4 精度と誤差

数値計算を効率よく行う事は重要であるが、その際に決して失ってはならないのが計算の信頼性である[1]。整数 1 は実数では $1.\dot{0}$ と 0 が無限に続く数であるが、計算機のリソース (CPU やメモリ) は有限であるので、当然無限を扱える訳ではない。従って、一般に数値計算の結果は有限な精度しか持ち合わせない (意味があるのは最初の数桁)、逆に言うと**常に誤差を含んでいる**事に注意すべきである。

以下では、数値計算におけるエラー、誤差について考えてみる。

4.1 エラー

数値計算を行う際に最初に明確にしておくべき事は、「何をどの精度で得たいか (計算したいか)」である。数値計算におけるエラーや誤差は、この目標とする精度と、数値計算が含む誤差両方を考慮して議論すべき事柄である。

エラー (間違い) には色々なものがある。例えばプログラム中に文法上のエラーがあれば、コンパイラはエラーを表示する[2]。ここでは、実行ファイルが計算した結果について考える。文法上は正しくても与えた数値が不適切で、不自然な結果を与える場合がある。このような場合は誰でもすぐに間違いと気が付くため、あまり深刻に考える必要はない[3]。

問題は、一見最もらしい値が得られた場合である。例えば、ある物理量を計算して、非常に小さいが有限な値が得られたとする。この「小さいが有限」である事に本当に意味があるか否かは慎重に判断すべきである。もしかしたら計算の前提にした理論の枠組上、厳密に 0 になるべき値なのに、数値計算の誤差 (の積み重ね) により有限な値になってしまったということもあり得るからである。

多くの場合、数値計算の誤差は目標とする精度に比べ十分小さいため、数値計算の誤差に神経質になる必要は無い。しかしながら、ニュートリノの質量の様に、0 か有限な値であるかが物理の根源 (本質) に関わる事も少なくなく、このような場合は慎重に吟味すべきである。また、数値計算の結果が 7 桁あったからといって、レポートにそのまま 7 桁記載しても意味が無い (最後の数桁は数値計算の誤差を含む)。

4.2 誤差

次に誤差の問題である。前述の通り、これは目標とする精度と切り離して考える事ができない。単に大きさの桁 (order) のみを知りたい場合もあれば、7 桁以上の高精度で知りたい場合もあるであろう。以下で、数値計算途上で遭遇する色々な誤差について調べてみる。

4.2.1 まるめ誤差

数値を四捨五入したり切り捨てたりするために最後の桁に現われる誤差を**丸め誤差**という。

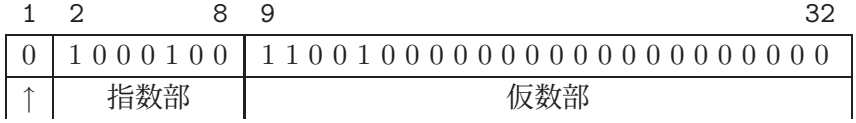
---

[1] 逆もある程度真であり、計算の信頼性を追い求めるあまり、極端に非効率な数値計算にならないように注意すべきである。

[2] 当然、オブジェクトや実行ファイルを生成する作業は行われない。

[3] すぐに気が付く為には物理の素養が必要ではあるが。

図 4.1: 10 進数の値 12.5 の 32 ビット表示。



計算機は数値を、電気信号のオン・オフの組合せ、すなわち 2 進数として記憶しており、1 つの数値に対して決められたビット数 (組合せ数) が割り当てられている。多くの計算機では単精度変数[4]に 32 ビットを割り当てている。つまり、数値 x を、浮動小数点表示を採用して、

$$x = \pm M \times 10^e \tag{4.1}$$

と、仮数部 M・指数部 e・全体の符号の 3 つの要素に分解し、合計 32 ビットで記憶させている。

例を示す。10 進数の数値 12.5 を記憶させる場合、

$$\begin{aligned} 12.5 = 8 + 4 + 0.5 &= 2^3 + 2^2 + 2^{-1} \\ &= (1100.1) = (0.11001) \times 2^4 \end{aligned} \tag{4.2}$$

である (カッコ内は 2 進数表記を表す) ので、図 4.1 のような 32 ビット表示になる。なお指数部はバイアス表示により、$e + 2^{7-1}$ を表示する (例の場合は、$4 + 2^{7-1} = 2^6 + 2^2$)。

このように仮数部のビット数に制限があるため、たとえば $1.\dot{0}$ のような数値を正確に記憶する事は出来ず、最後のビットは四捨五入されたり切り捨てたりせざるを得ない。このようにして発生する誤差が**丸め誤差**である。

