Title	経済分析におけるコンピューターの誤差問題(その2) : 誤差累積の法則
Sub Title	Errors generated by computer in economic analysis (2)
Author	鳥居, 泰彦 新井, 益洋
Publisher	慶應義塾経済学会
Publication year	1975
Jtitle	三田学会雑誌 (Keio journal of economics). Vol.68, No.7/8 (1975. 8) ,p.617(33)- 623(39)
JaLC DOI	10.14991/001.19750801-0033
Abstract	
Notes	研究ノート
Genre	Journal Article
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=AN00234610-19750801-0033

研究ノート

経済分析におけるコンピューターの誤差問題（その2）

——誤差累積の法則——

鳥居　泰彦
新井　益洋

7. 誤差累積の法則性と誤差累積の最小化

7.1 加算における誤差累積とその最小化（その1）

前節では，コンピューター内部で発生する誤差が累積していく様子を追跡する方法として，プロセス・グラフの方法が有効であることを示した。そこでプロセス・グラフの方法を応用して，四則演算で誤差が累積する状況を追跡してみた。その結果，コンピューターを用いる計算では，代数学が予想もしていなかった興味深い現象が起ることが判明した。プロセス・グラフによって見出した誤差累積の法則から，誤差を最小限度に抑えるような計算手順を幾つか知ることができた。

本節では，3個以上の数値の加算における誤差累積の様子を調べよう。

3個の数値，x_1, x_2, x_3 の加算

$$y=x_1+x_2+x_3 \tag{7.1}$$

を考えてみよう。ただし，この3個の数値はすべて正の値であるとする。また，x_1, x_2, x_3 はいずれもデータに固有の誤差を持っていないとする。

3個の数値の加算は2回の加算操作で完了するから，コンピューターの中で発生する誤差は次の2つの切り捨て誤差だけである。

r_1：第1回目の加算結果の切り捨ての相対誤差

r_2：第2回目の加算結果の切り捨ての相対誤差

さて，この3個の数値 x_1, x_2, x_3 を順に加えて行く時，誤差がどのように累積するかをプロセス・グラフ〔Fig. 7.1〕を用いて評価してみよう。全体の相対誤差は次のようになる。

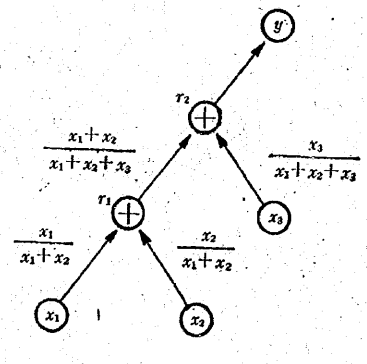
$$\frac{E_y}{y}=r_1\cdot\frac{x_1+x_2}{x_1+x_2+x_3}+r_2 \tag{7.2}$$

両辺に $y=x_1+x_2+x_3$ を掛ければ，最終計算結果 y の絶対誤差は次のようになる。

$$E_y=r_1(x_1+x_2)+r_2(x_1+x_2+x_3) \tag{7.3}$$

切り捨てによる相対誤差 r_1, r_2 の範囲は第4章で示

〔Fig. 7.1〕 $y=x_1+x_2+x_3$ のプロセス・グラフ



したように，有効桁数が t 桁のコンピューターでは

$$|r_1|\leqq 10^{-t+1}$$
$$|r_2|\leqq 10^{-t+1} \tag{7.4}$$

であるから (7.3) 式に代入して

$$|E_y|\leqq(2x_1+2x_2+x_3)\times 10^{-t+1} \tag{7.5}$$

となる。先に示したように x_1, x_2, x_3 は正数であるから，絶対誤差 E_y は，加えるべき数 x_1, x_2, x_3 の大小関係が

$$x_1\geqq x_2>x_3 \tag{7.6}$$

〔注〕この研究ノートは，慶應義塾大学産業研究所および情報科学研究所において鳥居・新井が行なっている共同研究の成果の一部である。本稿は前号（三田学会雑誌68巻6号，1975年6月号所収の「経済分析におけるコンピューターの誤差問題」，その1）の続編である。従って，章，節の番号も前号に続けることとし，今回は第7章から始める。なお，この稿はさらに次号に続く予定である。

