Numerical Recipes in C

「ニューメリカルレシピ・イン・シー,

C言語による数値計算のレシピ」

W.H.Press他著, 技術評論社1993.

1.3節より

1. 別紙の記述を読み,次の項目について内容をまとめよ.(25)

(a) 数値計算の精度を制約するデータ形式とその特徴は何か.

(b) 丸め誤差とは何か.

(c) 打ち切り誤差とは何か.

(d) 安定性とは何か.

(2006年度期末試験)

1. 別紙の記述に関して,以下からひとつを選んで半ページ程度で答えよ.(25)

(a)丸め誤差について知るところを述べよ.

(b)「打ち切り誤差はすべてプログラマの制御下にある」とある.その意味を解説せよ.

(c)漸化式 (1.3.4) の不安定性を具体的にしめせ.

(2005年度期末試験)

1. 誤差に関する別紙の記述を読み,半ページでまとめよ.(20)

(2004年度期末試験)

1.3 誤差，正確さ，安定性

本書は数値解析の予備知識を仮定していないが，いくつかの基本概念については共通の理解を必要とする。この節ではこれら基本概念を簡単に定義しておく。

コンピュータは無限の精度で数値を蓄えるわけにはいかない。コンピュータの扱える数値は近似値で，これは一定の個数の**ビット** (*bit*) すなわち 0 か 1 かの桁 (binary digit)，あるいはビットを 8 個並べた**バイト** (*byte*) で表される。ほとんどのコンピュータには何種類かの数値などの表し方，すなわち**表現** (*representation*) あるいは**データ型** (*data type*) というものがあり，プログラマが選べるようになっている。データ型が違えば，使用す

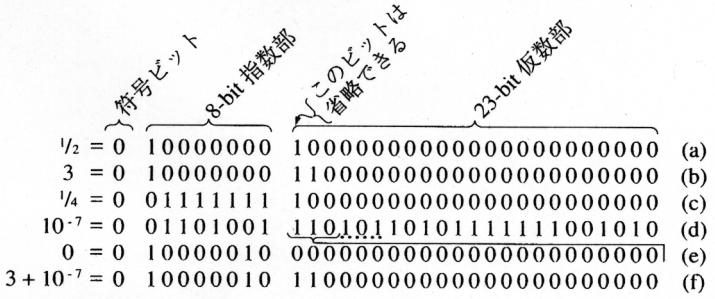

図 1.3.1：典型的な 32 ビット（4 バイト）形式の浮動小数点表現。(a) 数値 1/2（指数部の下駄に注目されたい），(b) 数値 3，(c) 数値 1/4，(d) 数値 10^{-7}，これは厳密には表せないので近似値，(e) 同じ数値 10^{-7} を，数値 3 と同じ指数部になるようにシフトしたため，有効数字はすべて失われ，ゼロになってしまった。2 数を加算する際には，このように指数部を同じにする。(f) 和 $3+10^{-7}$，これはこの精度では 3 に等しい。10^{-7} はそれ自身では十分正確に表せても，ずっと大きい数に加える際にはその正確さを保てない。

るビット数すなわち**語長** (*wordlength*) も違うかもしれないが，もっと基本的な点，すなわち `int` や `long` のような**固定小数点** (*fixed point*) 形式であるか，それとも `float` や `double` のような**浮動小数点** (*floating point*) 形式であるかという点でも異なりうる。

整数表現の数値は厳密である。整数表現の数値間の算術演算もまた厳密である。ただし，(i) 答は（通常，符号付きの）整数の表現可能範囲に入っていなければならない。また，(ii) 整数どうしの除算の商は小数部分を切り捨てて整数にされてしまう。

浮動小数点表現では，数値は内部的には符号ビット s（正負の区別を表す），指数 e（整数），仮数 M（正の整数）で表される。この表す数値は

$$s \times M \times B^{e-E} \tag{1.3.1}$$

である。ここで B は基数（基底）で，通常 $B=2$ であるが，$B=16$ のこともある。また，E は指数の**下駄** (*bias*) と呼ばれ，機械・表現形式に固有の定数である。図 1.3.1 に例を示す。

浮動小数点表現では，ビットの並びが異なっても同じ数値を表すことがありうる。例えば $B=2$ で，仮数部（浮動小数点表示の）の左側（高位の桁）に 0 のビットがある場合は，仮数部全体を左にシフトして（つまり 2 の累乗を乗算して），その分だけ指数部（浮動小数点表示の）を減らしても，同じ数値を表す。この操作により仮数部の最も左側に 0 のビットが来ないようにすることを**正規化** (*normalization*) という。ほとんどのコンピュータは，結果を常に正規化する。正規化すれば仮数部に無駄がなくなり，精度が上がる。正規化により $B=2$ では仮数部の最も左側のビットは常に 1 になるので，ほとんどのコンピュータではこのビットを省略し，1 ビット分だけ精度を稼いでいる。

浮動小数点表現の数値間の算術演算は一般に厳密ではない。たとえ被演算数が厳密に表現できても（つまり式 1.3.1 の形で正確に表せても），この事情に変わりはない。例えば，指数部が異なる 2 個の浮動小数点数を加える際には，大きい方の指数部に合わせるために，指数部が小さい方の数の仮数部を右にシフトし（2 で割り），その分だけ指数部を増す。こ

