# 第3章　浮動小数点数と丸め誤差

**ゆとり教育：**
**円周率の暗記を披露　福田康夫官房長官**
「3.14159265358… とかね」。福田康夫官房長官は 10 日の記者会見で，今年度から始まった完全週休２日制などの「ゆとり教育」に関連して，円周率の暗記を披露した。
「円周率は 3.14 と約 3 と，どちらがいいと思うか」との質問に答えたもので，福田長官は「正確に計算するか早く計算するかの考え方だと思う。私だったらもっとたくさん言えますがね」と述べた後，数字をすらすら。

毎日新聞 (2002 年 4 月 10 日) より

何度か触れているように，数値計算とは有限桁の浮動小数点数を対象とした計算，という意味で使われることが多い。一方，現在の数学の理論，特に初等解析学（微分積分）は連続性を持った実数 $\mathbb{R}$ の性質に依存して成り立っている。実数には無限桁の小数でしか表現できない数も含まれるため，有限桁に押し込めてしまうと元の真値とはズレが生じる。このズレを一般には「丸め誤差」と呼び，数値計算にはどうしてもつきまとってくる厄介者である[1]。本章では浮動小数点数とこの丸め誤差について，具体例を交えつつ述べていく。

## 3.1　浮動小数点数の標準規格

**問題 3.1.1**
任意の $a, b, c \in \mathbb{R}$ における加算・乗算については以下の法則が成立する。

**交換則** $a + b = b + a, ab = ba$

**結合則** $a + (b + c) = (a + b) + c, a(bc) = (ab)c$

有限桁の浮動小数点数においては，結合則が成立しないことが知られている。その理由を説明し，具体例を示せ。

現在の PC や WS で良く用いられているのは IEEE で規格化されているもので，一般には IEEE754 standard と呼ばれているものである。

**例題 3.1.1 (IEEE754 standard)**
基数は 2 であり，図 3.1 に示すような有限桁の浮動小数点表現が定められている。単精度・倍精度の仮数部が実際の bit 列より 1bit 分長いのは，最初の bit が必ず 1 になるように正規化されるた

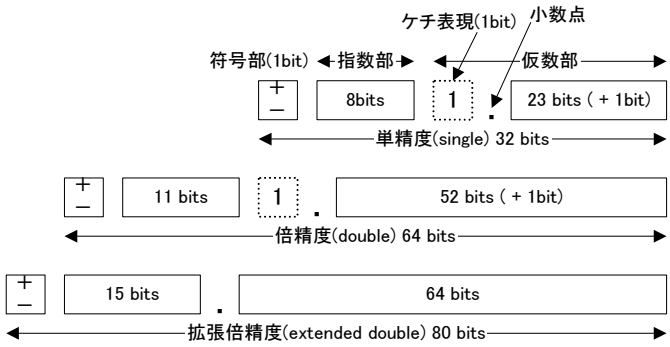


図 3.1: IEEE754 standard

め，実際にはその分は削ってあるからである。これをケチ表現と呼ぶ。C や Fortran 等のプログラミング言語で単精度・倍精度の値を 10 進数の文字列として出力すると，例えば 10.65387 は

```
10.65387
0.1065387e+02
1.065387e+01
1.065387e+1
1.065387e1
1.065387E+01
1.065387D+01
1.065387Q+01
```

のようになる。`e` や `E`，`D`，`Q` の前が 10 進数に変換後の仮数部，後ろが指数部を示している。

有限桁の浮動小数点数は離散的に存在するため，表現可能な数の間には隙間ができる。この隙間は指数部が大きくなるにつれて，絶対的な距離は大きくなるが相対的にはほぼ同じという特徴がある。この間隔をマシンイプシロン (machine epsilon) と呼ぶ。これにも幾つかの定義方法があるが，本書では以下のように定義する。

**定義 3.1.2 (マシンイプシロン)**
1 の次に大きい最小の正の有限桁の浮動小数点数を $1+\varepsilon_M$ と表現する時，この $\varepsilon_M$ をマシンイプシロン (machine epsilon) と呼ぶ。特に基数 $m$ で仮数部が $p$ 桁の場合，

$$\varepsilon_M = m^{-(p-1)} \tag{3.1}$$

となる。

従って，IEEE754 standard の場合のマシンイプシロンは

**単精度** $2^{-23} \approx 1.1920928955078125 \times 10^{-7}$

[1]但し，数値の形式を固定化したことによって，高速計算が可能になったという側面もある。丸め誤差は実用性を追い求めた結果の必然とも言える。

**倍精度** $2^{-52} \approx 2.22044604925031308084726333618164 \times 10^{-16}$

となる。

## 3.2 誤差の種類

近似値とは，真の値 (真値) が存在しそれに近いと考えられている数値であり，真値と近似値とのズレを誤差 (error) と呼ぶ。とかくこの世で見ることのできる数値は近似値であることが多い。数字自体に誤りはないようでも，それを導出するプロセス，または元となる式そのものに差異のある場合もある。科学技術計算において誤差が発生するプロセスををまとめたものが図 3.2 である。下線部は，コンピュータを用いた数値計算において発生する誤差である。