の時最大となり，反対に

$$x_1 \leqq x_2 < x_3 \tag{7.7}$$

の時最小となる。このことは(7.5)式の右辺の( )内の係数をみれば理解できるであろう。

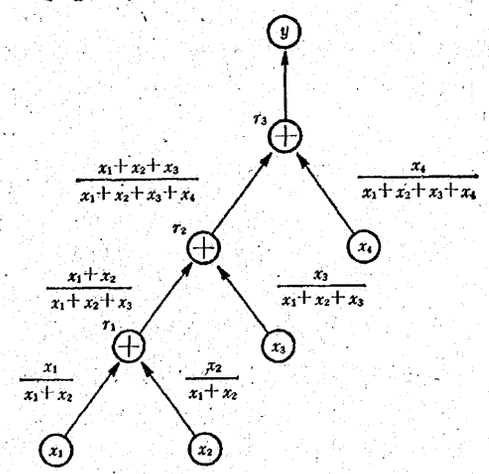
くどいようだが，事態をよりよく理解するために4個の正数 x_1, x_2, x_3, x_4 を加える逐次加算の場合を再論しよう。

$$y = x_1 + x_2 + x_3 + x_4 \tag{7.8}$$

1回目の加算 (x_1+x_2) で発生する切り捨ての相対誤差を r_1，2回目の加算 $((x_1+x_2)+x_3)$ で発生する切り捨ての相対誤差を r_2，3回目の加算 $((x_1+x_2+x_3)+x_4)$ で発生する切り捨ての相対誤差を r_3 とすると，全計算過程はプロセス・グラフ〔Fig. 7.2〕であらわせる。〔Fig. 7.2〕を用いて全体の相対誤差を評価すると次のようになる。

$$\frac{E_y}{y} = r_1 \cdot \frac{x_1+x_2}{x_1+x_2+x_3} \cdot \frac{x_1+x_2+x_3}{x_1+x_2+x_3+x_4} + r_2 \frac{x_1+x_2+x_3}{x_1+x_2+x_3+x_4} + r_3 \tag{7.9}$$

〔Fig. 7.2〕 $y = x_1+x_2+x_3+x_4$ のプロセス・グラフ



(7.9) 式の右辺第1項の $x_1+x_2+x_3$ を消去して $y = x_1+x_2+x_3+x_4$ を両辺に掛ければ，最終結果 y の絶対誤差は次のようになる。

$$E_y = r_1 \cdot (x_1+x_2) + r_2(x_1+x_2+x_3) + r_3(x_1+x_2+x_3+x_4) \tag{7.10}$$

有効桁数 t 桁のコンピューターでは，切り捨て誤差 r_1, r_2, r_3 の限界は

$$|r_i| \leqq 10^{-t+1} \quad (i=1, 2, 3) \tag{7.11}$$

であるから，(7.10)式に代入して

$$|E_y| \leqq (3x_1+3x_2+2x_3+x_4) \times 10^{-t+1} \tag{7.12}$$

となる。x_1, x_2, x_3, x_4 が正数であるから，y の絶対誤差 E_y は

$$x_1 \geqq x_2 > x_3 > x_4 \tag{7.13}$$

の時最大となり

$$x_1 \leqq x_2 < x_3 < x_4 \tag{7.14}$$

の時最小となる。

一般に，n 個の正数 $x_1, x_2, x_3, \cdots, x_n$ を加え合わせる場合について考えてみよう。

$$y = x_1 + x_2 + x_3 + \cdots + x_n \tag{7.15}$$

プロセス・グラフを使って(7.15)式の誤差の範囲を評価すると

$$|E_y| \leqq [(n-1)x_1 + (n-1)x_2 + (n-2)x_3 + \cdots + 3x_{n-2} + 2x_{n-1} + x_n] \times 10^{-t+1} \tag{7.16}$$

であることがわかる。

誤差 E_y は $x_1, x_2, \cdots, x_n$ の大小関係が

$$x_1 \leqq x_2 < x_3 < \cdots < x_n \tag{7.17}$$

の時に最小となることはもはや一目瞭然である。

言い換えれば，誤差の範囲を最小にするためには n 個の数値を小さいものから大きいものへと順番に加えていけばよい。

上のことから，次のような結論が得られたことになる。

3個以上の数値（正数データ）をコンピューターで加える時には，小さい数値から順番に加えると累積誤差は最小となり，大きい数値から順番に加えると累積誤差は最大になる。

読者は既に気付いた筈であるが，上の結論は代数学の常識に反する。代数学によれば加算では交換法則 $x_1+x_2 = x_2+x_1$ や結合法則 $(x_1+x_2)+x_3 = x_1+(x_2+x_3)$ が常に成立するが，コンピューターを用いるとこの常識は成り立たないのである。これはたびたび説明してきたように，コンピューターの中に記憶できる数値が有限だからである。加算の交換法則や結合法則は無限の有効桁数を持ったコンピューターの中でのみ真理なのである。

7.2 加算における累積誤差とその極小化（その2）

前節では大きさの違う3個以上の数値の加算について，誤差累積の法則をたしかめたが，今度は大きさがあまり違わない4個以上の正数値の加算について，もう1つの法則を導いておく。

