数値計算法 第2回

誤差 (1)

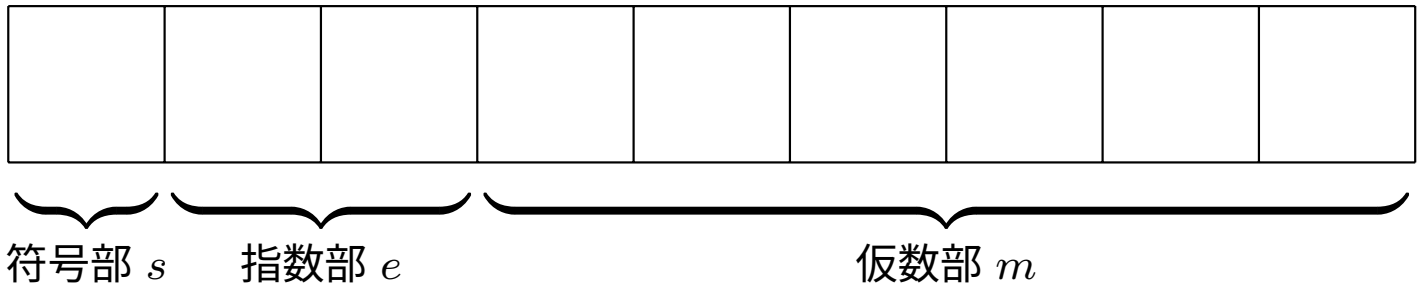
濱本 雄治 [1]

情報工学部 情報通信工学科

2025 年 4 月 21 日

[1] hamamoto@c.oka-pu.ac.jp

復習: 10 進 8 桁の浮動小数点数

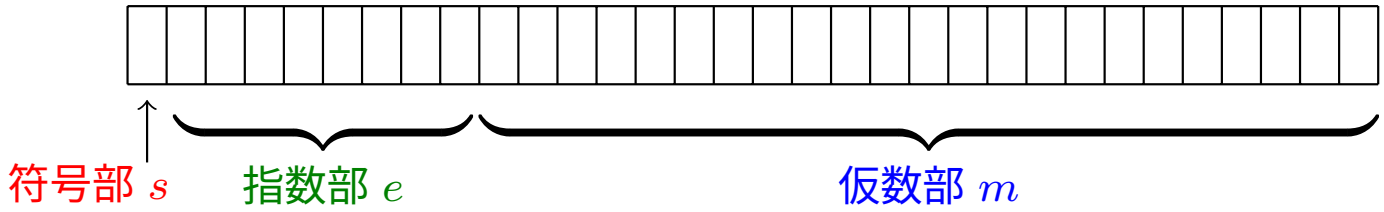


- 符号も含めた小数

$$(-1)^s \times 0.m \times 10^{e-50} \qquad (s = 0, 1)$$

- $s = 0, 1$ なので符号部は 1 ビット (2 進数 1 桁) でよい
- m は 6 つの数字の並びを表し、一番左の数字は 0 でない

復習: IEEE 単精度 (32 ビット) 浮動小数点数



- 符号部は 1 ビットの 2 進数で $s = 0, 1$
- 指数部は 8 ビットの 2 進数で $(00000000)_2 \leq e \leq (11111111)_2$
- 仮数部は残りの 23 ビット分だけ 0 or 1 を並べて表現
- $0 < e < 255$ のとき $(-1)^s \times (1.m)_2 \times 2^{e-127}$
- $e = 0, 255$ のとき

	$m = 0$	$m \neq 0$
$e = 0$	符号付きゼロ $(-1)^s \times 0$	$(-1)^s \times (0.m)_2 \times 2^{-126}$
$e = 255$	符号付き無限大 $(-1)^s \times \infty$	非数 (Not a number, NaN)

