第１章　数値計算における誤差

1.1　計算機における実数の表現

物理学の実験などでは，数字の有効桁を明示したい場合，

$$0.504 \times 10^3 \quad (= 504)$$
$$0.123 \times 10^{-1} \quad (= 0.0123)$$

のような書き方をすることがよくある。このように表現された数を（10 進）浮動小数点数と呼ぶ。

計算機の中では，実数は**２進浮動小数点数**で表現される。２進浮動小数点数の一般形は

$$\pm(0.d_1 d_2 \cdots d_t) \times 2^n \quad (= \sum_{i=1}^{t} d_i 2^{-i} \times 2^n) \tag{1.1}$$

となる。ここで，d_1,d_2,...,d_t はそれぞれ 0 または 1 である。± を**符号部**，$0.d_1 d_2 \cdots d_t$ を**仮数部**，2^n を**指数部**と呼ぶ。通常，指数 n は小数点のすぐ右の数字 d_1 が 1 になるように決める。このときの表現を**正規化表現**と呼ぶ。たとえば，0.001101×2^{-5} は正規化表現では 0.1101×2^{-7} となる。

２進浮動小数点数 (1.1) は，計算機の中で図 1.1 のように格納される。指数部の表現に k ビットを使う場合，ここで表現できる数は 2^k 通りとなる。したがって，指数の範囲を $L\,(<0)$ から $U\,(>0)$ までと決めた場合，$U-L+1 < 2^k$ ならば，この k ビットですべての指数が表現できる。また，仮数部の格納では，正規化表現で $d_1 = 1$ であることを用いて d_1 の格納を省略する（ケチ表現）。従って格納に必要なビット数は $t-1$ ビットとなる。以上の約束の下では，表現できる絶対値最大の数値，絶対値最小の数値はそれぞれ

$$(0.111\cdots1) \times 2^U = (1 - 2^{-t}) \times 2^U, \tag{1.2}$$
$$(0.100\cdots0) \times 2^L = 2^{L-1} \tag{1.3}$$

となる。なお，0 は上記の約束の範囲では表せない（常に $d_1 = 1$ だから）ので，すべてのビットが 0 の場合は 0 を表すと特別に約束するのが普通である。この場合，指数の取りうる範囲 $U - L + 1$ は 2^k より小さくなる。

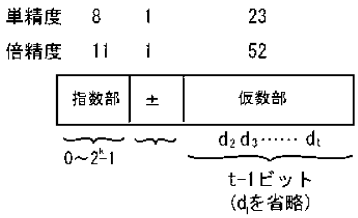

図 1.1: ２進浮動小数点数の格納

表 1.1: IEEE 方式の２進浮動小数点数に関する各種パラメータ

項目	単精度型	倍精度型
全ビット数	32	64
指数部ビット数	8	11
仮数部ビット数	23 $(t = 24)$	52 $(t = 53)$
L	-125	-1021
U	128	1024
絶対値最大の数	3.4×10^{38}	1.8×10^{308}
絶対値最小の数	1.2×10^{-38}	2.2×10^{-308}

最近のパソコンでは，２進浮動小数点数の表現に *IEEE* **方式**を使うのが普通である。IEEE 方式では，32 個のビットで１個の浮動小数点数を表す**単精度型**と 64 個のビットで１個の浮動小数点数を表す**倍精度型**とがある。それぞれについて，各部のビット数，L，U，表現できる絶対値最大・最小の数は表 1.1 のようになっている。

浮動小数点数に対して計算機内で演算を行った場合，結果の絶対値が表現できる最大の数より大きくなることがある。たとえば，非常に大きい数どうしの掛け算を行った場合，大きい数に対する指数関数の計算を行った場合などにこのようなことが起こりうる。これを**オーバーフロー**と呼ぶ。オーバーフローが起こると，プログラムは異常終了するのが普通である。一方，演算結果の絶対値が表現できる最小値より小さくなることがある。これを**アンダーフロー**と呼ぶ。アンダーフローが起こった場合，通常，計算機は演算結果を 0 で置き換えて計算を続行する。

