# 加減算の繰り返しによる桁落ち

## “式の値が 0 に近い” とは (方程式の解の精度的限界)

方程式 $f(x) = 0$ の解を見つけることとは, 数値的には $f(x) \approx 0$ となる $x$ を見つけることである. 次のような具体例を考える.

$$f(x) = x^3 - 15.70875x^2 + 61.69729x - 0.04844725 \tag{1}$$

の解は, $x \approx 0.0007853982$, $x \approx 7.844763$, $x \approx 7.863201$ である. この解の正確さは, 式の計算の際に生じる丸め誤差のために有効数字に上限ができ, 元の有効数字と同じになるとは限らない. 生じる丸め誤差を具体的に見積もるには, 前章で紹介された方法[1)] を用いて計算すればよい.

- $x \approx 0.0007853982$ の場合, (1) の各項の丸め誤差のうち最も大きいスケールの丸め誤差を持つと見積もられるのは第 3 項であり, その値は約 $\pm 2.4 \times 10^{-8}$ である. よって, この場合の $f(x)$ の丸め誤差は $\pm 2.4 \times 10^{-8}$ と見積もられる.
- $x \approx 7.844763$ の場合, (1) の各項の丸め誤差はいずれもほぼ同じスケールの丸め誤差を持つため, $f(x)$ の丸め誤差は 3 つの項の丸め誤差の和を取って $\pm 0.002$ と見積もられる[2)].
- $x \approx 7.863201$ の場合は $x \approx 7.844763$ の場合とほぼ同じ値となるため, $f(x)$ の丸め誤差も $\pm 0.002$ と見積もられる.

---

[1)] GFD ワークノート「絶対数の近い数の加減算による桁落ち (伊理テキスト 2 章)」の式 (40) を参照のこと.

[2)] 各項の値は $x^3 \approx 4.8 \times 10^2$, $-15.70875 \times x^2 \approx -9.7 \times 10^2$, $61.69729 \times x \approx 4.8 \times 10^2$ となり, いずれもほぼ同じスケールとなる. よって各項の値の平均を取った上で, 次数を評価して式の値 $f(x)$ の誤差を

以上より, 式 $f(x)$ の値そのものに対する丸め誤差が出るために, その誤差以上の精度で $f(x)$ を 0 に近づけようとする努力には意味がない. 結論として, $f(x)$ の誤差は前章で学んだ計算方法に起因するものと, いま上で見た $x$ の値に応じた誤差に起因するものがあり, 算法設計の際にはこの二つの誤差を十分に考慮しなければならないということがわかる. また, $f(x)$ の零点そのものの位置も正確には求めることができない.

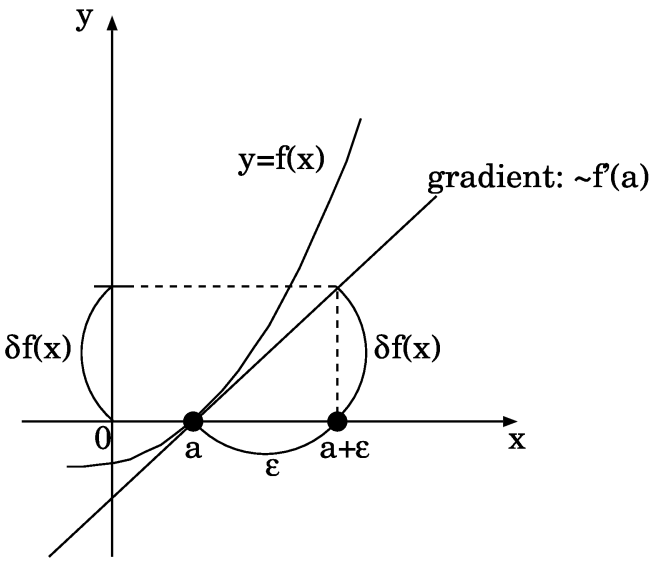


図 1　接線近似の模式図. $f(x)$ に $\delta f(x)$ の誤差があるとした場合, 零点の誤差 $\epsilon$ は式 (2) で求めることができる.

