シミュレーション概論ノート第2講 (計算機の誤差)

1　全般的注意

計算機における数値計算で取り扱われる数は厳密な意味の実数ではなく、あまり桁の長くない浮動小数点数である。今回の講義では浮動小数点に絡んで生じる数値計算に伴う丸め誤差や桁落ちの問題を例を通して理解していこう。

まず注意しておかなければいけないのは計算機では数値は基本的にビットを単位とする２進数によって表現される。実際にはこれを４ビット毎にまとめた１６進表現のものが多い。従って数 a は

$$a = \pm f \times \beta^m \tag{1}$$

という形に表現される。この場合 β を基数といい１６進表現であれば $\beta = 16$ である。f は仮数部で

$$f = \sum_j^n \frac{c_j}{\beta^j} \tag{2}$$

であり $0 < c_j < \beta - 1$ を満たす整数である。(1) 式で表現された数を β 進 n 桁浮動小数点数と呼ぶ。

例えば大多数の計算機において単精度の実数を表現するために１語 32 ビットを用いている。このうち最初の１ビットに符号、次の７ビットに指数 m、最後の２４ビットに仮数部の各桁が４ビットづつまとめられている。つまり１６進６桁の表現である。$16^6 = 10^{7.22}$ であるから１０進で８桁の有効数字しかないことに注意すべきである。

β進 n桁の表現における最大値は１６進６桁では 7.237005×10^{75} であり絶対値最小数は $45.397605 \times 10^{-79}$ である。最大値を越すとオーバーフローと呼ばれ計算は中断する。一方、最小数以下では０に置き換えて計算が実行される。この上限は充分大きい様に感じられるかもしれないが割算や対数ではしばしばオーバーフローによって計算が中断する場合があるので細心の注意が必要である。

2　丸め誤差

計算機の内部では数値は有限桁で表現されるので、計算機に入力される数値は原則として切捨てか丸め（１０進の場合の４捨５入）を受ける。一般に総称して丸めと呼ぶことが多い。丸めによって生じる誤差を丸め誤差という。

丸め誤差の累積を示す一つの例として、積分

$$I = \int_0^1 \frac{dx}{1+x^2} = \frac{\pi}{4} = 0.7853982 \tag{3}$$

を刻み幅 h を小さくしながら台形則

$$I_h = h[\frac{1}{2}f(a) + \sum_{j=1}^{n-1} f(a+jh) + \frac{1}{2}f(b)], \quad h = \frac{b-a}{n} \tag{4}$$

$$a = 0, \quad b = 1, \quad f(x) = \frac{1}{1+x^2} \tag{5}$$

によって数値積分をしてみる。この時、理論的には h^2 に比例する誤差があることが期待される[1] 。実際には刻みをどのように選択すればいいかを含めてプログラムを組む必要があるがここでは簡単に一定の刻みとする。アルゴリズムは極めて簡単で、まず

(i) 初期設定；$ev = \pi/4$ とする。

(ii) $h = 2^{-k}$ として $k = 1$ から 12 まで動かすループの設定をする。

(iii) ループ内で和に関係ない部分を台形公式 (3) の括弧内の値 s の初期値とする。

(iv) 和を取る（更にループ）

(v) 積分の値を求めるために h 倍して値を書き出す。

(vi) k のループの終了。プログラムの終了。

ということになる。Fortran77[2] で書き下すと以下の様になる。一般に $\pi = 4\mathrm{Arc}\tan(1)$ を使うと必要な精度だけ π を表記できることが知られている。また 100 行では Format をして書いてある。x はスペース、f は実数型を現している。$2f14.10$ は２つの実数変数を全体で１４桁、小数点以下１０桁表記させている。

```
      s=0.0
      ev=atan(1.0)

      do 10 k=1,12

        n=2**k
        h=1.0/n

        s=0.5*(1.0+0.5)

        do i=1,n-1
          s=s+1/(1+(h*i)**2)
        enddo

        s=s*h

        err=abs(s-ev)

      write(6,100) h,err
100   FORMAT(5x,2f14.10)
10    continue

      end
```

