数値計算における統計的誤差評価方法の提案

Propose of statistical error estimation in numerical process

幸谷　智紀*

Tomonori KOUYA

Abstract: In real numerical computations using finite-length floating-point numbers, the effects of rounding errors during the execution process are unavoidable and can lead to inaccurate final results for ill-conditioned or large problems. While it is possible to estimate the effect before execution by using forward or backward error analysis, the amount of effect tends to be larger than real one. We have already presented a method to estimate rounding errors by changing the rounding mode, but some computing environments, such as GPUs and TensorFlow, do not allow changing rounding mode via running programs. In this paper, we alternatively propose a statistical estimation method of round-off error. It aims to investigate how the difference in the final result statistically varies through its mean and distribution by inserting a random error larger than the rounding error for the floating-point arithmetic used. We finally demonstrate the effectiveness of the statistical estimation through concrete numerical examples.

1. 初めに

浮動小数点数を用いる計算は，従来の科学技術計算が担ってきた自然現象のシミュレーションのみならず，深層学習の利用の広がりとともに非線形活性化関数を取り込んだニューラルネットワーク，すなわち人間の脳細胞活動のシミュレーションにまで利用されるようになっている．一方，コンピュータ資源の源泉であるハードウェアの高速化は，デジタル回路の微細化の限界が近づきつつあり，限られたリソースを有効活用することが常に求められる．浮動小数点演算においても，ターゲットとする計算の量や質を担保できるよう，仮数部・指数部ともユーザのニーズに応じた長さのものを選択し，縦横に組み合わせて実行する混合精度演算の利用が日常的になりつつある．

とはいえ，有限桁の浮動小数点数を使用する限り，リアルな数値計算過程では丸め誤差が生じるという事情は今も昔も変わらない．当然のことながら，丸め誤差に鋭敏な悪条件問題や，丸め誤差の蓄積が問題となる大規模問題では，最終的な数値解の精度が失われ，ユーザの要求する正確さを担保することができないことがままある．そのため，計算過程でどの程度の精度が失われ，最終的な数値解の正確さがどの程度かを見積もる技術，すなわち誤差解析の知見が重要となる．

現在では，主要なアルゴリズムに対しては誤差解析が入念に行われており，古くは Wilkinson[6]，現在では例えば Higham[3] の著作が参考になる．これらの知識を用いることで，計算実施前に丸め誤差による影響を求めることも可能であるが，例えば区間演算のように厳密さを重視しすぎると実際よりも過大なものになりがちである．そこで我々は丸め方式を変えて得た数値解の差異を評価値として用いる手法を既に提案している．しかし，TensorFlow のように丸め方式の変更がユーザプログラムからはできないケースや，新たに丸め方式ごとの計算ルーチンを用意しなければならない GPU のような計算環境も存在する．このようなケースでは我々が提案する丸め誤差評価方法が使用できない。

かつて山下眞一郎は，簡易的に計算における誤差の鋭敏性を見積もるため，入力する浮動小数点数の末尾桁に誤差を自動的に挿入し，出力結果への影響を調べる「やまびこ (echo) 方式」を考案し，富士通の FORTRAN コンパイラにやまびこ方式を実行するオプションを設けた．厳密な誤差解析に基づくものではなく，職人芸的なアイディアではあるが，計算環境に依存することなく実行でき，プログラムを書き換える必要がないため，現在でも有用な方法と言える．

とはいえ，現代のコンパイラにやまびこ方式オプションを付加することは現実的でない。また，やまびこ方式そのものが，経験論に基づく手法であることから，数学的な裏付けに乏しいことは否めない．しかし，計算が大規模化し，環境も複雑化する昨今，これらの欠点を補いつつ，手軽に実行できる新たな「やまびこ方式」が今こそ必要になっているとも言える．

そこで本稿では，やまびこ方式同様，計算に使用する数値データに，末尾桁だけでなく，任意の大きさの統計的な誤差を挿入し，得られる数値解の差異の平均値や分散・標準偏差を計測することで丸め誤差の影響度を計測する方法を提案する．非線形関数であっても，最終的に得られた誤差の見積もり量を，入力値に挿入した誤差の統計量と比較することで，末尾桁の変化にとどまらない広範な出力結果の精度への影響度を「統計的に」説明できることが期待できる．数値計算過程において発生する丸め誤差を確率変数とする考え方は Henrich[2] から始まっているが，現在は Higham らが理論的な観点からの確率的誤差評価法を提案しており [4]，それを数値的に再現しているものとも解釈できる。