関数 $f(x)$ に $\delta f(x)$ だけの誤差があったとする. $f(x)$ を接線近似 (1 次近似) すると, $x$ のとりうる誤差 $\epsilon$ は,

$$\epsilon \approx \frac{\delta f(x)}{f'(a)} \tag{2}$$

となる (図 1 を参照). いま, 式 (1) において,

---

求めると

$$\begin{aligned}\delta f(x) &\approx (3+2+1) \times \frac{10^{-6}}{2} \times \frac{20 \times 10^2}{3} \\ &= 20 \times 10^{-4} \\ &= 0.002\end{aligned}$$

$$\begin{aligned} f'(0.0007853982) &\simeq 61.67, \\ f'(7.844763) &\simeq -0.145, \\ f'(7.863201) &\simeq 0.145 \end{aligned} \tag{3}$$

である. よって 10 進 7 桁の計算において $f(x)$ の零点そのものの位置は式 (2) より原理的に,

$$\begin{aligned} \left|\frac{2.5\times10^{-8}}{61.67}\right| &\simeq 4.0\times10^{-10}, \\ \left|\frac{0.002}{(-0.145)}\right| = \left|\frac{0.002}{0.145}\right| &\simeq 1.4\times10^{-2} \end{aligned} \tag{4}$$

程度の誤差を含むことになる. また, 接線近似自体が実際の関数曲線からわずかにずれるため, ここでも誤差が発生する.

上記のことを考えれば, 一般に流通している収束判定条件[3)]

$$\left|x^{(\nu)} - x^{(\nu-1)}\right| < \epsilon \tag{5}$$

を使った計算法についても, 解の位置に誤差があることを考慮したうえで計算を終了する条件 $\epsilon$ を決定しなければならない. 実際には $\epsilon$ をどのように設定するかが難しい. 詳しくは別回「ニュートン法」の回で取り上げる.

## 情報落ち / 積み残し

多数の数の和を取る際, 徐々に大きくなる被加算項に対して加算項が相対的に小さくなると, 扱える桁数に限界があるために加算からはみ出る, あるいはそもそも加算されないというような状況が起こり得る. これを「情報落ち」, あるいは「積み残し」と呼ぶ.

多数の数の和を求めようとして, 式に従って順々に足していくと, 図 2 のような現象が生じる.

これは被加算項が大きくなったために, 加算項の有効数字のうち斜線部分が結果に反映されなくなってしまっている. さらに計算を続けると, 図 3 のようになり, 加算項の情報は結果にまったく反映されなくなってしまう.

[3)] 繰り返し計算を用いて方程式の解を求めようとするとき, 連続する二つの $x$ の値 $x^{\nu}$, $x^{\nu-1}$ の差が $\epsilon$ 未満になれば, その $x^{\nu}$ を解として採用する方法. $x^{\nu}$ は解の第 $\nu$ 近似を表す.

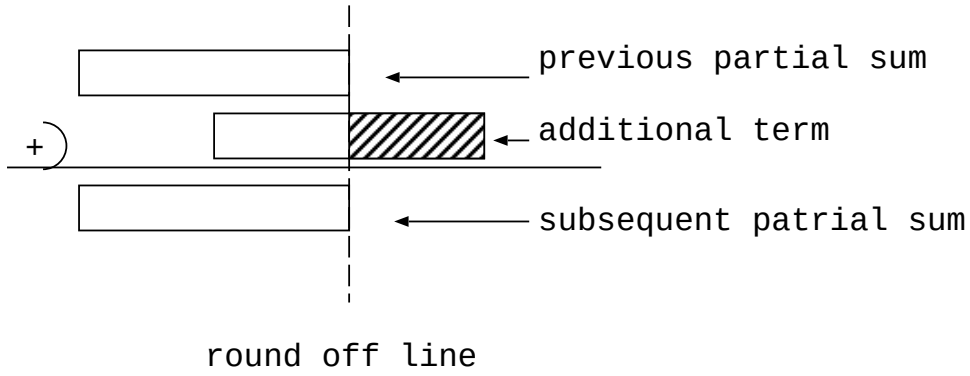


図 2　多数の和を求めるときに生じる現象の模式図. ある程度まで足し込むと, 被加算項が大きくなるために加算項の一部の情報が抜け落ちてしまう.