因みに Fortran90 自由形式であれば以下のようになる。宣言文の箇所の違いに注意。

```
real(8):: s=0.0,ev,h,err
```

[1] この説明は後ろに手で書いて行ないます。式が続くので

[2] Fortran90 の文法書は http://hyper.gaia.h.kyoto-u.ac.jp/fortran90.htm に公開されている。

```
ev=atan(1.0_8)

   do k=1,12

      n=2**k
      h=1.0_8/n
      s=0.5_8*(1.0_8+0.5_8)


      do i=1,n-1

         s=s+1.0_8/(1.0_8+(h*i)*(h*i))

      enddo


      s=s*h
      err=abs(s-ev)
      print'(5x,2f28.20)',h,err

   enddo

end
```

明らかに h を小さくしすぎると誤差がかえって大きくなることが読みとれる。
ここで図でその傾向を示した方がいい。そのためにコンパイルした後

$$a.out > daikei.dat \quad \hookleftarrow$$

としてみる。そうすると daikei.dat というデータファイルが出来る。次に gnuplot というソフトが使えるのでそれを用いて図を描くことにする。

$$gnuplot \quad \hookleftarrow$$

としてみよう。まずそのまま

$$plot \quad 'daikei.dat' \quad \hookleftarrow$$

と打ち込む。すると図が出来るがデータの点が 0 のあたりに集積して具合がよろしくない。そこで

$$set \quad logscale \quad xy \quad \hookleftarrow$$

として両方の軸を対数表示にしてからもう 1 回 plot 'daikei.dat' $\hookleftarrow$ と打ち込むと等間隔のデータが出来る。この傾きが-2 であれば理論通りであるが $h = 0$ の近傍で反対に大きくなっているのがよくわかる。尚、set nologscale とすると普通の線形のスケールに戻ることができる。gnuplot を終了するのは exit である。help によって on-line manual が現れるので利用すること。
また C でプログラムを書き直すと

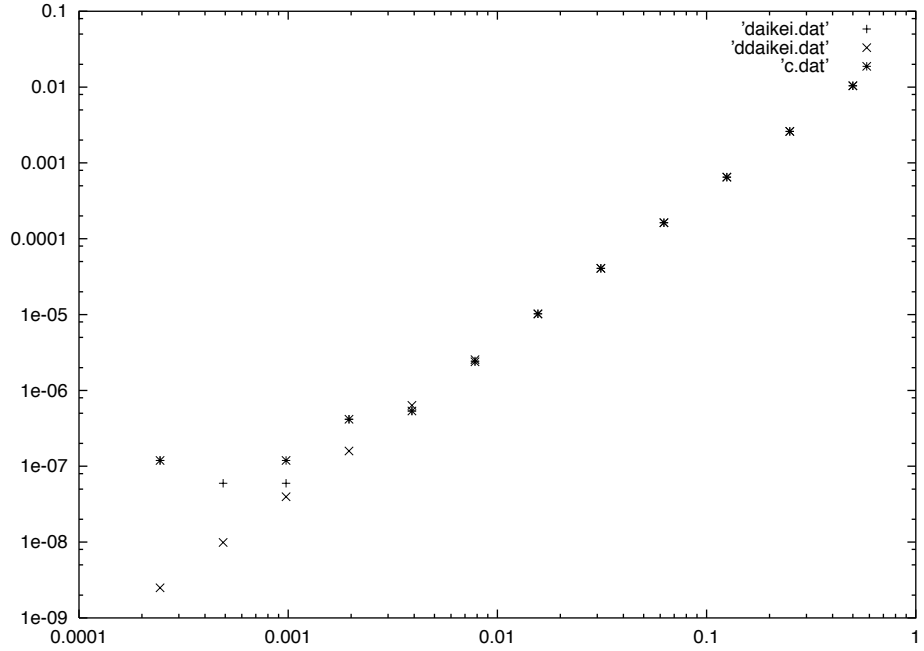

図 1: 丸め誤差。単精度の Fortran, 倍精度の Fortran,C の結果が図示してある。

```
#include <stdio.h>
#include <math.h>

main()
{
  int i, k, n;
  float h, s, ev, err;

  s = 0.0;
  ev = atan(1.0);

  for(k = 1; k <= 12; k++) {
    n = pow(2, k);
    h = 1.0 / n;

    s = 0.5 * (1.0 + 0.5);

    for(i = 1; i <= n-1; i++) {
      s = s + 1.0 / (1.0 + (h * i) * (h * i));
    }

    s = s * h;
```

```
    err = fabs(s - ev);

    printf("     %14.10f%14.10f\n", h, err);
  }

  return 0;
}
```