本論文では，この新たなやまびこ方式，すなわち「統計的誤差評価法」の提案を行い，NumPy や SciPy を用いた Python スクリプトを用いてその有用性を示す．

2023 年 5 月 26 日 受理

* 情報学部 コンピュータシステム学科

2. 浮動小数点演算における丸め誤差とその評価法

現在の主要な CPU や GPU における多くの浮動小数点演算は IEEE754-1985 規格で定められた 2 進 (binary) のものである．ここで使用される仮数部 t bit の浮動小数点数の集合を $\mathbb{F}_{bt}$ と書くこととする．従って，binary16(半精度) は $\mathbb{F}_{b11}$, binary32(単精度) は $\mathbb{F}_{b24}$，科学技術計算の標準である binary64(倍精度) は $\mathbb{F}_{b53}$ となる．

$a, b \in \mathbb{F}_{bt}$ に対する演算 $c = a \circ b\ (\neq 0)$ は，計算結果が t 桁に収まらない場合は桁を打ち切って (丸めて) 格納する．ここで丸めた値を $\tilde{c} \in \mathbb{F}_{bt}$ とする．その際に生じる誤差，すなわち丸め誤差は，

$$\delta := \frac{\tilde{c} - c}{c} \tag{1}$$

と表現できる．例えば $t+1$ 桁目を 0 捨 1 入して得たとすると，δ は必ず

$$|\delta| \le u = \frac{1}{2} 2^{1-t} = 2^{-t} \tag{2}$$

を満足する．ここで u を丸め誤差の最小単位と呼ぶ．

以下，やまびこ方式，丸めモード変更による計測方法，そして今回提案する統計的推定法について概説するが，共通しているのは，「有限桁の浮動小数点演算における (丸め) 誤差による影響は，同じアルゴリズムを使う限り，使用する浮動小数点数の仮数部桁数 (t) によらず一定である」という経験則に基づいていることである。

2.1 山下によるやまびこ方式

現時点で，丸め誤差評価方法としての「やまびこ方式」(echo system to evaluate effect of round-off errors) の学術論文は見つからず，公式 (?) 文書としては山下眞一郎 (元日本大学教授) が個人的に配布していた富士通の内部文書 (Fig.1) があるのみである。

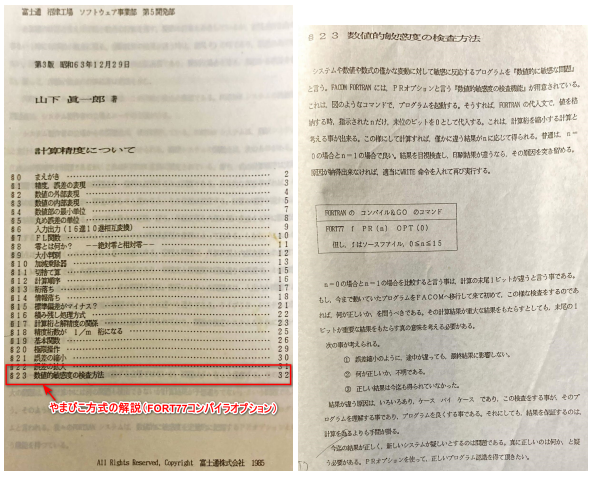

Fig.1 山下によるやまびこ方式の解説文書: 目次 (左) とやまびこ方式の解説ページ (右)

これによると，コンパイラオプションで指定した bit 数 (m) だけ，Fortran プログラム中の浮動小数点数の末尾 bit を強制的にゼロクリアすることで，オリジナルの計算結果との差異を調べることができる，というものであることが分かる。ゼロクリアする方法については判然としないが，単純に m bit 分をゼロで埋めるとすれば切り捨てによる丸めを行っていると解釈できる。

従って，やまびこ方式は，数学的には $\mathbb{F}_{bt}$ の浮動小数点演算結果を，切り捨て丸めを行って $\mathbb{F}_{b(t-m)}$ に計算桁数を減らした値と比較する丸め誤差評価方法であると言える。発案者本人の弁によれば，自分以外にこのやまびこ方式を利用して役立ったという報告は聞いておらず，世間的にも社内的にも普及したとは言えないとのことであった。