上記のように仮数部に 24 ビットを当てると精度は 10 進数で約 7.2 桁となる。単精度変数の倍の精度で扱う変数を倍精度変数というが、この場合の仮数部は 32 ビット増え 56 ビットとなり精度は 10 進数で約 16.8 桁にもなる。

これらの精度自身は、我々が普通欲しい精度を一般に上回っている。しかい、計算の規模が大きくなり繰り返し回数が増加すると、この丸め誤差もどんどん成長 (増加) する場合があるので注意が必要である。

4.2.2 桁落ち

有効数字をたくさんとっても、極めて近い数値の差を計算すると、その結果の精度は極端に小さくなる。例えば以下のプログラムを考える。

```
      program keta_ochi
      implicit none
c
      real   a,b,x,y
c
      a = 1.0e4
```

[4]FORTRAN 77 の場合、整数であれば `integer` または `integer*4` で宣言された変数、実数であれば `real` または `real*4` で宣言された変数。

```
b = 1.0e-4
x = a+b
y = a-b
write(*,*) x-y
stop
end
```

FORTRAN の文法の説明は次章以降に譲るが、この例では、変数 a, b, x, y は全て単精度変数 (32 ビット表示) であり、

1. a に $1,0 \times 10^4$ を代入
2. b に 1.0×10^{-4} を代入
3. x に $a+b$ を代入
4. y に $a-b$ を代入
5. $x-y$ を表示

という処理を行っている。表示される結果は、

$$x - y = 2b = 2.0 \times 10^{-4}$$

となる事が期待されるが、このプログラムを、

```
/home/teacher/z6wt01in/SAMPLE/keta_ochi.f
```

から各人のディレクトリにコピーして、

```
% frt -o keta_ochi keta_ochi.f
```

として、実行ファイル `keta_ochi` を作成して実行すると、

```
% keta_ochi↵
 0.000000000e+00
```

という答えが返ってくる。これは処理の 3. 及び 4. の段階で、b の加減を行った事が桁落ちにより反映していない事による (32 ビット表記により各人確認せよ)。

このような例は誰でも問題に気が付くであろうが、複雑な (手数のかかる) 数値計算を行う場合、知らない (気が付かない) 内に極めて近い数値の差を計算しており計算精度を著しく落してしまう事があるので注意が必要である[5]。

このような桁落ちの問題は、数値計算のアルゴリズムに工夫する事で避ける事が出来る場合が多い。例えば次のような 2 つの式を考える。

$$\frac{1}{\sqrt{a+b}-\sqrt{a}}, \qquad \frac{\sqrt{a+b}+\sqrt{a}}{b}$$

[5] $x-y$ で割算を行う行があったりすると、0 での除算でエラーになったりする。

$a, b > 0$ の場合、これら 2 つの式が等価である事は明らかである。しかしながら計算機を用いた数値計算では $a \gg b$ の場合 (例えば先程の例の、$a = 1.0 \times 10^4$,$b = 1.0 \times 10^{-4}$) の場合、どちらの式で計算するかで結果が異なることがある。

演習
前述の 2 つの式を実際に計算するプログラム、

```
/home/teacher/z6wt01in/SAMPLE/keta␣chi2.f
```

に置くので、各人で実行してみよ。何が起こるであろうか。a や b の値を変えて実行してみよ。

4.2.3 打ち切り誤差

平方根、円周率などの無理数を数値で表す場合、3.14 とか 3.141593 などと必ず途中で打ち切るので誤差が生じる。この誤差が目標とする精度に比べ十分に小さい場合でも、前述の丸め誤差により、計算の繰り返しによりその影響が増大し問題となる場合がある。

例えば平方根を求める操作と二乗を求める操作を考える。計算機で単精度変数の 2 の平方根を求めると、`1.41421354` と求められ最後の桁に丸め誤差が生じている事が分かる。これを二乗すると `1.99999988` となり元の 2 には戻らない。平方根を求める操作を 20 回繰り返すと、`1.00000060` となり、これを 20 回二乗すると `1.86813200` にしかならない。この値は元の 2 からはかなり離れており、繰り返しにより丸め誤差の影響が増大した事が分かる。さらに平方根を求める操作を 23 回繰り返すと、`1.00000000` となり、これを何回二乗しても `1.00000000` にしかならない。

演習
プログラムがある程度できる者は、上述の現象を確かめてみよ。そうでない者も、関数電卓を用いて試して見るとよい。**計算機の結果を無批判に受け入れる事の危険性が実感できるはずである。**

