浮動小数点型と数値計算 (その 1)

山本昌志*

2006 年 2 月 6 日

1　本日の学習内容

本日は教科書 [1] の第 8 章である．その中でも倍精度実数型 (double) の誤差について学習する．C 言語の double で宣言された変数は，64 ビットで表現される．この有限のビット数で表現されるため，どうしても誤差は避けられない．本日は，これに起因する以下の誤差について学習する．

- 丸め誤差
- 情報落ち
- 打ち切り誤差

2　C 言語での数の表現

2.1　整数型 (int)

本日は実数型の誤差について学習することになっているが，整数型についても述べておく．整数の場合，C 言語の int では 32 ビット (4 バイト) で表現される．このビット数で，- 2147483648〜2147483647 までを表す．整数型の場合，この範囲であれば後で述べる実数型で生じる誤差は起きない．整数型で気を付けることは，演算結果がこの範囲を超えるオーバーフローのみである．

最小整数と最大整数は，リスト 1 のプログラムで確認できる．この内容を理解するためには整数の 2 の補数表現の理解が不可欠である．忘れた者は，私の講義ノートの 2 の補数 のところを見よ．

リスト 1: int 型の最小と最大の確認のプログラム

```
1 #include <stdio.h>
2
3 int main(void){
4
5   int min, max;
6
7   min=0x80000000;
8   max=0x7fffffff;
```

*独立行政法人　秋田工業高等専門学校　電気情報工学科

```
 9
10    printf("size of int = %d\n",sizeof(int));
11    printf("min = %d\n",min);
12    printf("max = %d\n",max);
13
14    return 0;
15  }
```

2.2 倍精度実数型 (double)

実際の C 言語での小数の表現方法，すなわちメモリーのビットパターンがどうなっているかは，結構難しい．以前のプリントで少し示している が，これをすべて理解するのは結構大変である．興味のある者は，自分で勉強せよ．

教科書に示している 2 つの実数を 2 進数で表現すると，

$$(12.8)_{10} = (1.10011001100110011001\cdots \times 10^{11})_2 \tag{1}$$

$$(0.5)_{10} = (1.0 \times 10^{-1})_2 \tag{2}$$

となる．C 言語の倍精度実数型 (double) の場合，2 進数の小数点以下と指数部，符号がメモリーに格納されることになる．$(12.8)_{10}$ を 2 進数で表現すると循環小数になるので，コンピューターではその近似値を扱うことになる．私のパソコンでは，

$$(12.8)_{10} \Rightarrow (1.1001100110011001100110011001100110011001100110011010 \times 10^{11})_2 \tag{3}$$

という値が格納されていた．元々，循環小数であったものが途中で丸められているのである．したがって，12.8 ではなくその近似値

$$\begin{aligned}(1.1001100110011001100110011001100110011001100110011010 \times 10^{11})_2 \\ = (12.800000000000000071054273576010018587112426757812 5)_{10}\end{aligned} \tag{4}$$

がメモリーに格納されている．要するにコンピューターでは正確に実数を扱うことが出来ないのである．実数を扱うプログラムを書く場合，このことは肝に命じておかなくてはならない．

一方，$(0.5)_{10}$ は循環小数とはならず，2 進数の有限の桁数で表現できる．この場合は，メモリーに格納される値も正確に $(0.5)_{10}$ である．C 言語の倍精度実数型 (double) の場合，2 進数の仮数部が 53 桁以内であれば正確に実数を表現できる．仮数部は 53 ビットの情報で表現していることによる．したがって，誤差は 54 ビット目あたりで生じ，およそ $(2^{-54}) = 5.5 \times 10^{-17}$ となる．C 言語の double 型の演算では，10^{-16}〜10^{-17} 程度の計算誤差が生じる．この辺の詳しい話は，私の講義ノートの浮動小数点表示[1]を見て欲しい．

付録にこの辺を確認するプログラムを載せておく．

3 倍精度実数のいろいろな誤差

倍精度実数型 (double) のいろいろな誤差を，実際のプログラムで体験してもらう．

[1] http://www.akita-nct.jp/ yamamoto/lecture/2005/2E/7th/html/node5.html#SECTION00054000000000000000

3.1 丸め誤差

先程述べたように，実数は有限のビット数で表現されるため，2 進数での桁数が無限に必要な場合はメモリー中のデータには誤差が含まれる．この誤差を丸め誤差 (rounding error) と言う．以下のプログラムを実行して，それを確認せよ．

9 行 `%80.75f` は 80 カラム用意して，小数点以下 75 桁で表示せよという意味.