となる。倍精度の Fortran のプログラムについては省略するので各自の演習とすること。

以上の計算からわかることは刻みをあまりにも小さく取ると丸め誤差の累積の影響が顕著に現れるという事である。従って精度の向上のためにむやみに刻みを小さく取ることは避けるべきである[3] 。

3 情報落ち

大きさの異なる 2 つの浮動小数点数の加減を行なう場合、小さい方の指数部を大きい方の指数部に合わせて、仮数部の桁ずらしを行なってから加減が実行される。従って両者の大きさが桁違いの場合、小さい方の情報が失われることは明らかである。この現象を情報落ちという。

情報落ちは多数の数の加算（従って無限区間の積分など）には実際上の問題になる。例として

$$S_n = \sum_{i=1}^{n} \frac{1}{i^2} \tag{6}$$

を計算してみよう。$S_\infty = \zeta(2) = \pi^2/6 = 1.644934$ であることが知られている。しかし数値的に無限項の和を取ることは不可能なので値が収束した場合に計算を打ち切ることになる。例えば

```
          ss=(4.0*atan(1.0))**2/6.0

           write(*,*) 'ss=',ss

           n=1000000
           s=0.0

C   forward summation

           do i=1,n
              v=1.0/real(i)**2
              sv=s+v
              if (sv.eq.s) goto 11
              s=sv
           enddo
```

[3] 刻みを小さくすることは CPU を浪費することに繋がるのは言うまでもない

```
 11         ie=i-1
            write(6,100) ie,s,v

C       backwars summation

            s=0.0
            do i=n,1,-1
              v=1.0/real(i)**2
              s=s+v
            enddo
            v=1.0/real(n)**2

            write(6,100) n,s,v

 100       format(5x,i8,2f12.8)
            end
```

という風にプログラムを作ってみる。或は自由形式で

```
ss=(4.0*atan(1.0))**2/6.0
print*,'ss=',ss

n=1000000
s=0.0


! forward summation

  do i=1,n

     v=1.0/real(i)**2
     sv=s+v
     if(sv==s)exit
     s=sv

  enddo


  ie=i-1

  print100,ie,s,v

! backward summation
```

```
  s=0.0

  do i=n,1,-1

     v=1.0/real(i)**2
     s=s+v

  enddo

  v=1.0/real(n)**2

  print100,n,s,v


100 format(5x,i8,2f12.8)

end
```

ここで条件の判断に if 文が用いられる。条件 A のときに B それ以外は C という場合は

```
       if A then
         B
       else
         C
       endif
```

という具合に書く。if A goto 11 というのは条件 A を満たせば 11 行に飛べという命令。詳細は Fortran 文法書を見よ.

C で対応するプログラムを書くと

```
#include <stdio.h>
#include <math.h>

main()
{
  long int i, n, ie;
  float s, ss, v, sv;

  ss = 4.0 * atan(1.0);
  ss = ss * ss / 6.0;

  printf("ss = %12.8f\n", ss);

  n = 1000000;
```

```
  s = 0.0;

  /* forward summation */
  for(i = 1; i <= n; i++) {
    v = 1.0 / (((float)i) * ((float)i));
    sv = s + v;
    if(sv == s) {
      ie = i - 1;
      printf("     %8d%12.8f%12.8f\n", ie, s, v);
      break;
    }
    s = sv;
  }

  /* backwars summation */
  s = 0.0;
  for(i = n; i >= 1; i--) {
    v = 1.0 / (((float)i) *((float)i));
    s = s + v;
  }
  v = 1.0 / (n * n);

  printf("     %8d%12.8f%12.8f\n", n, s, v);

  return 0;
}
```

とでもしておけばよい。

この時の答えは前から順番に足すと 4096 項目で打ち切られて 1.64472532 となる。一方、$n = 10^6$ から始めて逆に遡って和を取ると 1.64493299 となり丸め誤差を除いて正しい結果を得る。このような無限級数の和は前もって解析的に変形したり適当な加速法を用いる必要がある。

4 桁落ち

ある意味で前節での情報落ちはそれほど深刻な問題ではないが計算機特有のこれから説明する桁落ちは深刻な問題である。2 つの浮動小数点数の大きさがほぼ等しく n 桁からなる仮数部のうち上位 k 桁が等しい場合を考える。2 つの引き算を行なうと上位 k 桁が打ち消して結果の仮数部には $n - k$ 桁の情報しか残らないことになる。これを桁落ちという。