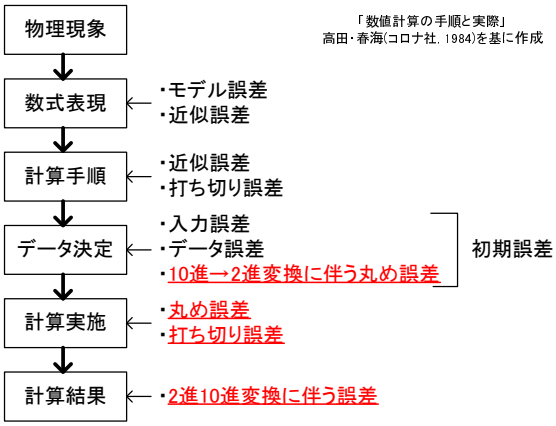


図 3.2: 数値計算における誤差

誤差の大きさを測る指標としてよく用いられものは次の二つである。

**定義 3.2.1 (絶対誤差 (absolute error)，相対誤差 (relative error))**
$a \in \mathbb{R}$ を真値，$\widetilde{a} \in \mathbb{R}$ をその近似値とする。このとき

$$E(\widetilde{a}) = |a - \widetilde{a}| \tag{3.2}$$

を $\widetilde{a}$ の絶対誤差と言う。更に

$$rE(\widetilde{a}) = \begin{cases} \left|\dfrac{a-\widetilde{a}}{a}\right| = \dfrac{E(\widetilde{a})}{|a|} & (a \neq 0) \\ |a - \widetilde{a}| = E(\widetilde{a}) & (a = 0) \end{cases} \tag{3.3}$$

を $\widetilde{a}$ の相対誤差と言う。

これはベクトル量まで拡大することが出来る (第 8 章参照)。実際に $E(\widetilde{a})$，$rE(\widetilde{a})$ を求める際には，真値 $a$ が不明であっても，$\widetilde{a}$ よりは $a$ により近い (と思われる) 別の近似値 $\widetilde{a}'$ を用いて，

$$E(\widetilde{a}) \approx |\widetilde{a}' - \widetilde{a}| \tag{3.4}$$

$$rE(\widetilde{a}) \approx \left|\frac{\widetilde{a}' - \widetilde{a}}{\widetilde{a}'}\right| \tag{3.5}$$

を (3.2) や (3.3) の代わりに用いる。

$\widetilde{a}$ が一次元量であるとき，どの桁までが真の値と一致しているかを示す量を用いると便利なことがある。これを有効桁 (significant digit) という。

**定義 3.2.2 (有効桁 (significant digit))**
$a \in \mathbb{R}$ を真値，$\widetilde{a} \in \mathbb{R}$ をその近似値とする。このとき

$$\lfloor -\log_m rE(\widetilde{a}) \rfloor \in \mathbb{N} \tag{3.6}$$

を $m$ 進法における $\widetilde{a}$ の有効桁という。$\lfloor x \rfloor$ は $x$ を越えない最大の整数を表わす。

これは，上から $\lfloor -\log_m rE(\widetilde{a}) \rfloor$ 桁は正しい数値であり，$\lfloor -\log_m rE(\widetilde{a}) \rfloor + 1$ 桁目が $\log_m rE(\widetilde{a})$ の小数部だけ怪しいとみることができる。

有限桁の浮動小数点数同士の四則演算において，絶対値が接近している数同士の加算或いは減算の絶対値がそれらよりも小さくなる時，被演算数の絶対誤差の大きさは余り変化しない分，演算結果の相対誤差が大きくなることになる。この現象を有効桁数が減少することから「桁落ち」と呼ぶ。

さて，この誤差が発生する原因は何か。[11] では実験や観測で発生する誤差を原因別に

1. 系統誤差
2. 偶発誤差
   (a) 過失誤差
   (b) 必然的偶発誤差

と分類した。以下,「・」は [11] から抜粋した文を表わす。

系統誤差とは「根本理論における間違い，機械の狂い，あるいは観測者個人の一定の癖とかの原因により，ある定まった規則的な関係で導入される誤差」をいう。これは後でいくらでも補正のきくものである。

2. の偶発誤差は「まったく偶然に不定な関係で起生する誤差」をいい，そのうち過失誤差はその名の通り，人間の不注意により発生するもので，これも努力次第で除くことができる。しかし必然的偶発誤差は,「測器の精度限界のところで必然的かつ偶発的に発生」するもので，完全に除去することは不可能である。

この除去不可能な入力データに付随する誤差を，2 進 $\longrightarrow$10 変換誤差と共に「初期誤差 (initial error)」と呼ぶことにする。