2.2 丸め方式変更による誤差の評価方法

本手法は，計算桁数を変えずに丸め誤差を検出する方法として考案したものである。発案したきっかけは，1991 年の卒業論文執筆時に，たまたま Intel 8087/80287(数値演算コプロセッサ) に丸め方式の切り替え方法があることを知り[1]，陽的 Runge-Kutta 法の C プログラムで丸めモードを変更した値の差異を調べたという経験による。当然，やまびこ方式の存在は知らず，後に博士論文執筆時に本方式を提案した査読論文[5]を山下本人に提示したところ，「これは僕のエコーシステムだね」という感想をもらったことで，初めてその存在を認知した。個人的には，やまびこ方式よりはシステム的に確実かつ簡単に丸め誤差の評価を実行することができると自負していたが，基本的な発想としては末尾 bit の変更による差異を調べるという点で同じものであるとも言える。

以下，Pytnon でこの方式[7]を実装したものを簡単に紹介する。CPU 計算で丸めモードを変更するためのソースコードは Git Hub*に公開してある。

現在，浮動小数点演算ユニットがハードウェアとして備わっている CPU では，デフォルトの丸めモードである RN(Round to Nearest) で binary32, 64 の計算が実行される。丸め方式としては，その他，$+\infty$ 方向への丸めを行う RP(Round to Plus infinity)，$-\infty$ 方向への丸めを行う RM(Round to Minus infinity) が備わっている．我々の提案する丸め誤差評価方法は，計算過程の前にこの丸めモード変更を行い，その際の最大値で相対誤差の評価を行うというもので，簡易的な区間演算 (interval analysis) であるともいえる．

丸めモード変更のためには，Linux, Windows, macOS それぞれに浮動小数点演算ユニットの状態を変化させるための標準的な関数が備わっており，それを利用すればよい．Linux, macOS では `fesetround` 関数，Windows では`_controlfp_s` 関数を使用する．我々は OS によらず Python 環境で丸めモードの設定と確認ができるよう，`rmode.py` を作成し，ここで次の 2 つの関数を定義した．

set_rmode 関数 ・・・`FE_NEAREST`(RN), `FE_UPWARD`(RP), `FE_DOWNWARD`(RM) を引数に与えて丸めモードを変更する．

get_rmode 関数 ・・・この関数実行時の丸めモードを標準出力に表示する．

`rmode.py` を読み込んで，これらの関数を使い，それぞれの丸めモードでの数列 $\{x_n^{\mathrm{RN}}\}, \{x_n^{\mathrm{RP}}\}, \{x_n^{\mathrm{RM}}\}$ を求め，その差異の最大値 $x_n^{\max} := \max\{|x^{\mathrm{RN}} - x^{\mathrm{RP}}|, |x^{\mathrm{RN}} - x^{\mathrm{RP}}|, |x^{\mathrm{RM}} - x^{\mathrm{RP}}|\}$ を

* https://github.com/tkouya/inapy/blob/master/chapter04/rmode.py

x_n^{RN} の絶対誤差の評価値として使用する．実用的には，区間解析のように RM モード値と RP モード値を求め，それぞれデフォルトの RN モードのと差異を取れば十分である。

3. 統計的誤差評価方法

今回提案する統計的誤差評価方法の模式図を Fig.2 に示す。このケースでは入力値として $x \in \mathbb{F}_{bt}$，計算結果は $f(x) \in \mathbb{F}_{bt}$ で出力するものとしている。

まず，$\mathbb{F}_{bt}$ の仮数部 t bit 内の特定の位置に乱数による相対誤差 r_i を x に与えて $\widetilde{x} := x(1+r_i)$ を生成する。次に，出力結果 $f(\widetilde{x}) = f(x)(1+s_i)$ を得てこの相対誤差 s_i を計測する。これらを比較することで r_i がどの程度 s_i まで拡大 (縮小) したかを知ることができる。また，試行回数を増やし，r_i の平均 (μ_r)・標準偏差 (δ_r) と s_i の平均 (μ_s)・標準偏差 (δ_s) を比較することで，統計的な正確性を担保することも可能である。

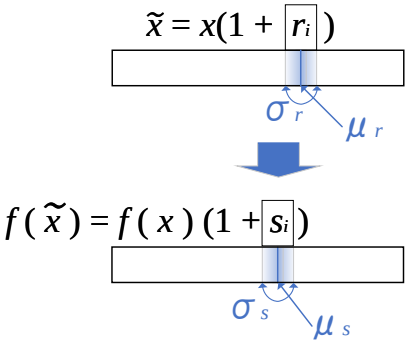

Fig.2　統計的誤差推定方法の概念図

考え方としては，やまびこ方式におけるゼロクリアを乱数の挿入に置き換えただけとも言えるが，次のような改良がなされているとも主張できる。

1. 挿入する乱数が依存する確率分布を変えることで，乱数的な丸め方式をエミュレートできる。
2. コンパイラオプションや丸めモード変更よりもプログラム中の数値に与える影響範囲を限定できる。
3. ゼロクリアや丸めモード変更より実装が容易である。
4. 試行回数を増やすことで統計的な正確性を担保できる。