桁落ちを防ぐには相応の工夫が必要である。例えば数値計算の常識（伊理正夫・藤野和建著（共立））では $x = 3.1834 \times 10^{-3}$ に対して $1 - 1/\sqrt{1+x}$ の計算をそのまま行なうと有効数字 2 桁しかない結果を得るが、分子を有理化してから計算すると有効数字 5 桁（全桁正しい）結果を得ることを示している。同様に 2 次方程式の根を求めるときにも正しい結果を得るためには根と係数

の関係から定めるべしと説明してある。

ここでは $\sqrt{x}$ の $x=2$ における微分の値 $1/(2\sqrt{2})=0.3535534$ を刻み幅 h の中心差分

$$\frac{f(x+h)-f(x-h)}{2h}, \quad f(x)=\sqrt{x} \tag{7}$$

によって計算してみる。アルゴリズムという程のものはないので直接 Fortran のプログラムを書くと

```
      ev=0.3535534

       do k=1,30

         h=1.0/2.0**k
         dif=(sqrt(2+h)-sqrt(2-h))/(2.0*h)
         err=abs(dif-ev)

         write(6,100) h,err
      enddo

 100  FORMAT(5x,2f14.8)

      end
```

となる。自由形式では

```
ev=0.3535534

do k=1,30


   h=1.0/2.0**k
   dif=(sqrt(2.0+h)-sqrt(2.0-h))/(2.0*h)
   err=abs(dif-ev)

   print'(5x,2f14.8)',h,err

enddo

end
```

である。計算すると（gnuplot で図示せよ）誤差は $h=1/2^7$ で最小になるがそれ以降は増大に転じる。それどころか $h=1/2^{28}$ で差分そのものが 0 になってしまう。

因みに C のプログラムは

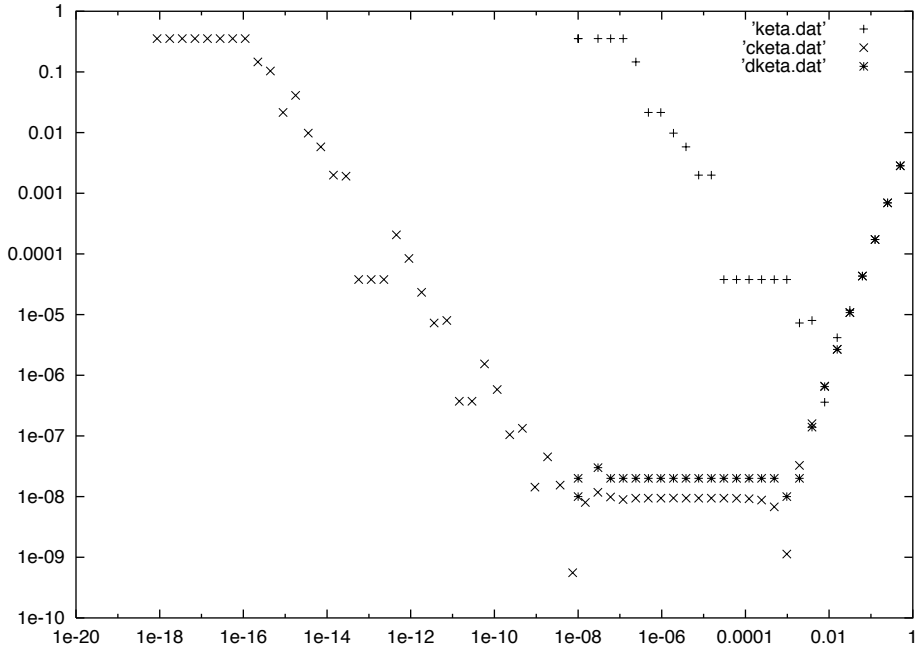

図 2: 桁落ちの例。横軸は刻み、縦軸は誤差を表す。やはり単精度と倍精度の FORTRAN,C がそれぞれデータとして与えられている。

```
#include <stdio.h>
#include <math.h>

main()
{
  int k;
  double h, ev, dif, err;

  ev = 0.3535534;

  for(k = 1; k <= 30; k++) {
    h = 1.0 / pow(2.0, k);

    dif = (sqrt(2.0 + h) - sqrt(2.0 - h)) / (2.0 * h);

    err = fabs(dif - ev);

    printf("     %14.8g%14.8g\n", h, err);
  }

  return 0;
}
```