以上は問題が立てられた時点で既に混入してしまっている誤差である。数値計算そのものに起因する誤差は

1. 打ち切り誤差 (truncation error) or 理論誤差 (theoritical error)
2. 丸め誤差 (round-off error)

である。丸め誤差については次節で詳細に論じる。打ち切り誤差は連続数学を対象とするアルゴリズムに依存して決まるので，各章の解説を参照されたい。

## 3.3 丸め誤差

小数の誕生と共に丸め誤差は発生したと言ってよい。われわれの扱わねばならない数が有限桁の小数の範囲を逸脱したとき，丸めという操作は必要不可欠のものとなった。

「丸め」(round-off) とは，前述したように実数を有限桁の小数に納める操作のことである。例えば

$$\sqrt{2} = 1.41421356\cdots$$

を 7 桁で納めるには 8 桁目以降を切り捨て (chop) すればよい。即ち，

$$1.414213$$

となる。このとき

$$\begin{aligned} E(1.414213) &= |\sqrt{2} - 1.414213| \\ &= 0.00000056\cdots \end{aligned}$$

となる。この $E(1.414213)$ が丸め誤差である。

この丸めの方法は誤差が大きくなるので，あまり良い方法とは言えない。そのため，単に切り捨てるだけではなく，丸め誤差を出来る限り小さくする $m/2 - 1$ 捨 $m/2$ 入 (10 進数では四捨五入) という方法が提案された。

先の例でそのやり方を示す。今，数字を 7 桁で納めたいとする。このときは 8 桁目の数字を見る。これが

4 以下のとき $\longrightarrow$ 8 桁目以降を切り捨て

5 以上のとき $\longrightarrow$ 7 桁目に 1 を加えて，8 桁目以降を切り捨て

となるようにする。今の例だと

$$1.4142135\underline{6}\cdots \Longrightarrow 1.414214$$

となる。このときの丸め誤差は

$$|\sqrt{2} - 1.414214| = 0.00000044\cdots$$

であり，単なる切り捨て法よりも小さくなっている。

浮動小数点数の丸め誤差は，相対誤差で見るとほぼ同程度になる。つまり，精度桁もほぼ一定になる。これを数値例で見ることにする。

まず固定小数点数の場合。先程と同様，10 進 10 桁とし中心に小数点を置く。このとき 1.96933 を丸めると，

$$1.9693\boxed{3}\,5 \Longrightarrow 1.9693\boxed{4}$$

であり，丸め誤差は

$$E(1.96934) = 0.000005$$

となる。同様に，47155.094923 は

$$47155.0949\underline{\boxed{2}3} \Longrightarrow 47155.0949\boxed{2}$$
$$E(47155.09492) = 0.000003$$

である。従って相対誤差は

$$\begin{aligned} rE(1.96934) &= \frac{0.000005}{1.969335} \\ &\approx 2.539 \times 10^{-6} \\ rE(47155.09492) &= \frac{0.000003}{47155.094923} \\ &\approx 6.362 \times 10^{-11} \end{aligned}$$

となる。

これに対して，仮数部 5 桁の浮動小数点数では

| | | | 丸め誤差 |
|---|---|---|---|
| 1.969335 | $\Longrightarrow$ | 1.9693 | 0.000035 |
| 47155.094923 | $\Longrightarrow$ | 4.7155 | 0.094923 |

だから相対誤差は

$$\begin{aligned} \frac{0.000035}{1.969335} &\approx 1.777 \times 10^{-5} \\ \frac{0.094923}{47155.094923} &\approx 2.013 \times 10^{-6} \end{aligned}$$

である。固定小数点数よりもその差は少ない。浮動小数点数の場合，$m$ 進仮数部 $t$ 桁であれば丸め誤差は

$$rE(\overline{a}) \leq \frac{1}{2}m^{-(t-1)} = \frac{1}{2}\varepsilon_M \tag{3.7}$$

で抑えられる。この $\frac{1}{2}m^{-t+1}$ を丸め誤差の最小単位と呼び，$u$ と書く。つまり，仮数部の桁数が固定小数点数のそれを同じであれば，相対誤差で見る限り，浮動小数点数の方が丸め誤差は小さくなる。

1985 年に IEEE754 という浮動小数点数の規格が採択された。この規格の草案は Kahan が中心となって，Intel の技術者と共に取りまとめたものである。

この規格では 2 進数が採用されている。かつての大型コンピュータで広く使用されていた 16 進数は外された。その理由は，2 進と 16 進で仮数部のビット数を同じにした場合，後者に無駄が出る可能性があるからである。例えば，16 進で 2 桁，2 進だと 8bit の場合，仮数部が

$$\begin{aligned} 16\text{ 進} &: \quad \underline{0001}\quad 1011 \\ 2\text{ 進} &: \quad 1101\quad 1101 \end{aligned}$$

という bit パターンになることがある。この下線部 3bit 分，2 進よりも精度的に損をすることになる。これは 16 進の性格上，4bit ごとに正規化 (仮数部の最上位桁を非零にする操作) されるためである。