前回の小テストの解説

10 進数の -3 を IEEE 形式単精度浮動小数点数のビット列で表せ。
ただし例えば 0 が n 桁続くとき $\underbrace{0\cdots0}_{n\text{ 桁}}$ のように略記してよい。

$$
\begin{aligned}
-3 &= (-1)^1 \times (11)_2 && \leftarrow 3 \text{ を } 2 \text{ 進数表示} \\
&= (-1)^1 \times (1.1)_2 \times 2^{128-127} && \leftarrow \text{小数点のシフト}
\end{aligned}
$$

$$
\therefore s = 1, \quad e = 128 = 2^7 = (1\underbrace{0\cdots0}_{7\text{ 桁}})_2, \quad m = 1\underbrace{0\cdots0}_{22\text{ 桁}}
$$

1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

誤差

- コンピュータの浮動小数点数は近似値なので、誤差を含む

$$(\text{絶対})\,\text{誤差} = \text{近似値} - \text{真の値}$$

- 誤差の絶対値より相対値の方が重要

$$\text{相対誤差} = \frac{(\text{絶対})\,\text{誤差}}{\text{真の値}} = \frac{\text{近似値}}{\text{真の値}} - 1$$

相対誤差の例

- 円周率の真の値 $\pi = 3.141592\cdots$ を
  - $\pi \simeq 3.14$ と近似すると

$$\text{絶対誤差} = 3.14 - 3.141592\cdots = -0.001592\cdots$$

$$\text{相対誤差} = \frac{-0.001592}{3.141592\cdots} = -0.000506\cdots$$

  - $\pi \simeq 3$ と近似すると

$$\text{絶対誤差} = 3 \quad - 3.141592\cdots = -0.141592\cdots$$

$$\text{相対誤差} = \frac{-0.141592}{3.141592\cdots} = -0.0450\cdots$$

精度

- 近似値が真の値とどれだけ一致するか

精度の見積り方

- 相対誤差の小数点以下何桁目にゼロでない数字が現れるかで定義
- 精度の桁数 $= -\log_{10}|$ 相対誤差 $|$ と定義してもよい

- $\pi \simeq 3.14$ と近似すると

$$\text{相対誤差} = -0.000506\cdots \quad \to \quad \text{精度 4 桁}$$

$$-\log_{10}|\text{相対誤差}| \simeq 3.29 \quad (\sim 4)$$

- $\pi \simeq 3$ と近似すると

$$\text{相対誤差} = -0.0450\cdots \quad \to \quad \text{精度 2 桁}$$

$$-\log_{10}|\text{相対誤差}| \simeq 1.34 \quad (\sim 2)$$

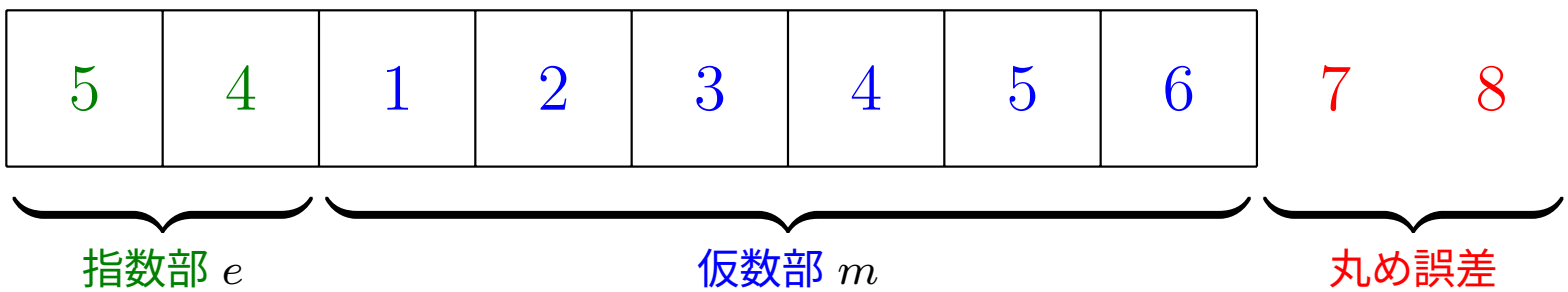
誤差の種類

誤差	原因	備考
丸め誤差	有効桁数が有限	
情報落ち	値に大きな差がある数の加減算	
桁落ち	値が近い数の減算	次回
打切り誤差	無限回の演算を有限回で打ち切り	次回
離散化誤差	連続量を離散値で近似	次回