大きさがほぼ同じ4つの正数値 x_1, x_2, x_3, x_4 を次のように表わそう。

$$\begin{aligned} x_1 &= w + d_1 \\ x_2 &= w + d_2 \\ x_3 &= w + d_3 \end{aligned} \tag{7.18}$$

$$x_4=w+d_4$$

これは，x_1, x_2, x_3, x_4 を共通値 w と微小値 d_1, d_2, d_3, d_4 に分解したことをあらわす。d_1, d_2, d_3, d_4 は w に比べて充分に小さいとする。従って，4つの数値 x_1, x_2, x_3, x_4 はいずれも w に極めて近いという想定である。

この4つの数値を前節で行なった方法で順番に加えるとすれば，切り捨て誤差の累積は(7.16)式によって次の範囲になる。

$$\begin{aligned}|E_y|&\leqq[3(w+d_1)+3(w+d_2)\\&\quad+2(w+d_3)+(w+d_4)]\times10^{-t+1}\\&=[9w+3d_1+3d_2+2d_3+d_4]\times10^{-t+1}\end{aligned}\qquad(7.19)$$

d_i は微小であるから捨象すれば (7.19) 式は次のように考えてもさしつかえないであろう。

$$|E_y|\leqq9w\times10^{-t+1}\qquad(7.20)$$

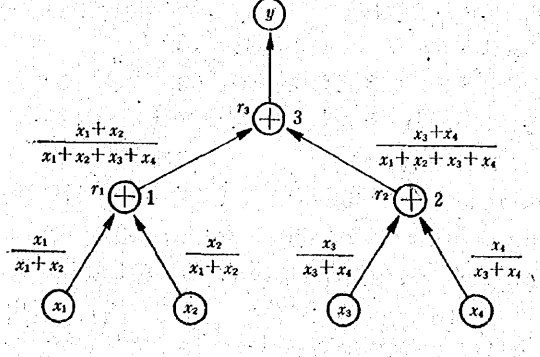
実際には x_1, x_2, x_3, x_4 を加えるのであって w を4回加えるわけではないが，誤差の範囲は(7.20)式で近似してよいであろう。この場合，x_i は殆んど同じ大きさであるから逐次加えていく順序は入れ替っても累積誤差は変らないと考える。

以上が x_1, x_2, x_3, x_4 を順番に加える場合であるが，次に同じ4つの数値を

$$y=(x_1+x_2)+(x_3+x_4)\qquad(7.21)$$

の方法で加えるという特殊な方法を試みる。実は，この方法をとると誤差の限界は小さくなるのである。このことをプロセス・グラフ〔Fig. 7.3〕で調べてみよう。

〔Fig. 7.3〕 $y=(x_1+x_2)+(x_3+x_4)$ のプロセス・グラフ



〔Fig. 7.3〕では加算ステップの記号⊕の横に演算の順番 1, 2, 3 を附してある。各ステップで発生する切り捨て相対誤差を r_1, r_2, r_3 とすると，全体の相対誤差は

$$\frac{E_y}{y}=r_1\frac{x_1+x_2}{x_1+x_2+x_3+x_4}+r_2\frac{x_3+x_4}{x_1+x_2+x_3+x_4}+r_3\qquad(7.22)$$

である。これまでに度々述べたように，r_1, r_2, r_3 はいずれも最大限 10^{-t+1} であるから，相対誤差の限界は次のようになる。

$$\left|\frac{E_y}{y}\right|\leqq\left(\frac{x_1+x_2}{x_1+x_2+x_3+x_4}+\frac{x_3+x_4}{x_1+x_2+x_3+x_4}+1\right)\times10^{-t+1}\qquad(7.23)$$

両辺に $y=x_1+x_2+x_3+x_4$ をかけて整理すれば絶対誤差は次のようになる。

$$|E_y|\leqq(2x_1+2x_2+2x_3+2x_4)\times10^{-t+1}\qquad(7.24)$$

(7.24)式を(7.12)式と比べてみればその意味が重大であることがわかるであろう。4個の数値を加え合わせる場合には〔Fig. 7.3〕に図示したようにトーナメント式に2つずつの和を作る操作をくり返せば，誤差の限界を示す(7.24)式の右辺の（ ）の中のウェイトはすべて“2”になる。

トーナメント加算の誤差限界の方が逐次加算のそれより小さいということを4個の数値の加算 ($y=x_1+x_2+x_3+x_4$) の場合について確かめてみよう。(7.24) 式に(7.18)式を代入すると，

$$\begin{aligned}|E_y|&\leqq(2x_1+2x_2+2x_3+2x_4)\times10^{-t+1}\\&=[2(w+d_1)+2(w+d_2)+2(w+d_3)\\&\quad+2(w+d_4)]\times10^{-t+1}\\&=[8w+2d_1+2d_2+2d_3+2d_4]\times10^{-t+1}\end{aligned}\qquad(7.25)$$