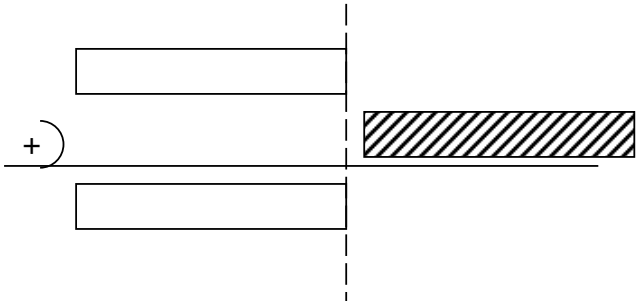


図 3　加算項の欠如の様子を表す模式図. 加算項の情報はすべて抜け落ちてしまう.

# 積み残しの典型例

積み残しの典型例を 3 つほど挙げてみる.

## 台形則近似による定積分の数値計算

具体的な例として次のような定積分を考える.

$$I = \int_a^b f(x)\, dx. \tag{6}$$

これを台形則近似によって解こうとすれば,

$$I = \frac{b-a}{N}\left[\frac{1}{2}f(a) + \sum_{n=1}^{N-1} f\left(a + \frac{n}{N}(b-a)\right) + \frac{1}{2}f(b)\right] \tag{7}$$

を計算することになる (図 4 参照. ).

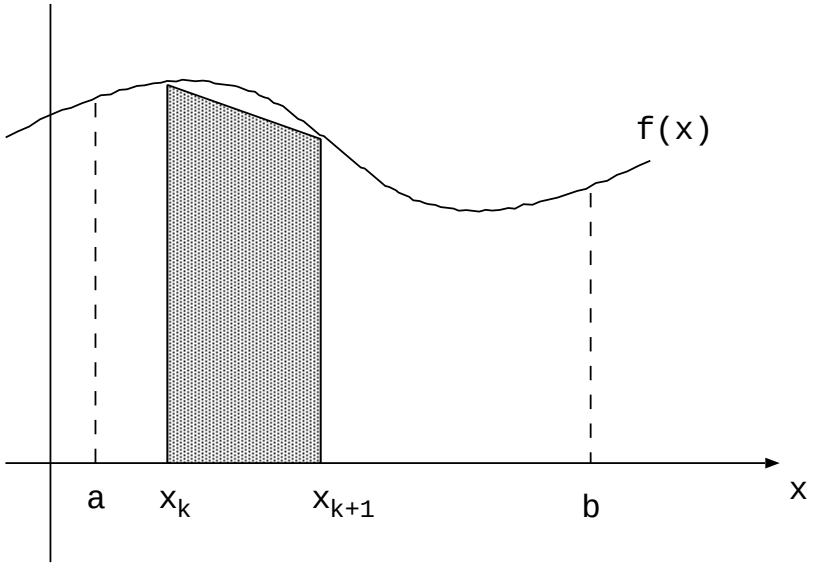


図 4　台形則近似の概念図. 台形則近似では関数 $f(x)$ の積分値を, 図の斜線部分のような台形の面積の集まりと考える. 斜線部分の面積は $\{f(x_{k+1}) + f(x_k)\} \times \frac{b-a}{N} \times \frac{1}{2}$ で与えられる.

原理的には, 分割数 $N$ を大きくしていくことで, 数値解はより本来の値に近づくことが期待される. しかしながら加算項が細分化され小さくなるため, 前述の積み残しが発生する可能性が出てくる.