丸め誤差

- コンピュータの有効桁数が有限であることに起因
- 1234.5678 を 10 進 8 桁の浮動小数点数で表すと

$$1234.5678 = 0.12345678 \times 10^{54-50}$$



$$\text{丸め誤差} = 1234.56 - 1234.5678 = -0.0078$$

$$\text{相対誤差} = \frac{-0.0078}{1234.5678} = -0.00000631 \quad \text{(精度 6 桁)}$$

- 10 進小数 0.1 を IEEE 単精度浮動小数点数で表すと

$$0.1 \times 2 = 0.2 + 0$$

$$0.2 \times 2 = 0.4 + 0$$

$$0.4 \times 2 = 0.8 + 0$$

$$0.8 \times 2 = 0.6 + 1$$

$$0.6 \times 2 = 0.2 + 1$$

$$0.2 \times 2 = 0.4 + 0 \quad (\text{以下、繰り返し})$$

$$\vdots$$

$$\therefore 0.1 = (0.000110011\cdots)_2 = (1.\underbrace{10011\cdots}_{\text{23 桁}})_2 \times 2^4$$

$$\text{丸め誤差} = (-0.\underbrace{0\cdots0}_{\text{23 桁}}\underbrace{1100\cdots}_{\text{丸め誤差}})_2 \times 2^4$$

情報落ち

- 値に大きな差がある数の加減算に起因
- 10 進 8 桁の浮動小数点数 (仮数部が 6 桁) で

$$123.456 + 0.0001$$

を計算すると

$$\begin{array}{rcccccccccccc} & 1 & 2 & 3 & . & 4 & 5 & 6 & & & & & \\ + & & & 0 & . & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline & 1 & 2 & 3 & . & 4 & 5 & 6 & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} \end{array}$$

となり 0.0001 の加算が無視される

情報落ちの例: 計算機イプシロン

- 「1 より大きい最小の数」と「1」との差
- 計算機イプシロンより小さな数を 1 に足し引きすると情報落ち
- IEEE 単精度浮動小数点数の場合

$$\varepsilon = (1.\underbrace{0\cdots0}_{22\text{ 桁}}1)_2 - (1.\underbrace{0\cdots0}_{22\text{ 桁}}0)_2$$

$$= (0.\underbrace{0\cdots0}_{22\text{ 桁}}1)_2$$

$$= 2^{-23} \simeq 1.19 \times 10^{-7}$$

- IEEE 倍精度浮動小数点数の場合

$$\varepsilon = 2^{-52} \simeq 2.22 \times 10^{-16}$$

C 言語での計算機イプシロン

flt_dbl_epsilon.c

```
#include <stdio.h>
#include <float.h> /* FLT_EPSILON, DBL_EPSILON */
int main(){
  printf("%.8e\n", FLT_EPSILON);
  printf("%.8e\n", DBL_EPSILON);
}
```

実行結果

```
$ gcc flt\_dbl\_epsilon.c
$ ./a.out
1.19209290e-07
2.22044605e-16
```

プログラムで確認

epsilon.c

```
#include <stdio.h>


int main(){
  float eps = 1.0;
  for(int i=0; i<30; ++i){
    printf("%.6e   %.12f   ", eps, 1 + eps);
    printf(1+eps==1? "true\n": "false\n");
    eps /= 2.0;
  }
}
```

実行結果

```
$ gcc epsilon.c
$ ./a.out
1.000000e+00   2.000000000000   false
5.000000e-01   1.500000000000   false
.
.
.
2.384186e-07   1.000000238419   false
1.192093e-07   1.000000119209   false
5.960464e-08   1.000000000000   true
2.980232e-08   1.000000000000   true
```