微小部分 d_i を捨象すれば

$$|E_y|\leqq8w\times10^{-t+1}\qquad(7.26)$$

となる。(7.26)式の誤差限界は，明らかに(7.20)式の限界より小さい。つまり，逐次加算の誤差限界よりもトーナメント加算の誤差限界の方が小さいことが確かめられた。

次に，n 個の正数 $x_1, x_2, \cdots, x_n$ を加え合わせるトーナメント加算について考える。この場合，$n=4, 8, 16, 32, \cdots$ のようにちょうど 2^k になっていてトーナメント構造が完全な形になる場合とそれ以外の場合では一般化の方法が異なる。ここでは簡単のために $n=2^k$ の場合，つまり完全な形のトーナメント加算について考えてみよう。

プロセス・グラフは省略するが，$n=8=2^3$ の場合の絶対誤差は

$$|E_y|\leqq(3x_1+3x_2+\cdots+3x_8)\times10^{-t+1}\qquad(7.27)$$

$n=16=2^4$ の場合の絶対誤差は

$$|E_y|\leqq(4x_1+4x_2+\cdots+4x_{16})=10^{-t+1}\qquad(7.28)$$

同様に $n=2^k$ の場合の絶対誤差は

$$|E_y|\leqq(kx_1+kx_2+\cdots+kx_n)=10^{-t+1}\qquad(7.29)$$

となる。

これは，(7.16)式から逐次加算の絶対誤差の限界

$$|E_y| \leqq [(n-1)x_1+(n-1)x_2+(n-2)x_3+\cdots+x_n] \times 10^{-t+1}$$
$$=[(2^k-1)x_1+(2^k-1)x_2+(2^k-2)x_3+\cdots+x_n] \times 10^{-t+1} \qquad (7.30)$$

にくらべてはるかに小さい。

$n=128=2^7$ の時は，逐次加算方式はトーナメント加算方式のおよそ9倍の誤差を生む。データが1024個ならば約50倍以上の誤差を生む。実際の経済分析計算では500回や1000回の加算はむしろ頻繁にあらわれるから，この違いは決して無視できない。

以上の分析結果を要約しておこう。

4個以上の数値を加える時は，1つずつ順番に加えていく逐次加算よりは，2個の数値の和を作っていくトーナメント加算の方が累積誤差ははるかに小さい。

この法則もまた，代数学の常識に反する。代数的には

$$y=x_1+x_2+x_3+\cdots\cdots+x_n$$
$$=(x_1+x_2)+(x_3+x_4)+\cdots\cdots+(x_{n-1}+x_n)$$
$$=\{(x_1+x_2)+(x_3+x_4)\}+\cdots$$
$$+\{(x_{n-3}+x_{n-2})+(x_{n-1}+x_n)\} \qquad (7.31)$$

は常に真理であるが，有効桁数が限られたコンピューターの世界では常に真理ではないのである。

8. その他の誤差累積の法則と対策

8.1 減算における誤差累積の対策

2個の数値 x_1, x_2 の間の減算

$$y=x_1-x_2 \qquad (8.1)$$

における誤差累積は，5.3 節で示したように次の式で評価できる。

$$\frac{E_y}{y}=\frac{x_1}{x_1-x_2}\cdot\frac{E_{x_1}}{x_1}-\frac{x_2}{x_1-x_2}\cdot\frac{E_{x_2}}{x_2} \qquad (8.2)$$

(8.2) 式をみればすぐ気付くことだが，x_1 と x_2 が非常に近い値の場合には面倒な問題が起きる。つまり，

$\frac{x_1}{x_1-x_2}$ と $\frac{x_2}{x_1-x_2}$ が共に非常に大きな値になってしまう。

このことを具体例でみよう。有効桁数 t が4桁のコンピューターで2つの数値 $x_1=0.6531$, $x_2=0.6530$ の間の減算を行なう。いま，$x_1=0.6531$ は切り捨てを行なった後の数値であるとする。従って x_1 は次の範囲の相対誤差を持っていると考えられる。

$$\frac{E_{x_1}}{x_1} \leqq \frac{0.0001}{0.6531}=0.00015311 \fallingdotseq 0.015\% \qquad (8.3)$$

簡単のために x_2 は切り捨て誤差を持たないとすれば，差の計算（$y=x_1-x_2$）によって生ずる相対誤差の限界は

$$\frac{E_y}{y} \leqq \frac{0.6531}{0.6531-0.6530}\times 0.00015 - \frac{0.6530}{0.6531-0.6530}\times 0$$
$$\fallingdotseq 1.0=100\% \qquad (8.4)$$