この規格では，前述の通り 4 つの 2 進数の丸め方が規定されている。特に RN 方式は概ね 0 捨 1 入であるが，そのやり方は単純なものとは言い難い。森口は，恐らくはこの RN 方式についてであろう，自著 [23, p.208–209] の中で次のように述べている。

> （略）著者は，まだ製造業者に問い合わせていないが，次のように想像している。それは，浮動小数点演算で，仮数部を 24 ビットに丸めるとき，**単純な 0 捨 1 入**—つまり，次のビットが 0 ならば切り捨て，1 ならば切り上げること—をやればいいものを，それをやる代わりに，恐らくは作成者の誤解に基づく **JIS の丸め方**—次のビットが 0 ならば切り捨て，1 ならば丸めた結果が偶数になるように切り捨て・繰り上げを決める—をやっているということである。（中略）ところで，JIS Z 8401（数値の丸め方）には，**次のビットが 1 であって，それが切り捨てて 1 になったものであることがわかっている場合には，それは切り上げる**旨が明記してあり，われわれが問題にしている場合について言えば，ほとんど確率 1 でそれは切り捨てた結果であるから，結局（次の 1 ビットだけ見て丸めるには）「普通の 0 捨 1 入」によるべきである。信頼されている有名会社の製品にもこんな例があるので，読者諸君には自分の使う機種について，丸め方の実験をしてたしかめておくことをお勧めする。

森口の言う「誤解に基づく ··· 丸め方」が，実は IEEE754 の RN 方式である。具体的に書けば

| 末尾桁 | その 1 つ下 | | 結果 |
|---|---|---|---|
| 0 | 0 | | 00 |
| 0 | 1 | 切り捨て $\Longrightarrow$ | 00 |
| 1 | 0 | | 10 |
| 1 | 1 | 切り上げ $\Longrightarrow$ | (1)00 |

(3.8)

となる。単純な「0 捨 1 入」であれば末尾桁の一つ下の桁のみ見て判断するから，丸めのパターンは 2 つになる。

| 末尾の 1 つ下 | | 結果 |
|---|---|---|
| 0 | 切り捨て $\Longrightarrow$ | 0 |
| 1 | 切り上げ $\Longrightarrow$ | (1)0 |

(3.9)

(3.8) のような丸め方は，確かに不自然に思える。

このような丸め方を採ったのは,「結果が整数になることが多い」からであるらしい [7, p.51]。これを偶数丸めと呼ぶ。

IEEE754 規格制定後，代表的な CPU メーカの Intel や Motorola がこの規格に準拠した浮動小数点演算のための chip—数値演算プロセッサ (NDP, Numeric Data Processor) を開発，製造した。代表的なワークステーション，パーソナルコンピュータの CPU を製造するこの二大メーカが採用した影響は大きく，今では PC 向けの全ての CPU にこの IEEE754 規格に基づいた浮動小数点演算 unit が内蔵されている[2]。これから先も IEEE754 以外の浮動小数点規格が使われることはないであろう。

## 3.4 桁落ちの例とその解決策

長々と述べてきたように，有限桁の浮動小数点演算には丸めに伴う誤差が生じる。この結果，加減算を含む計算を行うと相対誤差の極端な増大を引き起こすことがある。この相対誤差の悪化現象を桁落ち (loss of significant digits) と呼ぶ。

[2]昔，NDP がない場合は，多倍長計算と同様，ライブラリで IEEE754 standard をエミュレーションしていた。

この節では大きな桁落ちを伴う初等的な問題を 2 つ取り上げる。一つは，数値計算のテキストでは必ず取り上げられる 2 次方程式 (正式には実定数 2 次代数方程式) の解の公式における桁落ちの例，もう一つは，これも複雑系の分野では必ず取り上げられるロジスティック写像の例である。

**例題 3.4.1 (実定数 2 次代数方程式)**
実定数の 2 次代数方程式

$$ax^2 + bx + c = 0$$

を考える。一般に 4 次までの代数方程式は解の公式が存在する。この場合はよく知られているように，重複も含めた 2 つの解 $x_1$，$x_2$ は

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{3.10}$$

$$x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{3.11}$$

である。

今，$a = 1.01, b = 2718281, c = 0.01$ という係数において，この解の公式 (3.10), (3.11) を素直に適用し，IEEE754 倍精度で計算すると

$$\widetilde{x_1} = -2.69136732673266949 \times 10^{6}$$
$$\widetilde{x_2} = -3.68840623610090498 \times 10^{-9}$$