我々の提案する統計的誤差評価方法は，工業製品の製造過程における抜き取り検査のように，素朴だが統計的経験論に基づく誤差の検出方法であると言える。

4. 数値例

ここでは下記 Python 環境下で統計的誤差評価方法を試みた結果を示す。

OS　Windows 11 Pro 22H2
CPU　Intel Core i7-9700K 3.6GH
RAM　16GB
Python　3.10.6, NumPy, SciPy, matplotlib

4.1 連立一次方程式

まず，実係数連立一次方程式 (3) に対して統計的誤差評価法を適用した結果を示す。

$$A\mathbf{x} = \mathbf{b},\ A \in \mathbb{F}_{b53}^{n\times n}, \mathbf{x} = [1,2,..,n]^T \in \mathbb{F}_{b53} \qquad (3)$$

A の要素 $a_{ij}(i,j,1,2,...,n)$ は一様乱数で与えたものを使用する。今回は a_{ij} に $O(10^{-5})$ の一様乱数を相対誤差 r_{ij} として与え，$\widetilde{a_{ij}} = a_{ij}(1+r_{ij})$ を用として持つ $\widetilde{A}$ を係数行列として直接法 (LU 分解 + 前進・後退代入) で解き，誤差を含む $\widetilde{x_i} = x_i(1+s_i)$ の相対誤差 s_i を評価する。

その結果を Fig.3 に示す。横軸が $|r_i|$ の平均値 (左)，中央値 (中)，標準偏差 (右)，　縦軸が $|s_i|$ の平均値，中央値，標準偏差である。

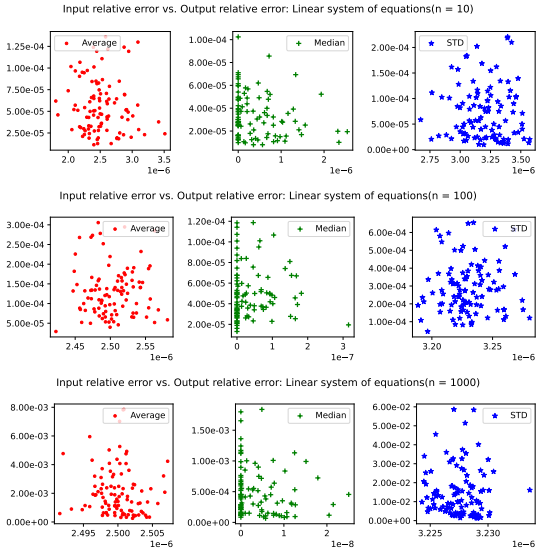

Fig.3　連立一次方程式の誤差: n = 10(上), 100(中), 1000(下)

この事例では，$n = 10$ ではほぼ 10 進 10 桁の精度を維持しているが，$n = 100$ で 1 桁，$n = 1000$ で 2 桁程度の損失が現れている。実際，平均値はその変異を的確に捕えており，誤差の推定が出来ていると言える。平均値を中心に対象となる一様乱数を使用しているため，n が大きくなるにつれて初期誤差の中央値は 0 に収束してくる様子が見られ，出力の中央値についても同様の傾向が見られる。標準偏差については，初期誤差においては一定なのに対し，出力値については平均値と比例して大きくなっている。

以上の結果から，本事例に関しては適切な誤差評価が出来ていると言える。

4.2 常微分方程式の初期値問題

次に，常微分方程式の初期値問題 $\mathbf{y}' = \mathbf{f}(t,\mathbf{y})$ に対し，初期値 $y(t_0)$ に誤差を挿入して，積分区間 $[t_0, t_{\mathrm{end}}]$ に ODE ソルバーを適用し，t_{end} における誤差を評価してみる。

使用するのは，Rössler モデルである。ここで $\alpha = 1/5$, $\beta = 1/5, \mu = 5$ である。

Table 1　Rössler モデルの平均相対誤差

誤差 Order		10^{-5}		
t_{end}	n	Rel.Err.(y_1)	Rel.Err.(y_2)	Rel.Err.(y_3)
50	10	2.6e-05	2.8e-04	4.5e-05
	50	2.4e-05	2.5e-04	4.0e-05
	100	2.0e-05	2.1e-04	3.4e-05
100	10	1.7e-04	3.2e-03	1.6e-03
	50	9.0e-04	4.8e-03	2.6e-03
	100	1.1e-03	6.7e-03	3.2e-03
200	10	1.1e-01	6.0e-01	2.0e-01
	50	7.2e-02	4.0e-01	1.5e-01
	100	8.0e-02	4.2e-01	1.9e-01
500	10	1.8e-01	2.5e-01	1.4e-01
	50	1.5e-01	3.3e-01	9.4e-02
	100	8.6e-02	2.3e-01	5.4e-02