となる。つまり，わずか0.015%の相対誤差を持った数と全く誤差を持たない数との間の差を計算したのに，答は驚くべきことに最大限100%の誤差を持つ可能性がある。

このことから，次のことが言える。

コンピューター計算では，値が近い数値の差の計算はできるだけ避ける必要がある。

筆者等は，コンピューターの有効桁数との兼ね合いで，差の計算が無意味な答となってしまう限界を判定する方法を解明して，それに基づいて対策をたてることができると考えているが，本稿ではこの問題にこれ以上は立ち入らない。

8.2 乗算における誤差の累積とその対策

第4章で示したように，乗算では，個々の誤差にかかる累積のウェイトは1である。そのため，誤差の累積は加算，減算の時ほどには加速度的な拡大はしない。従って，乗算自体については，誤差累積を抑えるために対策を特に深刻に考える必要はない。

ここでは，問題点が比較的はっきりしているケースとして下記のような計算問題をとり上げておく。

$$y=(x_1-x_2)\cdot x_3 \qquad (8.5)$$

(8.5) 式の問題は，実際に2通りの方法で計算できる。第1の方法は(8.5)式の通り (x_1-x_2) を先に実行して，しかる後に x_3 を掛ける方法である。第2の方法は

$$y=x_1\cdot x_3-x_2\cdot x_3 \qquad (8.6)$$

を計算する方法である。

もし x_1 と x_2 が非常に近い数であれば前節で述べたように (x_1-x_2) の部分が非常に大きな誤差を生むから，直感的にも (8.5) 式の方式よりは (8.6) 式の方式の方がよさそうである。この直感は以下のようにプロセス・グラフを使いながら確かめてみると正しくない。

〔Fig. 8.1〕は $y=(x_1-x_2)\cdot x_3$ のプロセス・グラフであり，これによると累積誤差の範囲は (8.7) 式のよう

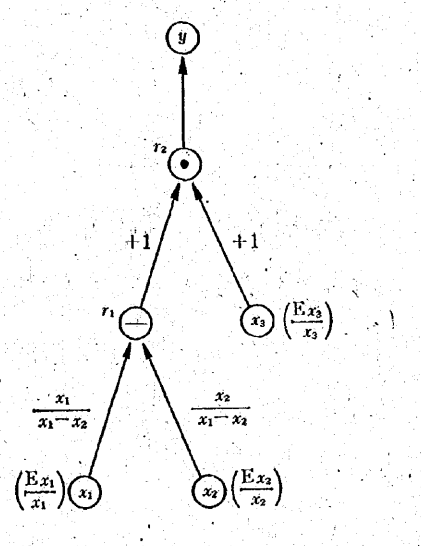
になる。

$$\frac{E_y}{y}=\frac{x_1}{x_1-x_2}\left(\frac{E_{x_1}}{x_1}\right)-\frac{x_2}{x_1-x_2}\left(\frac{E_{x_2}}{x_2}\right)+r_1+\frac{E_{x_3}}{x_3}+r_2$$

$$\left|\frac{E_y}{y}\right|\leqq\left(\frac{x_1}{x_1-x_2}-\frac{x_2}{x_1-x_2}+3\right)\times10^{-t+1}$$

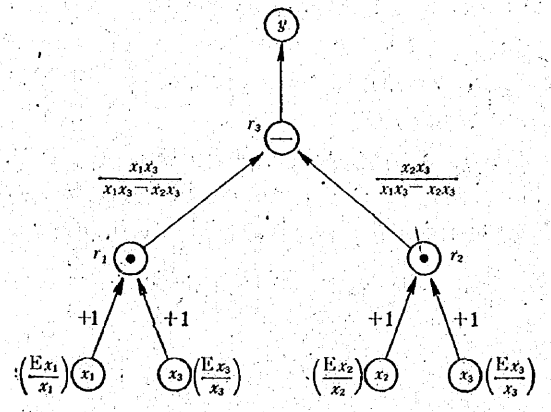
$$=4\times10^{-t+1} \qquad (8.7)$$

〔Fig. 8.1〕 $y=(x_1-x_2)\cdot x_3$ のプロセス・グラフ



一方，(8.6) 式のように先に掛け算を行なう場合は，〔Fig. 8.2〕のプロセス・グラフであらわせる。この場合の累積誤差の範囲は，(8.8) 式のようになる。