となる。$x_2$ の真の値は

$$x_2 = -3.67879553291216532392049 9\cdots \times 10^{-9}$$

であるから，一致している桁は上位二桁止まりである。

2 次方程式の場合，$|b| >> |ac|$ の時，$|b| \approx \sqrt{b^2 - 4ac}$ となり，$x_1$ もしくは $x_2$ のどちらかの絶対値が極端に小さくなる。この例では $a, c$ に 10 進→ 2 進変換の際に丸め誤差が混入し，更に $\sqrt{b^2 - 4ac}$ の計算で丸め誤差が発生し，分子の加減算の結果，桁落ちが起こって相対誤差が極端に悪化するのである。

**問題 3.4.1**
次の 2 次方程式の解を 10 進 5 桁の浮動小数点演算で求めよ。

$$3x^2 - 350x + 2 = 0$$

**例題 3.4.2 (ロジスティック写像)**
桁落ちする計算の例として，ロジスティック写像

$$f(x) = 4x(1 - x) \tag{3.12}$$

を使った，漸化式

$$x_{i+1} := f(x_i) \tag{3.13}$$

によって導出される実数列 $\{x_i\}_{i=1}^{100}$ を，IEEE754 単精度及び倍精度で計算した結果を示す。プログラムは例えば C 言語で記述すると図 3.3 のようになる。初期値は全て $x_0 = 0.7501$ と指定した。この数列の計算は断続的な桁落ちが無限に発生する悪条件な計算の一例である。

桁落ちが発生する理由は，$0 < x_i < 1$ であれば，$1 - x_i$ のところで，必ず少しずつではあるが桁落ちが起こり，この関数の性質から再び桁落ちする領域に $x_{i+1}$ が戻ってくるからである。従って，断続的に桁落ちした結果，$i$ が大きくなるにつれて，精度が悪化していくことになる。

```
#include <stdio.h>

main()
{
    int i;
    float x[102]; /* 単精度 */

    x[0] = 0.7501;

    for(i = 0; i <= 100; i++)
    {
        x[i+1] = 4 * x[i] * (1 - x[i]);
        if(i%10 == 0)
         printf("%5d %25.17e\n", i, x[i]);
    }
}
```

図 3.3: ロジスティック写像を計算する C プログラム

以下は IEEE754 単精度で配列を宣言したときの結果である。

```
  0   7.50100016593933105e-01
 10   8.44516813755035400e-01
 20   1.22903920710086823e-01
 30   4.97778177261352539e-01
 40   5.81643998622894287e-01
 50   5.58012068271636963e-01
 60   2.28748228400945663e-02
 70   9.98548150062561035e-01
 80   9.42180097103118896e-01
 90   2.12938979268074036e-01
100   7.56864011287689209e-01
```

以下は IEEE754 倍精度で配列を宣言したときの結果である。

```
  0   7.50099999999999989e-01
 10   8.44959536022011990e-01
 20   1.42939724528399537e-01
 30   8.54296020314155413e-01
 40   7.74995884542777125e-01
```

```
 50   7.95132827423636751e-02
 60   2.73872762849587226e-01
 70   9.97021556611396687e-01
 80   3.52785425069070790e-01
 90   6.35558111134618908e-01
100   5.15390006286616020e-01
```

両者見比べると，$x_{20}$ ではもう最初の一桁程度しか精度がないことがわかる。よって，それ以降の計算はどうも怪しい，と勘ぐらねばならない。事実，$x_{30}$ 以降の両者の値は全く異なっている。

桁落ちの厄介な所は，正確な桁が加減算により失われることにある。この喪失はその桁が正確であるから起こるのであって，**起きるべくして起きている**のである。従って，このように桁落ちを起こす問題を解決するには，桁落ちが発生しない数値で計算を行うようにするか，桁落ちが発生しても困らないぐらい浮動小数点数の桁数を増やす他ない[3]。

前者に比べて後者の解決方法は安易である。詳細は次章に譲るが，任意の桁数を設定できるソフトウェアは数多く存在するので，それらを使えば計算の実行そのものはそれ程面倒なものではない。前者を行うためには，問題の設定そのものを変えてしまうか，アルゴリズムを別なものにして，桁落ちを起こさない数値を出現させるしかなく，これは理論的な背景をよく知っていなければ実行できないものである。数値計算の研究としては大変に興味深いものであるし，ハードウェアで直接処理できる範囲の浮動小数点数だけで処理できればそれに越したことはなく，前者の道を取ることが数値計算の「王道」であることは事実であるが，それは誰にでも実行できるものではない。それに反して，後者の方法は，当然計算速度は落ちる上，必要となる記憶容量も多くなるが，安易に実行できるため「ユーザに優しい」方法であると言える。