## 常微分方程式の初期値問題の数値解法

常微分方程式

$$\frac{dy}{dx} = f(x, y) \tag{8}$$

の厳密解は,

$$y = \int_{x_0}^{x} f(\xi, y(\xi))d\xi + y(x_0) \tag{9}$$

である. これを計算機で数値計算できるように離散近似することを考える. まず (9) の積分区間を $n$ 分割し,

$$y_{n+1} = \int_{x_n}^{x_{n+1}} f(\xi, y(\xi))d\xi + y(x_n) \tag{10}$$

とあらわす. ここで区間 $x_n \leq x < x_{n+1}$ において $f(x_n, y_n)$ の値 は一定であると仮定すれば, (10) は,

$$\begin{aligned} y_{n+1} &= \int_{x_n}^{x_{n+1}} f(\xi, y(\xi))d\xi + y_n \\ &= f(x_n, y_n) \int_{x_n}^{x_{n+1}} d\xi + y_n \\ &= f_n(x_{n+1} - x_n) + y_n \\ &= f_n h + y_n \end{aligned} \tag{11}$$

となる. ただし,

$$x_{n+1} = x_n + h, \quad f_n = f(x_n, y_n). \tag{12}$$

ゆえに, (9) を離散近似した式は,

$$y_{n+1} = y_n + hf_n \tag{13}$$

となる. (13) をよく見てみれば, 積み残しの起こることは想像に難くない. つまり, $n$ を無限大にもっていき, 刻み幅 $h$ を極端に小さくすることを考える. すると, 計算がある $n$ に達したとき $y_n$ に比べて被加数 $hf_n$ が丸め誤差として計算に反映されなくなってしまう.

## 大量の統計データからその平均と分散を計算する

1 円玉 30 個の目方を精密天秤で量り, それら目方のデータの平均と分散を求めるというものである. 伊理テキスト 17 ページ表 3-1 を以下に掲載する.

表 3-1 1 円玉の目方 (単位:g)

| | | | | | |
|---|---|---|---|---|---|
| 0.9997 | 1.0007 | 0.9995 | 1.0002 | 1.0001 | 1.0004 |
| 1.0005 | 0.9998 | 0.9996 | 1.0004 | 1.0004 | 1.0004 |
| 0.9993 | 0.9999 | 0.9988 | 1.0001 | 0.9999 | 0.9994 |
| 0.9997 | 0.9991 | 1.0002 | 1.0004 | 0.9989 | 0.9989 |
| 1.0000 | 1.0009 | 1.0006 | 0.9993 | 0.9997 | 1.0001 |

このデータを用いて実際に平均 $\bar{x}$ と 分散 $v$ を計算してみる. 目方を $x_i(i = 1, \cdots, n; n = 30)$ とし,

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{14}$$

$$v = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2 \tag{15}$$

を用いて 2 進 24 桁の精度 (丸めは 0 捨 1 入) で計算を行った. 実際のプログラムは「プログラム例: 1 円玉の目方の平均と分散」を参照のこと. 結果は

$$\bar{x} = 0.99989671 \tag{16}$$

$$v = 1.78813934 \times 10^{-7} \tag{17}$$

となった. 尚, $v$ を定義式[4)]

$$v = \frac{1}{n}\sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{18}$$

で計算すると,

$$v = 3.08314554 \times 10^{-7} \tag{19}$$

となった[5)]. この食い違いはもちろん, 積み残しに依るものである. (17) では, 積み残しによって値が大きく減ってしまっている. また, いまは 30 個のデータで確認してみた. しかし, 伊理テキストによれば (15) の計算をランダムな $n = 1000$ 個のデータに対して行うと,

$$v = -1.2517 \times 10^{-6} \tag{20}$$

[4)] 分散の定義式 (18) を変形すると (15) になる. 各自確認すること.

[5)] この違いは使用するコンパイラに依存する. 例えば富士通 Fortran だと結果に違いは生じなかった.

という結果になったという. これは理論的には起こり得ない. なぜならば,

$$\sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} [\bar{x}^2 + 2\bar{x}(x_i - \bar{x}) + (x_i - \bar{x})^2]$$
$$= n\bar{x}^2 + \sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{21}$$

であるので, 理論的には,

$$\frac{1}{n} \sum_{i=1}^{n} x_i^2 > \bar{x}^2 \tag{22}$$

である. したがって, (15) は理論上負の値を取りえない. しかし, 実際には (15) の計算の過程で積み残しが発生し, (22) の関係が満たされなくなってしまったのである.