〔Fig. 8.2〕 $y=x_1\cdot x_3-x_2\cdot x_3$ のプロセス・グラフ



$$\frac{E_y}{y}=\left\{\left(\frac{E_{x_1}}{x_1}\right)+\left(\frac{E_{x_3}}{x_3}\right)+r_1\right\}\cdot\frac{x_1x_3}{x_1x_3-x_2x_3}$$

$$-\left\{\left(\frac{E_{x_2}}{x_2}\right)+\left(\frac{E_{x_3}}{x_3}\right)+r_2\right\}\cdot\frac{x_2x_3}{x_1x_3-x_2x_3}+r_3$$

$$\left|\frac{E_y}{y}\right|\leqq\left\{\left(\frac{x_1x_3}{x_1x_3-x_2x_3}-\frac{x_2x_3}{x_1x_3-x_2x_3}\right)\times3+1\right\}$$

$$\times10^{-t+1}=4\times10^{-t+1} \qquad (8.8)$$

(8.7) 式と (8.8) 式を比べれば，次のように結論できる。

$(x_1-x_2)\cdot x_3$ と $(x_1x_3-x_2x_3)$ はどちらの計算方法をとっても誤差の限界は全く同じである。

同様に $(x_1+x_2)\cdot x_3$ の計算も，誤差の限界は全く同じである。

$$y=(x_1+x_2)\cdot x_3 \qquad (8.9)$$

〔Fig. 8.3〕は (8.9) 式の通りに先に和を計算してから掛け算をする場合のプロセス・グラフであり，〔Fig. 8.4〕は

$$y=x_1x_3+x_2x_3 \qquad (8.10)$$

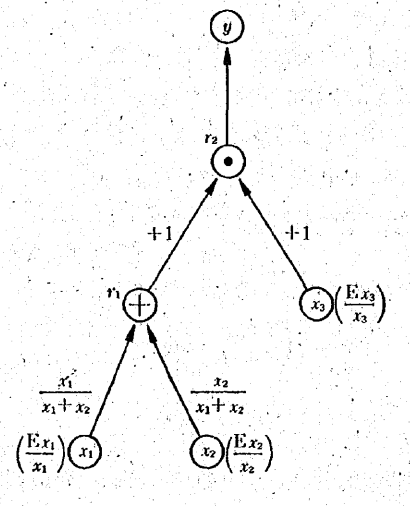
のように先に掛け算を行なう場合のプロセス・グラフである。〔Fig. 8.3〕の方式の累積誤差の範囲は次のようになる。

$$\frac{E_y}{y}=\frac{x_1}{x_1+x_2}\left(\frac{E_{x_1}}{x_1}\right)+\frac{x_2}{x_1+x_2}\left(\frac{E_{x_2}}{x_2}\right)+r_1$$

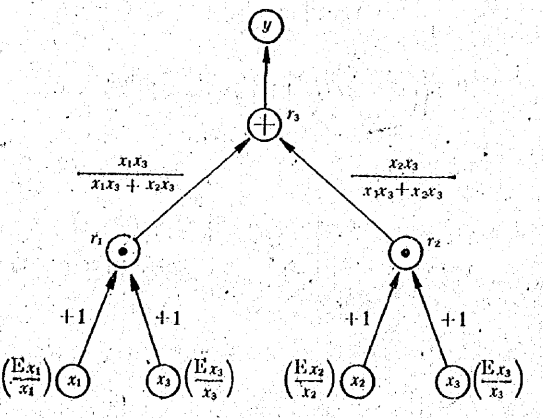
$$+\left(\frac{E_{x_3}}{x_3}\right)+r_2$$

$$\left|\frac{E_y}{y}\right|\leqq4\times10^{-t+1} \qquad (8.11)$$

〔Fig. 8.3〕 $y=(x_1+x_2)\cdot x_3$ のプロセス・グラフ



〔Fig. 8.4〕 $y=x_1x_3+x_2x_3$ のプロセス・グラフ



一方〔Fig. 8.4〕の方式の累積誤差の範囲は次のようになる。

$$\frac{E_y}{y}=\left\{\left(\frac{E_{x_1}}{x_1}\right)+\left(\frac{E_{x_3}}{x_3}\right)+r_1\right\}\cdot\frac{x_1x_3}{x_1x_3+x_2x_3}+\left\{\left(\frac{E_{x_2}}{x_2}\right)+\left(\frac{E_{x_3}}{x_3}\right)+r_2\right\}\frac{x_2x_3}{x_1x_3+x_2x_3}+r_3$$

$$\left|\frac{E_y}{y}\right|\leqq 4\times10^{-t+1} \qquad (8.12)$$

つまり，$(x_1+x_2)\cdot x_3$ と $(x_1x_3+x_2x_3)$ はどちらの計算方法をとっても誤差の限界は全く同じである。

8.3 除算における誤差の累積とその対策

除算の誤差累積の様子は，全般的には判明していない。ただ，前節の乗算の場合について $(x_1-x_2)\cdot x_3$ や $(x_1+x_2)\cdot x_3$ について若干の性質が判明したのと同程度のことはわかっている。