リスト 2: 丸め誤差を確認するプログラム

```
1  #include <stdio.h>
2
3  int main(void){
4
5    double x;
6
7    x=0.3;
8
9    printf("%80.75f\n",x);
10
11   return 0;
12 }
```

練習問題

[**練習 1**] リスト 2 を書き換えて，誤差が生じない例を探せ．

[**練習 1**] 倍精度実数型で 1 以上の整数の場合はどうなるか?．プログラムを書き換えて調べよ．

3.2 情報落ち

倍精度実数型の精度は，およそ $(2^{-54}) = 5.5 \times 10^{-17}$ と先ほど述べた．この精度よりも大きさの差 (絶対値) が小さい 2 つの実数の和と差の演算はできない．このことを以下のプログラムで確認する．`z` の値は，`x` の値に 1000000 回 `y` の値を加算するプログラムとなっている．すなわち，

$$z = x + 1000000y \tag{5}$$

である．計算結果がどうなるか確認せよ．

リスト 3: 情報落ち誤差を確認するプログラム

```
1  #include <stdio.h>
2
3  int main(void){
4
5    double x, y, z;
6    int i;
7
8    x=1.0;
9    y=1e-17;
10
11   printf("%80.75f\n",x);
12   printf("%80.75f\n",y);
```

```
13
14
15      z=x;
16      for(i=1;i<=1000000;i++){
17          z+=y;
18      }
19
20      printf("%80.75f\n",z);
21
22      return 0;
23  }
```

練習問題

[**練習 1**] リスト 3 を書き換えて，誤差が生じない例を探せ．

[**練習 2**] 積や商の場合はどうか?．調べよ．

3.3 打ち切り誤差

三角関数を計算する場合，以下のような級数を使う．

$$\begin{aligned}\sin x &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \cdots \\ &= \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^{2n-1}}{(2n-1)!}\end{aligned} \tag{6}$$

コンピューターでは無限級数を計算することはできない．これをきちんと計算するためには，無限界の計算回数が必要で，無限の時間がかかる．それに，精度が有限であるため，ある程度以下の計算は無意味である．そのため，精度内に計算が収まったら，計算を止めるようになっている．途中で計算を打ち切るので，この誤差を**打ち切り誤差**と言う．

4 倍精度実数の精度

教科書 [1] の p.243 に書かれているように，実数型の精度はヘッダーファイル<float.h>に書かれている．倍精度実数型 (double) の場合の精度は DBL_EPSILON の値で指定されている．調べてみよう．手順は，以下の通りである．

1. ヘッダーファイル<float.h>を探す．以下のコマンドをターミナルから入力してしばらく待つ．

   ```
   find /usr/lib -name float.h
   ```

2. ヘッダーファイルがある場所へ，cd コマンドで移動する．

   ```
   cd ディレクトリー
   ```

3. cat コマンドを使って，ファイルの中身をディスプレイに表示させる．

   ```
   cat ファイル名
   ```

4. DBL_EPSILON が書かれている場所を探す.

ファイル中の特定の語句を探す場合は，grep コマンドを使うのが普通である．以下のようににしても良い.

```
grep "DBL_EPSILON" float.h
```

私のパソコンの場合，DBL_EPSILON の値は

```
#define DBL_EPSILON 2.2204460492503131e-16
```

となっていた.

5 ループ文の注意

for 文などのループ文で，ループ回数の制御に実数型は極力使ってはならない．実数型を使うと，丸め誤差や情報落ちにより，所定の回数ループ文が実行されないことがある．実際の例は，教科書 [1] の List 8-5 や List 8-6 の通りである.

整数を制御変数にして，実数の値はその整数を用いた演算により求めるのは常識的な方法である．教科書の List 8-7 のようにする.

6 付録

実数の表現方法を確認するプログラムを載せておく.

リスト 4: $(12.8)_{10}$ をメモリーに格納しその状態を確認するプログラム

```
1  #include <stdio.h>
2
3  int main(void){
4    double f;
5    char *a;
6
7    f=12.8;
8
9    a=(char *)&f;
10
11   printf("%p\t%x\n",a,(unsigned char)*a);
12   printf("%p\t%x\n",a+1,(unsigned char)*(a+1));
13   printf("%p\t%x\n",a+2,(unsigned char)*(a+2));
14   printf("%p\t%x\n",a+3,(unsigned char)*(a+3));
15   printf("%p\t%x\n",a+4,(unsigned char)*(a+4));
16   printf("%p\t%x\n",a+5,(unsigned char)*(a+5));
17   printf("%p\t%x\n",a+6,(unsigned char)*(a+6));
18   printf("%p\t%x\n",a+7,(unsigned char)*(a+7));
19
20   printf("%60.55f\n",f);
21
22
23   return 0;
24 }
```

実行結果

```
0xbf909928      9a
0xbf909929      99
0xbf90992a      99
0xbf90992b      99
0xbf90992c      99
0xbf90992d      99
0xbf90992e      29
0xbf90992f      40
  12.800000000000000710542735760100185871124267578125000000
```

参考文献

[1] 紀平拓男, 春日伸弥. プログラミングの宝箱　アルゴリズムとデータ構造. ソフトバンクパブリッシング(株), 2004 年.