1.2　丸め誤差

1.2.1　丸めと丸め誤差

計算機に入力の数値を与える場合，その数値が２進浮動小数点数として正確に表されるとは限らない。$\sqrt{2}$ のような無理数や，有効桁数が（２進数換

算で）仮数部の桁数より大きい数などはもちろん，10 進数では有限桁で表せる 0.2 のような数も，２進数では $0.0011001100110011\cdots$ のような循環小数になり，有限桁では表せない。これらの数を，それに近い２進浮動小数点で近似することを**丸め**と呼ぶ。10 進数の場合と同様，丸めの方式には**切り捨て**，**切り上げ**，**四捨五入**（２進数の場合は 0 捨 1 入）などがある。丸めによって生じる誤差を**丸め誤差**と呼ぶ。

丸めは演算を行った後でも必要になる。これは，仮数部が t 桁の浮動小数点数に対して演算を行った結果は，一般に仮数部 t 桁では正確に表せないからである。たとえば，(わかりやすくするため 10 進数で考えるが）0.123×10^0 と 0.567×10^{-3} という２つの仮数部 3 桁の浮動小数点数を加える場合，まず指数部を揃えるために桁ずらしを行うことにより，結果は

$$\begin{array}{rl} & 0.123000\times10^0 \\ +) & 0.000789\times10^0 \\ \hline & 0.123789\times10^0 \end{array}$$

となり，仮数部は 6 桁になる。この場合，四捨五入による丸めを行うことにより，結果を 0.124×10^0 のように表示できる。

いま，$x=(0.d_1d_2\cdots d_td_{t+1}\cdots)\times2^n$ という数を 0 捨 1 入により仮数部 t 桁の２進浮動小数点数に丸める場合を考える。隣り合う２個の浮動小数点数 $\underline{x}$，$\overline{x}$ を

$$\begin{aligned} \underline{x} &= (0.d_1d_2\cdots d_t)\times2^n \\ \overline{x} &= (0.d_1d_2\cdots d_t+2^{-t})\times2^n \end{aligned}$$

とおくと，

$$\underline{x}\le x\le\overline{x} \tag{1.4}$$

であり，x は $\underline{x}$，$\overline{x}$ のうち，近い方に丸められる（図 1.2)。丸めた結果を $\tilde{x}$ とすると，丸め誤差の上限は明らかに

$$|\,x-\tilde{x}\,|\le2^{-t+n-1} \tag{1.5}$$

となる。また，x は

$$2^{n-1}\le x<2^n \tag{1.6}$$

の範囲にあるから，丸めによる相対誤差は常に

$$\frac{|\,x-\tilde{x}\,|}{|\,x\,|}\le\frac{2^{-t+n-1}}{2^{n-1}}=2^{-t} \tag{1.7}$$

を満たす。

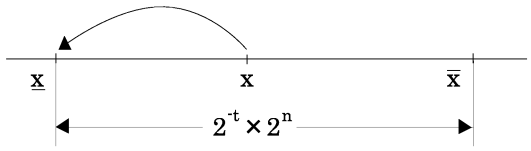

図 1.2: もっとも近い浮動小数点数への丸め

1.2.2　情報落ち

丸め誤差が原因となって生じる現象のうちで，数値計算上特に注意しなくてはならないものとして，情報落ちと桁落ちがある。ここでは，まず情報落ちについて説明する。

浮動小数点数に対する加算で，２つの数の指数部が異なる場合，たとえば

$$x = (0.d_1 d_2 \cdots d_t) \times 2^n,$$
$$y = (0.c_1 c_2 \cdots c_t) \times 2^m$$