の右シフトにより，右側（低位の桁）のビットは失われる。もし2数の絶対値が非常に異なるなら，小さい方の数は右シフトの結果ゼロになってしまう。

浮動小数点数 1.0 に加えたとき 1.0 と異なる結果になるような最小の正の浮動小数点数 ϵ_m のことを**機械精度** (*machine accuracy*) または**機械エプシロン**という。語長32ビットの典型的な2進（$B=2$）コンピュータの ϵ_m はほぼ 3×10^{-8} 程度である。荒っぽく言えば，機械精度 ϵ_m はその浮動小数点表現の相対精度，つまり，仮数部の最下位のビットが変化したときの相対誤差と考えることができる。浮動小数点数間の算術演算では，さらに少なくとも ϵ_m の相対誤差を生じると覚悟すべきである。この型の誤差を**丸め誤差** (*roundoff error*) という。

この ϵ_m は，表現可能な最小の浮動小数点数では決してない。最小の浮動小数点数は指数部のビット数によるが，ϵ_m は仮数部のビット数による。

計算を重ねるに従って，丸め誤差も累積する。もし最終的な答を得るまでに N 回の算術演算を行ったならば，全体の丸め誤差が $\sqrt{N}\epsilon_m$ の程度であるなら非常に幸運と言わなければならない（正負の丸め誤差がランダムに生じると，ランダムウォークの原理により平方根が現れる）。しかし，この誤差評価は，次の二つの理由で，現実離れしている。

1. 計算の規則性のため，あるいはコンピュータの事情のため，丸め誤差が同じ方向に累積することが，非常にしばしば生じる。この場合は全体の誤差は $N\epsilon_m$ の程度である。
2. たった1回の演算でも，非常に大きな誤差が生じることがある。一般にこのようなことは，非常に近い2数の減算で生じる。高位の桁がほとんど相殺され，低位の数ビットだけが残る。こんな「偶然」の減算はめったに生じないと思われるかもしれないが，そうではない。数式によっては，こういうことが起こる確率が非常に高いものがある。例えば，2次方程式の解の公式

$$x=\frac{-b+\sqrt{b^2-4ac}}{2a} \tag{1.3.2}$$

で $ac\ll b^2$（ac が b^2 よりずっと小さい）のとき，こういったことが生じる（§5.5でこの場合の回避法を述べる）。

丸め誤差はコンピュータのハードウェアの特性であるが，もう一つ，これとは異なる種類の誤差がある。こちらは，プログラムあるいはアルゴリズムの特性であり，ハードウェアにはよらない。多くの数値計算のアルゴリズムは，連続的なものを離散化して近似する。例えば，積分を数値的に求めるには，すべての点で関数値を計算するわけではなく，離散的な（飛び飛びの）点で関数値を計算する。また，関数値を求めるには，無限級数の有限個の主要項を加えて求める。無限個の項を計算するわけではない。このような場合には，点の個数とか項の個数のように，調整可能なパラメータがあり，「真」の答はそのパラメータが無限大になったときに得られる。現実には，パラメータの値は十分大きいが有限な値でなされる。

こういった場合に，真の答と現実の計算による答との食い違いを**打切り誤差** (*truncation error*) という。丸め誤差のない無限精度の「完全な」コンピュータでも，打切り誤差は存

在する。一般則として，丸め誤差はプログラマにはどうすることもできない。ただ，アルゴリズムをうまく選んで丸め誤差を不用意に拡大することを防ぐくらいしかできない（後述の「安定性」の解説参照）。しかし，打切り誤差はすべてプログラマの制御下にある。実際，少々誇張して言えば，打切り誤差を最小にすることが数値解析のすべてである。

ほとんどの場合，打切り誤差と丸め誤差は強く影響し合わない。まず無限精度のコンピュータで実行したときの打切り誤差があって，それに演算回数に伴う丸め誤差が加わると考えることができる。

しかし，ときとして，魅力的に見える方法が「不安定」であることもある。ここでいう「不安定」とは，最初の段階の計算に紛れ込んだ丸め誤差が次第に拡大され，最後には真の答を呑み込んでしまうことを意味する。無限精度のコンピュータなら不安定な方法も有用であろうが，現実の不完全な世界では，アルゴリズムは安定でなければならないし，もし不安定なアルゴリズムを使わなければならないならば，十分に注意しなければならない。

不安定なアルゴリズムの（やや人為的であるが）簡単な例を挙げよう。いわゆる「黄金分割比」

$$\phi \equiv \frac{\sqrt{5}-1}{2} \approx 0.61803398 \tag{1.3.3}$$

の2乗，3乗，4乗，... を計算したいとしよう。累乗 ϕ^n は次の簡単な漸化式に従うことが容易に証明できる。

$$\phi^{n+1} = \phi^{n-1} - \phi^n \tag{1.3.4}$$

したがって，最初の2個の値 $\phi^0 = 1$，$\phi^1 = 0.61803398$ がわかれば，式 (1.3.4) を次々に適用すればよい。こうすれば1回の減算だけで済むので，ϕ の乗算より速い。

残念ながら，漸化式 (1.3.4) はもう一つの解 $-\frac{1}{2}(\sqrt{5}+1)$ を持つ。この漸化式は線形であり，不要な解は絶対値が1より大であるため，不要な解が丸め誤差のために少しでも混入すれば，それは指数関数的に成長する。語長32ビットの典型的な機械では，式 (1.3.4) を使うと $n = 16$ くらいで全く見当外れの答になる。このとき ϕ^n はまだ 10^{-4} 程度である。要するに，漸化式 (1.3.4) は「不安定」であり，上の目的には使えない。

安定性の問題は本書でこれから何度も，さらに複雑な装いで登場する。

参考文献

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), Chapter 1.[*7]

Johnson, Lee W., and Riess, R. Dean. 1982, *Numerical Analysis*, 2nd ed. (Reading, Mass.: Addison-Wesley), §1.3.

訳注*7　第2版が1992年に出ている。