ここに取り上げた実例は初等的なものであるため，アルゴリズムを改変することが可能である。後者の手法を使った解決策は次章で示すことにする。

**例題 3.4.3 (2 次方程式の改良)**
解の公式 (3.10), (3.11) を次のように改変する。まず

$$x_{1'} = \frac{-b - \mathrm{sign}(b)\sqrt{b^2 - 4ac}}{2a} \tag{3.14}$$

として一方の解 $x_{1'}$ を求める。ここで sign($b$) は $b < 0$ の時 $-1$ を，それ以外の時は 1 を返す関数である。これを計算した上でもう片方の解 $x_{2'}$ を

$$x_{2'} = \frac{c}{ax_{1'}} \tag{3.15}$$

として求める。

こうすることで，桁落ちを引き起こす分子の計算を避けることが可能となる。例題 3.4.1 で示した係数の例を計算してみると

$$\begin{aligned} x_1 &= -2.69136732673266949 \times 10^{6} \\ x_2 &= -\underline{3.67879553291216534} \times 10^{-9} \end{aligned}$$

[3]問題そのものに初期誤差が含まれている場合は，どうやってもその影響を逃れることはできない。ここで議論しているのは，初期誤差ゼロ，即ち，2 次方程式の係数やロジスティック写像の初期値に誤差は含まれないか，誤差は任意に調整できる場合に限られる。

となり，ほぼ末尾桁まで精度を回復させていることが分かる。

**問題 3.4.2**
問題 3.4.1 の 2 次方程式を改良された上記の方法で求めよ。

## 3.5 丸め誤差解析の手法

コンピュータでの計算における丸め誤差解析を最初に手掛けたのは，von Neumann と Goldstain であった。私はこの論文をきちんと読んでいないので，参考文献には挙げていない。

これをもとにして Wilkinson が具体的な問題に，自身の体験を踏まえて浮動小数点数の丸め誤差解析を行った。この Wilkinson のとった丸め誤差解析の手法は，今も利用されている。

今それ [41] を見ると，その頃しか使用されていなかった浮動小数点数形式 (例えば Block-Floating-Point Number など) の誤差解析に結構なページ数が割かれており，見るべきところはそれ程多くなはい。しかし，そこにある代数方程式や連立一次方程式の数値例，誤差解析は今もなお古びてはいない，貴重なものである。

Wilkinson の解析法は，計算で発生する丸め誤差をすべて入力誤差として取り扱い，その摂動量 (perturbation) を見積もることで，誤差解析をしようというものである。

この違いを具体例で示す。

今

$$ax = b \ \ (a \neq 0,\ a, b \in \mathbb{R})$$

という一次方程式の場合を考える。

普通は，

$$x \longleftarrow \frac{b}{a}$$

という計算で，

$$E(x) \leq \frac{b}{a}u$$

なる丸め誤差が入ると考える。これを前進誤差解析という。ここで $u$ は前述の丸め誤差の最小単位 $u = 1/2m^{-(t-1)}$ である。

Wilkinson はこれを次のように考えた。演算では誤差は入らないものとし，$E(x)$ は最初から $a$ または $b$ に混入しているとする。この場合は $b$ に含めた方が都合がよい。

こうして最初の方程式を

$$a\widetilde{x} = \widetilde{b} \ \ (\text{ここで}\widetilde{b} = b + E(b),\ \widetilde{x} = x + E(x))$$

とし，$E(x) \leq \frac{b}{a}u$ を満たすような，$E(b)$ の範囲を

$$\begin{aligned} x + E(x) &= \frac{b + E(b)}{a} \\ &= \frac{b}{a} + \frac{E(b)}{a} \end{aligned}$$

から得た，

$$|E(b)| = |E(x)a|$$

から $|E(x)| \leq \frac{b}{a}u$ を使って

$$|E(b)| \leq bu$$

を得る。

両者とも結果は同じものだが，後者の方は，式が複雑になったときに見通しがつけやすい。

欠点は共通している。どちらも丸め誤差の「限界」を求めるため，実際の量よりも過大に見積もりがちになることである。

一次方程式の例ではわかりづらいから，別の問題を挙げる。

$$\begin{aligned} y &= \sum_{i=1}^{m} x_i \\ x_i &\in \mathbb{R} \end{aligned}$$

なる計算を考える。丸め誤差の最小単位は前と同様に $u$ とし，各 $x_i$ には誤差はないものとする。

この計算を

$$\begin{aligned} y &\longleftarrow x_1 \\ y &\longleftarrow y + x_2 \\ y &\longleftarrow y + x_3 \\ &\vdots \\ y &\longleftarrow y + x_n \end{aligned}$$

という順序で行う。こうして得る $y$ に含まれる誤差 $E(y)$ の限界を求める。

各段階の計算

$$y \longleftarrow (y + E(y)_{i-1}) + x_i$$

で発生する丸め誤差 $E(y)_i$ は

$$\begin{aligned} |E(y)_i| &\leq |(y + E(y)_{i-1}) + x_i|u \\ &\leq \sum_{j=1}^{i} |x_j|u + \sum_{j=1}^{i-1} |x_j|u^2 + |E(y)_{i-2}|u^2 \\ &\cdots \end{aligned}$$

であるから，最終的には

$$\begin{aligned} |E(y)| \leq \quad & |x_1|(u + u^2 + \cdots + u^{n-1} + u^n) \\ + \quad & |x_2|(u + u^2 + \cdots + u^{n-1}) \\ & \vdots \\ + \quad & |x_n|u \end{aligned}$$