の加算を行う場合には，まず桁ずらしを行って２つの数の指数部を合わせる必要がある。たとえば $m = n - 3$ のときは，

$$\begin{array}{lll} & 0.d_1 d_2 d_3 d_4 \cdots d_t & \times 2^n \\ +) & 0.0\ 0\ 0\ c_1 \cdots c_{t-3} c_{t-2} c_{t-1} c_t & \times 2^n \end{array}$$

のように，y の仮数部を３桁右にずらす必要がある。

いま，y が x に比べて極端に小さく，$m = n - t - 1$ となる場合を考えると，桁ずらしをして加算を行った結果は

$$\begin{array}{lll} & 0.d_1 d_2 \cdots d_t 0\ 0\ 0 \cdots 0 & \times 2^n \\ +) & 0.0\ 0\ \cdots 0\ 0\ c_1 c_2 \cdots c_t & \times 2^n \\ \hline & 0.d_1 d_2 \cdots d_t 0\ c_1 c_2 \cdots c_t & \times 2^n \end{array}$$

となる。ところが，結果を仮数部 t 桁になるよう 0 捨 1 入で丸めると，第 $t+1$ 桁の 0 以下の数値は省略されてしまい，結果は $0.d_1 d_2 \cdots d_t \times 2^n$ で x そのものとなってしまう。このような現象を，(y に入っている情報が完全に抜け落ちてしまうという意味で) **情報落ち**という。

情報落ちが問題となる場合として，級数の計算が挙げられる。たとえば

$$S_n = \sum_{i=1}^{n} \frac{1}{i^2} = 1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{n^2} \tag{1.8}$$

を計算する場合，$\lim_{n\to\infty} S_n = \frac{\pi^2}{6}$ だから，部分和は一定値 $\frac{\pi^2}{6}$ に近づいていく。一方，足される項は $\frac{1}{i^2}$ で，どんどん小さくなる。したがって，n が非常に大きい場合，ある i より後では，情報落ちが生じ，加算を行っても部分和は変化しなくなってしまう。また，非常に長い２本のベクトル $\mathbf{a}$，$\mathbf{b}$ の内積

を計算する場合も，情報落ちにより小さな要素の値が内積に反映されなくなることは起こりうる。

級数の計算で情報落ちの影響を少なくするには，数列の各項の大きさがわかっている場合，小さい項から順に足していくことが有効である。たとえば，上の例では

$$S_n = \frac{1}{n^2} + \frac{1}{(n-1)^2} + \frac{1}{(n-2)^2} + \cdots + \frac{1}{4} + \frac{1}{9} + 1 \tag{1.9}$$

と計算することにより，部分和と足される項との大きさの差が小さくなり，情報落ちが起きる可能性が減少する。しかし，ベクトルの内積の場合には，一般には各項の大きさに規則性がないため，このような方法は取りにくい。このような場合に情報落ちを避けるには，計算の精度を上げ，倍精度型あるいは4倍精度型を使うことが有効である。

1.2.3 桁落ち

浮動小数点数に対する減算（あるいは正の数と負の数との加算）で，両方の数が非常に近い場合，たとえば仮数部の上位 k 桁が等しい場合を考える。このとき，減算の結果は

$$\begin{array}{rrl} & x & = 0.d_1 d_2 \cdots d_k d_{k+1} \cdots d_t \times 2^n \\ -) & y & = 0.d_1 d_2 \cdots d_k c_{k+1} \cdots c_t \times 2^n \\ \hline & x-y & = 0.0\ 0\ \cdots 0\ b_{k+1} \cdots b_t \times 2^n \\ & & = 0.b_{k+1} \cdots b_t \times 2^{n-k} \end{array}$$