4.3 精緻化の限界

次に、より精密に計算したために (正確には計算しようとしたために)、かえって正解との一致が悪くなる例を示す。定積分、

$$I = \int_1^2 \frac{1}{x^2} dx \tag{4.3}$$

を考える。正解は $I = 1/2$ である。この積分を、計算機を用いて、数値計算で求める事を考える。

4.3.1 区分求積法

数値積分については後程詳しく説明するが、ここでは最も簡単な区分求積法により計算する。問題の定積分は、$f(x) = 1/x^2$、x 軸、$x = 1$、$x = 2$ の計 2 つの線で囲まれる部分の面積を求めることに相当する。区分求積法では、$x = 1$ と $x = 2$ を n 等分して、各ブロックを長方形とみなして面積の近似計算を行う。均等に n 等分した場合、横幅 h は $1/n$ であり、長方形の縦の長さは、x の大きい方の辺の長さを採れば順に、

$$f(1+h), f(1+2h), f(1+3h), \cdots, f(1+nh)$$

表 4.1: 区分求積法による、単精度での計算結果。

n	数値積分値	0.5 との差
10	0.4639551	−0.0360449
100	0.4962645	−0.0037355
1000	0.4996252	−0.0003748
10000	0.4999619	−0.0000381
100000	0.4999943	−0.0000057
1000000	0.4997711	−0.0002289
10000000	0.5491630	0.0491630

となり、積分 I は、

$$I \approx \{f(1+h) + f(1+2h) + f(1+3h) + \cdots + f(1+nh)\}h \tag{4.4}$$

で与えられる。

一般化すると、区分求積法では積分を以下の式で求める。

$$\int_a^b f(x)dx \approx \{f(a+h) + f(a+2h) + f(a+3h) + \cdots + f(a+nh)\}h \tag{4.5}$$

ただし

$$h = \frac{b-a}{n} \tag{4.6}$$

である。積分の定義 (考え方) から、n が大きいほど、つまり h が小さいほど良い近似になると期待される。

4.3.2 単精度での計算

例題の場合、区分求積法では、

$$I = \sum_{i=1}^{n} \frac{1}{(1+ih)^2}h \tag{4.7}$$

を計算することになる (ただし、$h = 1/n$)。単精度変数でいろいろな n の値について計算した例を表 4.1 に示す。

n を大きくしていくと、計算結果はいったんは正解の 0.5 に近付いていくが、$n = 100000$ を越えるとかえって正解からずれてしまう。これは計算機が記憶できる実数の有効桁数に厳しい制限がある事の反映であり、精密にした (n を大きくし、h を小さくした) つもりが、計算の繰り返しが飛躍的に多くなり、結果丸め誤差が蓄積して正解からずれてしまったのである。

4.3.3 たった 1 回の計算で満足しない

では、

$$\int_1^2 \frac{1}{x^2}dx = 0.5491630 \tag{4.8}$$

表 4.2: 区分求積法による、倍精度での計算結果。

n	数値積分値	0.5 との差
10	0.4639551275	−0.0360448725
100	0.4962645830	−0.0037354170
1000	0.4996251458	−0.0003748542
10000	0.4999625015	−0.0000374985
100000	0.4999962500	−0.0000037500
1000000	0.4999996250	−0.0000003750
10000000	0.4999999625	−0.0000000375
100000000	0.4999999963	−0.0000000037

などといった誤った結論を出さないようにするにはどうすればよいであろうか?

表 4.1 を見れば分かるように、いろいろな n に対して複数回計算した結果があれば、計算結果が異常か否かについてある程度の情報を得る事ができる。逆に言えば、ある n に対する計算結果のみがあっても、その結果の妥当性について議論する事は一般には困難である。

「数値計算の常識」(伊理正夫・藤野和建著) にかかれているが、ただ 1 回の計算をしただけでは結果の正しさについてほとんど何もわからないということ、そして、系統的に計算方法を変えて2回以上計算すると非常に有用な情報が得られるということは、かなりの一般性をもって言える。

今日においては計算機の速度は劇的に向上しており、多くの場合複数回の計算を行う事はそれほどの手間を要しない。1 回の計算で満足する事なく、計算の確からしさを確認するようにして欲しい。

4.3.4 倍精度での計算

その他の (安易な) 解決策は、変数の精度を単精度から倍精度にし、有効桁数を 10 桁近く増やして計算を行う事である。先ほどの例で、倍精度で計算した結果を表 4.2 に示す。この場合は、$n = 10^7$ や 10^8 とした場合でも期待通りの計算をしてくれている。倍精度の計算には、正確さと引き替えに計算時間が増加するというデメリットが存在する。何でも倍精度で計算するというのは勧められない (倍精度で計算する事は正しさの証明にはならない) が、今日の高速な計算機においては計算時間の増加は問題とならない事も多いので、状況に応じて倍精度での計算を行うとよい。