まず，

$$y=(x_1-x_2)/x_3 \qquad (8.13)$$

$$y=x_1/x_3-x_2/x_3 \qquad (8.14)$$

の2つの計算の精度を比較してみよう。

(8.13) 式の計算のプロセス・グラフは〔Fig. 8.5〕に，また(8.14) 式の計算のプロセス・グラフは〔Fig. 8.6〕になる。

(8.13) 式の方法の場合，累積誤差の限界は

$$\frac{E_y}{y}=\frac{x_1}{x_1-x_2}\cdot\left(\frac{E_{x_1}}{x_1}\right)-\frac{x_2}{x_1-x_2}\left(\frac{E_{x_2}}{x_2}\right)+r_1-\left(\frac{E_{x_3}}{x_3}\right)+r_3$$

$$\left|\frac{E_y}{y}\right|\leqq 2\times10^{-t+1} \qquad (8.15)$$

となるのに対して，(8.14)式の方法の累積誤差の限界は

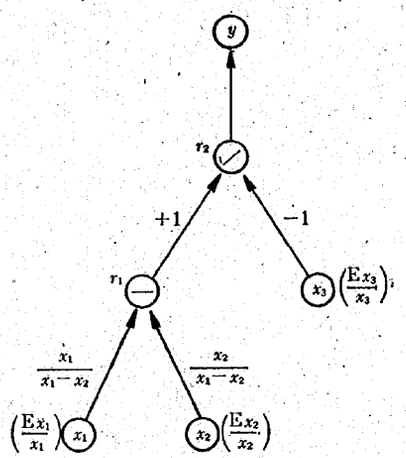
$$\frac{E_y}{y}=\left\{\left(\frac{E_{x_1}}{x_1}\right)-\left(\frac{E_{x_3}}{x_3}\right)+r_1\right\}\frac{x_1/x_3}{x_1/x_3-x_2/x_3}=\left\{\left(\frac{E_{x_2}}{x_2}\right)-\left(\frac{E_{x_3}}{x_3}\right)+r_2\right\}\frac{x_2/x_3}{x_1/x_3-x_2/x_3}+r_3$$

$$\left|\frac{E_y}{y}\right|\leqq 2\times10^{-t+1} \qquad (8.16)$$

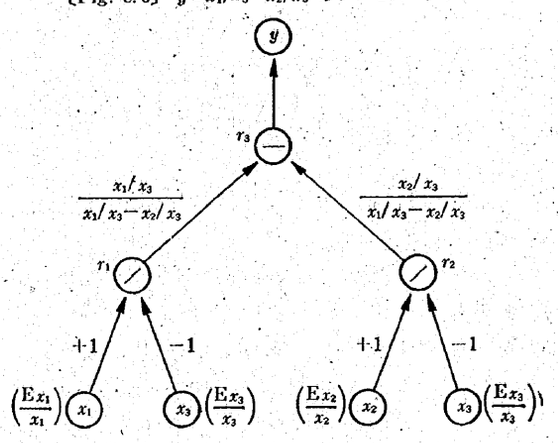
となる。以上から判明したことを要約すれば，

$(x_1-x_2)/x_3$ と $x_1/x_3-x_2/x_3$ はどちらの方式で計算しても誤差の限界は同じである。

〔Fig. 8.5〕 $y=(x_1-x_2)/x_3$ のプロセス・グラフ



〔Fig. 8.6〕 $y=x_1/x_3-x_2/x_3$ のプロセス・グラフ



同様にして

$$y=(x_1+x_2)/x_3 \qquad (8.17)$$

$$y=x_1/x_3+x_2/x_3 \qquad (8.18)$$

の2つの方式を比較する。従来と同様に〔Fig. 8.7〕，〔Fig. 8.8〕のプロセス・グラフを用いながら累積誤差の限界を計算すると (8.17) 式の方式については

$$\frac{E_y}{y}=\frac{x_1}{x_1+x_2}\left(\frac{E_{x_1}}{x_1}\right)+\frac{x_2}{x_1+x_2}\left(\frac{E_{x_2}}{x_2}\right)+r_1-\left(\frac{E_{x_3}}{x_3}\right)+r_2$$