から

$$|E(y)| \leq \max_i |x_i| \sum_{i=1}^{n} (n - i + 1)u^i$$

を得る。通常 $u \ll 1$ であるから，

$$|E(y)| \leq (n \max_i |x_i|)u$$

と考えてよい。

この評価の限界まで丸め誤差が大きくなることは滅多にない。その理由は 3 つある。

1. 演算で丸めが必ず起こるわけではない。演算結果が仮数部の桁数内に収まれば丸め誤差は発生しない。
2. ここでは丸め誤差を絶対値で評価しているが，実際には符号を持つ。正負の丸め誤差が打ち消しあって，蓄積量はそれほど増えない。
3. 各 $x_i$ の order 差が大きい場合，絶対値最大の成分の加算の際の丸めのみで通常は評価してよい。

通常は 2. による効果が大きい。この弱点を克服するため，統計的な丸め誤差評価法が提案されている。私が最初にこれを見たのは Henrici の本であったが，彼の発案ではないらしい。彼の書 [8] では，Brouwer が最初であるとしている。

この確率的誤差評価法は，発生する丸め誤差を一様乱数とみなし，その蓄積量を標準偏差で量ることで，誤差限界 (error bound) を小さくすることができる。このやり方は現実の問題によく合うが，その数学的な扱いに無理があると言われている。つまり，丸め方が決まっていれば，丸め誤差の出方も規則性があり，乱数とみなすことはできないと言うのである。しかし丸め方は規則的であるが，扱われる数値は千差万別でこれは乱数とみなしてよい。従って，そこから出る丸め誤差も乱数と思って差し支えないだろう。個々の問題に対するものさしとしては特に理論的な無理はないと思われる。

先の前進，後退誤差解析との違いは

**完璧を求めるか否か**

ということに尽きる。確率的評価法は現実の問題で適正な丸め誤差の order を期待できる反面，そこで得られる評価はあくまで確率的なものでしかない。前進・後退誤差解析法は 100%の安全性を確保できるが，実際の誤差の order からは遠く離れてしまうことが多い。どちらを選択するかは利用者の価値観で決めるべきである。

## 3.6 数値計算における誤差解析

ここでは私見に基づく，数値計算における誤差解析の方法を簡単に述べる。当然のことと思われることばかりだろうが，以下述べるようなことをきちんと明記したものを私は読んだことがない。そこで，一節設けることにした。

数値計算で求めることのできる数値は，前節でも述べたように，大抵は真値に対する近似値でしかない。最悪の場合，有効桁を全く得ることができない。

従って，計算する側が必要とする精度を，求めた近似値が持っているがどうかをチェックすることは非常に重要なことだ。しかし，これは言うに易しく，行うに難いことである。上手にやらないと，いや，うまくやっても結構労力のいる作業である。できるなら最小のエネルギーで行いたい。

そのためには，先人の得た結果を，次のように分割された各ステップに適用するのがよい。

A. 初期誤差の摂動解析

B. 計算過程の誤差解析

(a) 理論誤差解析

(b) 丸め誤差解析

まず A. で，初期誤差による摂動の限界を求める。ここで，計算での誤差が全くないと仮定したとき，最大でどのぐらいの精度が見込めるか，あるいは最悪の場合はどの程度か，を調べる。一般に**悪条件 (ill-conditioned)** といわれる問題は，初期誤差による影響だけで，一桁も出ないことがある。やっても無駄な計算をしないだけでなく，実際の計算で必要となる精度の見積もりも合わせて行える。

まとめると，この段階では

- 最大でどの程度の精度が得られるか
- 必要とする精度を得るには，計算での浮動小数点数の仮数部の桁数をどのぐらいにしたら良いのか

を調べる。

計算における浮動小数点数の精度は，次の理論誤差解析，丸め誤差解析での結果を合わせて判断する。

理論誤差解析では，無限回の操作を必要とする解法，例えば方程式の根を求めたり，微分方程式の近似解を計算する等の場合に，反復回数をどの程度増やすことで，もしくは刻み幅を小さくすることで，どの程度の理論誤差（打ち切り誤差）を減らすことができるかを調べる。これは，大雑把に言って，計算回数に制限を設けなければいくらでも小さくできるものである。そこで必要とされている精度を得るためにはどの程度の計算回数が要求されているかを調べる。但し，直接法 (第 7 章参照) ように有限回の代数的操作によって解が得られるアルゴリズムでは理論誤差は考えない。

次の丸め誤差解析では，前進・後退誤差解析もしくは確率的評価法による誤差限界を得，その order が初期誤差のそれよりも小さく，あるいは同程度にする，その浮動小数点数の精度を見積もる。丸め誤差評価は，いずれも丸め誤差の最小単位 $u$ をファクターとして持つ。これは浮動小数点数の桁数を増やすことで小さくできる (第 4 章参照)。

