有効桁数が少ないため本来は限りなく 0 に近いデータ）が残ってしまい、その結果大きな誤差値となってしまっています。精度を 24 桁まで上げる処理も追加して行うことで、Maple ではより高精度の計算結果を得ることが可能です。