$$\left|\frac{E_y}{y}\right|\leqq 2\times10^{-t+1} \qquad (8.19)$$

となり，(8.18) 式の方式については

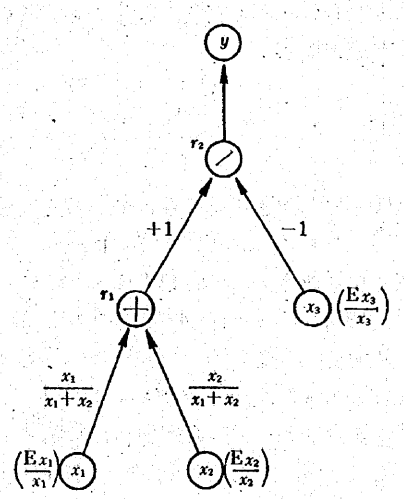
$$\frac{E_y}{y}=\left\{\left(\frac{E_{x_1}}{x_1}\right)-\left(\frac{E_{x_3}}{x_3}\right)+r_1\right\}\frac{x_1/x_3}{x_1/x_3+x_2/x_3}+\left\{\left(\frac{E_{x_3}}{x_2}\right)-\left(\frac{E_{x_3}}{x_3}\right)+r_2\right\}\frac{x_2/x_3}{x_1/x_3+x_2/x_3}+r_3$$

$$\left|\frac{E_y}{y}\right|\leqq 2\times10^{-t+1} \qquad (8.20)$$

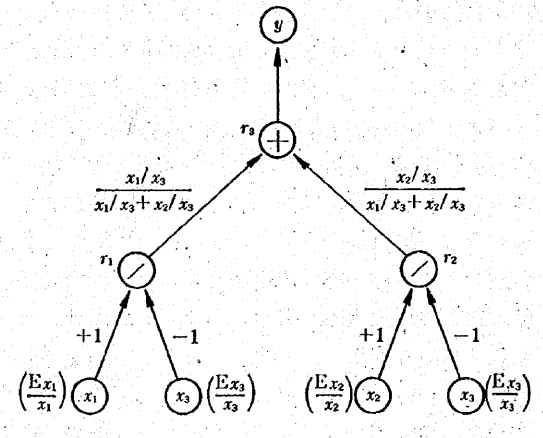
となって，どちらの方式を採用しても累積誤差の限界

は全く同じであることが判明する。

〔Fig. 8.7〕 $y=(x_1+x_2)/x_3$ のプロセス・グラフ



〔Fig. 8.8〕 $y=x_1/x_3+x_2/x_3$ のプロセス・グラフ



9. コンピューターによる分析計算の注意事項

以上の8章にわたる分析から判明したことをとりあえず要約しておきたい。筆者等はこの後，同じ誤差分析の方法を適用して回帰分析等の分析計算の精度問題を検討するつもりであるが，その前に，今までの分析からだけでもかなり有益な結論が得られたと思う。

第1に，コンピューターの有効桁数（本稿の分析では一貫して t の記号を用いてきた）が大きいほど切り捨て誤差や計算誤差の累積は少ない。実際には，筆者等が実験に用いたIBM-1620のようなvariable length machineは殆んどないから，決められた長さのプレシジョンの2倍のダブル・プレシジョンで計算をするのが一つの対策になる。本稿第4章の分析で示したように一般に，n 進法のコンピューターで有効桁数が t 桁の時，切り捨ての相対誤差は

$$\left|\frac{\varepsilon_X}{X}\right| \leqq n^{-t+1} \qquad (9.1)$$

であるから，ダブル・プレシジョンにすれば誤差の限界は

$$\left|\frac{\varepsilon_X}{X}\right| \leqq n^{-2t+1} \qquad (9.2)$$

となって，シングル・プレシジョンの場合の n^t 分の1に急速に縮小する。

第2に，コンピューター内部の rounding（まるめの操作）は，一般の市販コンピューターでは「切り捨て」が普通であるが，もし「四捨五入」や「偶捨奇入」のような symmetric rounding をすれば，誤差の限界は半分に縮小する。

第3に，3個以上の数値の加算は，数値の小さいものから順番に加えれば，累積誤差が最小になる。反対に数値の大きいものから順番に加えれば，累積誤差は最大になる。従って，平均値，分散等のモーメントを計算する際に起る大量のデータの加算は，できれば小さい数値から順にあらかじめソートした上で行なえば誤差累積を最小限にくい止めることができる。

第4に，4個以上の比較的大きさが同じ位の数値の加算は，単純に逐次的に加えていくよりは，2個ずつの和を作るトーナメント式に演算をくり返す方が誤差累積ははるかに小さくてすむ。

その他，8章で調べた様々の特殊な結論を得たが，これらは，今後，より一般的な対策へと拡張していく必要があろう。

鳥居 泰彦（経済学部助教授）
新井 益洋（産業研究所助手）

〔注〕 以上の説明に用いたプロセス・グラフの方法は，McCracken と Dorn によって下記の文献ではじめて提案されたものである。前号で文献リストに掲載しなかったのでここに追記する。

Deniel D. McCracken, William S. Dorn, *Numerical Methods and FORTRAN Programming* (John Wiley & Sons, 1964).