# 積み残しの回避方法

多数の数の和をつくるときに生じる積み残しの回避方法を列挙する.

## 倍精度を用いる

単純に, 計算機の表示できる有効桁数を増やす方法である. 計算コストはかかるが簡単で頭を使わない方法である.

## 補助変数の導入

単精度計算でも倍精度計算に近い結果を出すことができる方法である. 多数の数の和をつくる場合には, 部分和を $S$ とすると,

do
　　$S = S + x_i$
end do

というような計算をすることになる. ここへ積み残された値を表す補助変数 $R$ (初期値は 0 ), 及び作業変数 $T$ を導入して,

$$
\begin{aligned}
&\text{do}\\
&\qquad R_2 = R_1 + x_i\\
&\qquad S_2 = S_1 + R_2\\
&\qquad T = S_2 - S_1\\
&\qquad R_1 = R_2 - T\\
&\qquad S_1 = S_2\\
&\text{end do}
\end{aligned}
$$

と計算し, 誤差を再評価することで積み残しを軽減することができる. また, 計算結果は $S_2$ に格納される. なお, 上のアルゴリズムでは次のような計算を行っている. [6)]

1. 丸め誤差などで $S$ に正確には足し込まれなかった過去の計算の過不足分を $R$ に代入する
2. $x_i$ は $R$ に足し込まれる
3. $S_2 = S_1 + R_2$ で $S_2$ に足し込まれる (この足し算で積み残しが発生する)
4. $T = S_2 - S_1$ で, 実際に足し込まれた量を計算
5. $R_1 = R_2 - T$ で新しい過不足分を計算し, 次の加算へ持ち越す
6. $S_1 = S_2$ で次の加算の為に $S_1$ を再定義する

## 2 分木法

積み残しの回避方法としては「2 分木法」も理論的には良いとされている. 2 分木法とは端から順に和をとるのではなく, 被加算項を 2 つずつペアにして和をとり, それらをさらに 2 つずつ和をとるということを繰り返す方法である. 各項似たもの同士の足し算をすることで, 加算項と被加算項の差が大きくならないようにしている. 2 分木法は被加算項が $x_1, x_2, \cdots$ と順々に読み込まれるときでも, 必要なメモリーは $\log_2 n$ 程度で済む.

[6)] 実際にプログラムを実行してみると, 積み残しが格納される変数 R1 に負の値が格納されるループがあった. 理論的には積み残しは正となるように思われ, なぜ負の値になるのか分からなかった. 丸めによってこの現象が発生するのではないかという意見が出たため, 次回はこれを確認したい. fortran コンパイラは丸めの方式を指定できる.

$n$ 個の数を加算する場合に, どの程度のメモリが必要なのかを確かめる. 2 分木法の考えに基づき, $n$ を分割すると,

$$\begin{aligned} n &\sim \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + \frac{n}{2^k} \\ &= \sum_{k=1}^{\frac{n}{2^k} \leq 1} \frac{n}{2^k} \end{aligned} \tag{23}$$

となる. 最後の項の数字は必ず 1 以下になるので $n \leq 2^k$ の範囲で $k$ の最小値が必要なメモリ量になる.

$$\begin{aligned} \frac{n}{2^k} &\leq 1 \\ n &\leq 2^k \\ \log_2 n &\leq k \end{aligned} \tag{24}$$

よって, 必要なメモリは $\log_2 n$ だが実際に 2 の乗数のときは 1 メモリ消費が増えるので「程度」としている.

これまで積み残しの回避方法を 3 つほどを考察してきた. しかし, 本当の解決策は丸め誤差の観点を重視して計算を設計することである. 例えば 1 円玉の例で言えば, 1 g からのずれの部分だけを取り上げて平均, 分散を計算するのがよい. すなわち, $x_i - 1.0000$ を用いて計算をすれば積み残しは起こらない[7)].

# 参考文献

伊理正夫・藤野 和建, 1985: 数値計算の常識, 共立出版

[7)] この操作のことを「オフセットをとる」という. オフセットとは必要なデータの位置を基準点からの差で表した値のことである.