Table 2　Rössler モデルの平均相対誤差

誤差 Order		10^{-10}		
t_{end}	n	Rel.Err.(y_1)	Rel.Err.(y_2)	Rel.Err.(y_3)
50	10	1.4e-10	1.5e-09	2.5e-10
	50	2.0e-10	2.2e-09	3.7e-10
	100	2.0e-10	2.1e-09	3.6e-10
100	10	5.9e-09	4.4e-08	4.3e-09
	50	7.6e-09	5.7e-08	5.5e-09
	100	6.2e-09	4.7e-08	4.5e-09
200	10	7.0e-02	4.0e-01	1.5e-01
	50	4.9e-02	2.6e-01	1.1e-01
	100	4.4e-02	2.4e-01	7.9e-02
500	10	7.3e-02	2.5e-01	5.5e-02
	50	1.0e-01	1.7e-01	7.9e-02
	100	7.8e-02	2.0e-01	6.4e-02

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -(y_2 + y_3) \\ y_1 + \alpha y_2 \\ \beta + y_3(y_1 - \mu) \end{bmatrix} \tag{4}$$

初期値は $\mathbf{y}(0) = [1, 0, 0]^T$ とし，ここに $O(10^{-5})$ と $O(10^{-10})$ の誤差 r_i を挿入して $\widetilde{y_i} = y_i(1 + r_i)(i = 1, 2, 3)$ を初期値として与え，SciPy の Integration パッケージの `solve_ivp` 関数を用いて解いた。その結果を Table 1 と Table 2 に示す。すべて t_{end} における相対誤差の平均値で，試行回数は 10, 50, 100 と変化させて，初期誤差を $O(10^{-5})$, $O(10^{-10})$ としてそれぞれの場合に対して平均値の変化を見ることにした。

まず試行回数の違いにより，相対誤差の平均値に有意な差が生じていないことが分かる。実用的には 10 回以下で十分な値が得られているとみてよい。

このパラメータの Rössler モデルの場合，$t_{\text{end}} \geq 200$ となると binary64 計算でも有効桁がほとんど失われ，この事例でも，初期誤差のオーダーに寄らず，相対誤差がほぼ同じ値になってしまっていることからも分かる。従って，我々が提案する統計的誤差推定法の有効性を確認するには $t_{\text{end}} = 50$, 100 の場合に限られる。前者の場合，有効桁は初期誤差の 10 進 5 桁, 10 桁程度と変化がないが，後者の場合，1 桁程度の損失が出ていることが分かる。初期誤差のオーダー差がそのまま残っており，Röessler モデルにおいて，$t_{\text{end}} = 100$ で丸め誤差の影響が大体 10 進 1 桁程度出ることが推定できる。

以上の結果から，本事例に関しては適切な誤差評価が出来ていると言える。

5.　まとめと今後の展開

以上，連立一次方程式と常微分方程式の初期値問題に対し，統計的誤差評価法を適用し，その有用性を確認した。今後の課題としては，他の問題，特に非線形問題や大規模問題に対して本手法が有効であるかを確認することが挙げられる。

参考文献

1) 春海佳三郎, 西見二昭, 小林史典ほか. 数値演算プロセッサ: 浮動小数点演算入門から高速演算プログラミングまで. IF スペシャル. CQ 出版社, 1987.
2) ヘンリッチ, 清水留三郎・小林光夫訳. 計算機による常微分方程式の解法 I, II. サイエンス社, 1973.
3) Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.
4) Nicholas J. Higham and Theo. Mary. A new approach to probabilistic rounding error analysis. *SIAM Journal on Scientific Computing*, Vol. 41, No. 5, pp. A2815–A2835, 2019.
5) 幸谷智紀, 永坂秀子. IEEE754 規格を利用した丸め誤差の測定法について. 日本応用数理学会論文誌, Vol. 7, No. 1, pp. 79 – 89, 1997.
6) J. H. Wilkinson. *Rounding Errors in Algebraic Process (Reprint edition)*. Dover, 1994.
7) 幸谷智紀. Python プログラミング環境における多倍長精度数値計算について. 静岡理工科大学紀要, Vol. 28, pp. 23–31, 2020.