となり，結果の有効数字は $t-k$ 桁で2個の入力の有効数字から大きく減少する。これを**桁落ち**と呼ぶ。

桁落ちは，入力に含まれる相対誤差を大きく拡大する。これを見るため，減算の入力となるべき真の値を x，y，それぞれに加わった誤差を Δx，Δy として，減算の入力を

$$\tilde{x} = x + \Delta x$$
$$\tilde{y} = y + \Delta y$$

と書く。さらに，結果の真の値を $z = x - y$，その誤差を Δz と書く。すると減算の結果は（丸める前の時点で）

$$\begin{aligned} \tilde{z} &= \tilde{x} - \tilde{y} \\ &= (x + \Delta x) - (y + \Delta y) \\ &= (x - y) + (\Delta x - \Delta y) \\ &= z + \Delta z \end{aligned} \tag{1.10}$$

となる。これより

$$\begin{aligned}
&\Delta z = \Delta x - \Delta y \\
&\frac{\Delta z}{z} = \frac{\Delta x}{z} - \frac{\Delta y}{z} = \frac{x}{z}\cdot\frac{\Delta x}{x} - \frac{y}{z}\cdot\frac{\Delta y}{y} \\
&\left|\frac{\Delta z}{z}\right| \le \left|\frac{x}{z}\right|\left|\frac{\Delta x}{x}\right| + \left|\frac{y}{z}\right|\left|\frac{\Delta y}{y}\right|
\end{aligned} \tag{1.11}$$

が成り立つ。式 (1.11) は，結果 z の相対誤差が入力 x の相対誤差の $|x/z|$ 倍程度に拡大されうることを示す。いま，桁落ちが起こる状況のときは x と y が非常に近くて $|z| \ll |x|$ であるから，相対誤差の拡大率は非常に大きくなりうる。

桁落ちが起こりやすい例として，２次方程式 $ax^2 + bx + c = 0$ の解の公式

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{1.12}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{1.13}$$

がある。いま，$b > 0$ かつ $b^2 \gg 4ac$ とすると，$\sqrt{b^2 - 4ac} \approx b$ だから，x_1 の計算においては分子で桁落ちが起こる。一方，x_2 の計算では，分子が負の数 $-b$ と正の数 $\sqrt{b^2 - 4ac} \approx b$ との差だから，桁落ちは生じず，精度上の問題は起こらない。x_1 の計算において桁落ちを避けるには，次のように「分子の有理化」を行う。

$$\begin{aligned}
x_1 &= \frac{(-b + \sqrt{b^2 - 4ac})(-b - \sqrt{b^2 - 4ac})}{2a(-b - \sqrt{b^2 - 4ac})} \\
&= \frac{4ac}{2a(-b - \sqrt{b^2 - 4ac})} \\
&= \frac{2c}{-b - \sqrt{b^2 - 4ac}}
\end{aligned} \tag{1.14}$$

変形した式では，分母は符号が違う数どうしの引き算だから，桁落ちの問題は生じない。

桁落ちが起こるもう一つの例として，Taylor 展開による指数関数 e^{-x} の計算が挙げられる。この場合，たとえば $x = 20$ とすると e^{-x} は 10^{-9} 程度のオーダーの非常に小さい数になる。ところが，Taylor 展開の式

$$e^{-x} = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots \tag{1.15}$$

において最初のほうの項は 1 のオーダーであるから，計算の過程で結果が 0 に近くなるような引き算が生じ，そこで桁落ちが生じて精度が悪くなっていると予想される。これを防ぐには，

$$\begin{aligned}
e^{-x} &= \frac{1}{e^x} \\
&= \frac{1}{1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!}\cdots}
\end{aligned} \tag{1.16}$$

のように変形すればよい。式 (1.16) では分母の級数はすべての項が正であるから，桁落ちは起こらず，精度の良い計算が可能となる。

数値計算を行うに当たっては，なるべく上記のような式変形を行い，値が近い数どうしの減算を避けるようにすることが，精度を保つ上で重要である。ただし，このような工夫はいつでも可能とは限らないため，必要に応じて倍精度計算，４倍精度計算などを併用することも重要である。

1.3 打ち切り誤差

（第２回の授業へ）

1.4 誤差の伝播

（第２回の授業へ）