とでもしておけばいいであろう。

一般に普通の大きさを持った値に対して加減を行なった場合に著しく小さな値が出た場合は桁落ちを疑ってみる必要がある。桁落ちは数値計算のいろいろな所に生じる現象であるが、それに気付かずに計算を続行させしばしば重大な誤りを犯す場合がある。桁落ちには充分気をつける必要があるのである。

問：$2.718282x^2 - 684.4566x + 0.3161592 = 0$ を次の２つの方法で解き、答えを比べよ。（１）２次方程式の解の公式を用いた場合、（２）大きい方の桁落ちしない解を x_1 として桁落ちする小さい解 x_2 との間の解と係数の関係を用いたもの。

5　数学的補足：台形公式の誤差に就いて

台形公式は図のように $y = f(x)$ を折れ線近似し、積分 $I = \int_a^b f(x)dx$ を台形の面積の和で近似したものである。折れ線を

$$y = F(x)$$

としたとき、

$$F(x) = \frac{x - x_{j+1}}{x_j - x_{j+1}} f(x_j) + \frac{x - x_j}{x_{j+1} - x_j} f(x_{j+1})$$

となる $(x_j \leq x \leq x_{j+1})$。

$f(x)$ を Taylor 展開することで $f(x)$ と $F(x)$ の誤差は、

$$f(x) - F(x) = \frac{f''(\xi)}{2}(x - x_j)(x - x_{j+1})$$

である事を示す事が可能である。但し $x_j \leq \xi \leq x_{j+1}$ であり、等式を充たすように ξ を選んである。これから $x_j \leq x \leq x_{j+1}$ で、

$$|f(x) - F(x)| \leq \frac{1}{2} \max_{x_j \leq \xi \leq x_{j+1}} \left|f''(\xi)\right| \left|(x - x_j)(x - x_{j+1})\right|$$

と評価できる。よって、

$$\begin{aligned}
\left| \int_{x_j}^{x_{j+1}} (f(x) - F(x))\, dx \right| &\leq \int_{x_j}^{x_{j+1}} |f(x) - F(x)|\, dx \\
&\leq \frac{1}{2} \max_{x_j \leq \xi \leq x_{j+1}} \left|f''(\xi)\right| \int_{x_j}^{x_{j+1}} \left|(x - x_j)(x - x_{j+1})\right| dx \\
&= \frac{h^3}{12} \max \left|f''(x)\right|
\end{aligned}$$

と評価できる。従って全区間では、

$$\begin{aligned}
\left| \int_a^b f(x)dx - \int_a^b F(x)dx \right| &\leq \frac{h^3}{12} \sum_{j=0}^{N-1} \max \left|f''(\xi)\right| \\
&\leq \frac{h^3}{12} N \max_{a<\xi<b} \left|f''(\xi)\right| \\
&= \frac{(b-a)^3}{12N^2} \max_{a<\xi<b} \left|f''(\xi)\right|
\end{aligned}$$

f, a, b は given なので N 大で誤差は $1/N^2 \sim h^2$ で押さえられる。

6　倍精度のプログラムについて

以下に倍精度の Fortran90 の固定形式のプログラム例を置く。まずは台形公式における丸め誤差について。

```
      double precision s,h,err,ev
      integer k,i,n

      s=0.0d0
       ev=datan(1.0d0)

       do 10 k=1,12

         n=2**k
         h=1.0d0/dble(n)

         s=0.5d0*(1.0d0+0.5d0)

         do i=1,n-1
           s=s+1.0d0/(1.0d0+(h*dble(i))**2)
         enddo

         s=s*h

         err=dabs(s-ev)

       write(6,100) h,err
100    FORMAT(5x,2f14.10)
10     continue

       end
```

次いで桁落ちのプログラム。

```
      real*8 ev,h,dif,err
      integer k

      ev=0.3535534

       do k=1,30

         h=1.0d0/2.0d0**k
         dif=(dsqrt(2.0d0+h)-dsqrt(2.0d0-h))/(2.0d0*h)
```

```
         err=dabs(dif-ev)

         write(6,100) h,err
      enddo

100   FORMAT(5x,2f14.8)

      end
```