もう一度，以上の操作をまとめると次のようになる。

1. 初期誤差による誤差限界を求める。
2. 理論誤差解析によって，必要な反復回数，刻み幅を求める。
3. 丸め誤差限界が要求精度以内となるように，浮動小数点数の精度を決める。

このように，誤差解析を幾つかの段階に分離することで，第三者への説明はかなり楽になる。例えば，

> 初期誤差がこれだけ入っていますから，どう頑張ってもこれが精度の限界です。これだけ桁数を取っていますから，丸め誤差の影響は初期誤差の誤差の order よりも下になっていますし，反復回数も増やしていますから，打ち切り誤差の影響もありません。もし，これ以上の精度を望まれるのであれば，初期誤差を減らして，データそれ自体の精度を上げるしかありません。

と言えば，かなりわかりやすいし，説得力があると思う。

しかし，このやり方がすべての現象を解明できるわけではない。平野 [30]・永坂 [26] の言う「誤差の cancel」がその良い例である。

これは，各々の数値に混入した，独立でない初期誤差あるいは丸め誤差が，偶然ではなく**必然的**に打ち消されてしまう現象をいう。この現象はアルゴリズムに強く依存する。同じ種類の問題に対するアルゴリズムでも，cancellation が起きるものもあれば，起きないものもある。従って，この現象の説明には，今言ったような方法は向いていない。

ただ，誤差の cancel が誤差評価に及ぼす影響は，それを過大評価にすることである。従って，誤差の cancellation がその解法において重要な役割を持つようになるとき以外は，一般に，この現象を無視することで経済的に損をすることはあっても，評価の信頼性を落とすことはないと言える。

## 演習問題

1. 1/3 を 10 進 5 桁の浮動小数点数に変換した際に発生する相対丸め誤差を求めよ。
2. 10 進 5 桁の浮動小数点数を用いて計算した $\widetilde{a} \approx \pi + e$ と，10 進 10 桁で計算した $\widetilde{a}' \approx \pi + e$ をそれぞれ求め，これを真値の代わりに用いて $E(\widetilde{a})$ と $rE(\widetilde{a})$ の近似値をそれぞれ求めよ。
3. 2 次方程式 $-\sqrt{2}x^2 - 1234x - \pi = 0$ の解を 10 進 5 桁の浮動小数点演算を用いて出来うる限り正確に求めよ。
4. 例 3.4.2 において，IEEE754 単精度で計算した値に含まれる相対丸め誤差を全て計算し，縦軸に相対丸め誤差を，横軸に項数をプロットした折れ線グラフを描け。
5. 次の漸化式

   $$a_0 = 1,\ a_{2k} = -\frac{n}{2k}\sum_{j=1}^{k}\frac{1}{2j+1}a_{2(k-j)}$$

   で得られる数列 $\{a_0, a_1, a_2, a_3, ..., a_n\}$ を考える。但し，$n$ は偶数の定数，$k$ が奇数の時は $a_k = 0$ とする。$n = 256$ の時，この数列の偶数番目の値を実際に計算し，どの程度桁落ちするかを確認せよ (小野・藤野の問題)。
6. ロジスティック写像の別計算方法を考える。以下の説明は下條 [33, pp.39–43] による。

   $x_i$ を

   $$x_i = \sin^2 \pi\theta_i \tag{3.16}$$

   とし，変数 $\theta_i$ へ置き換える。従って漸化式 (3.13) は

   $$\sin^2 \pi\theta_{i+1} := 4\sin^2 \pi\theta_i(1 - \sin^2 \pi\theta_i)$$

より，右辺は $4\sin^2\pi\theta_i\cos^2\pi\theta_i = \sin^2 2\pi\theta_i$ となる。よって，これは

$$\theta_{i+1} := 2\theta_i$$

という漸化式に帰着される。これは単なる等比数列であり，$\theta_i$ は三角関数の引数であるから，

$$\theta_i = 2^i\theta_0 \pmod 1$$

として良い。

初期値 $x_0 = 0.7501$ であるから，$\theta_0 = \pm 0.27929998192256631489\cdots$ となる。これを用いれば (3.16) で任意の $x_i$ が計算できる。

しかしこの方法でも精度を改善することは困難である。その理由を考えよ。

## 参考図書

**数値計算の常識**
伊里正夫・藤野和建
共立出版
1985 年

数値計算全般を，具体例をふんだんに盛り込んで語っている希有な一冊。説明はそれなりである故，初学者は一通り数値計算について勉強した後に読んだ方が良いだろう。

**Numerical Computing with IEEE Floating Point Arithmetic**
Michael L. Overton
SIAM
2001 年

100 ページ足らずの薄い本だが，IEEE754 standard に限らず，有限桁の浮動小数点数を扱う際に必要となる知識は大体記述してある。普通はこの程度ぐらい知っておけば十分であろう。