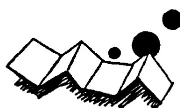
解 説



数学ソフトウェアの現状と問題点†

二 宮 市 三††

1. はじめに

数学ソフトウェア (Mathematical Software) という言葉が初めて公式に使われたのは，1970年 Purdue 大学で開催されたシンポジウム[1)] のタイトルとしてであった．言葉の元来の意味からいえば，ソート，マージ，数式処理などのソフトウェアを包含してもよいわけであるが，そもそもの使われ方から一貫して数値計算ソフトウェア (Numerical Software) を意味している．数値計算ソフトウェアの中には，統計解析パッケージや特殊の分野専用の応用ソフトウェアなどのものがあるが，ここではそういうものの共通的な素材となっている最も基本的なソフトウェアについて考えることとする．

さて，本解説では電子計算機による数値計算の高度化と大型化に伴い，ますますその重要性を高めつつある数学ソフトウェアについて，性能，評価，選択，開発，流通，教育などの面からその現状を展望し，その中から問題点を探り，将来への指針としたいと考える．

2. 標準関数ソフトウェア

数学ソフトウェアの中でも最も基本的なものは標準関数ソフトウェアであろう．一部には標準関数をハードウェア化しようとする研究もあり，また CORDIC のようにハードウェア化を実行している例もあるが，大勢は圧倒的にソフトウェアであり，今後もしばらくはこの情勢がつづくものと考えられる．

標準関数ソフトウェアは，それ自身非常に利用頻度の高いものであると同時に，特殊関数を初めとする他の多くのソフトウェアによって引用されているので，その性能の良否の影響する所は甚大である．そのために，他のほとんどのソフトウェアが FORTRAN などの高級言語で書かれているのに対して，標準関数ソフトウェアは，それぞれの機械のハードウェアの特性を利用しつくすためにもっぱらアセンブリ言語で書かれている．これは一面当然のことではあるが，その反面標準関数というものを一般の計算機利用者から縁遠いものにしていることも事実である．彼らは初歩教育のときからそのようにしつけられているためか，標準関数を空気や水のように全くその存在を意識しないか，あるいはブラックボックスとして，漠然とではあるが絶対の信頼を抱いているかどちらかである．しかし，人が作ったソフトウェアである以上，絶対ではありえないし，いろいろな困難や研究者の挑戦を刺激するような興味ある問題点も存在する．

2.1 標準関数の精度

標準関数の性能の中で最も重要なものはその精度である．一般に標準関数は，単精度用であれ，倍精度用であれあるいは，四倍精度用であれ，使用桁数一杯の精度を出すことを狙いとして作られている．これはいいかえれば，相対誤差規準に基づく精度を問題にしているのであり，使用桁数を10進 s 桁とすれば，最大相対誤差を 10^{-s} 以下に抑えるということである．浮動小数点表示の建前からすればこれは当然の要請であるが，その達成は関数の種類や，引数の値によっては必ずしも可能ではない．具体的にいうと，平方根や指数関数では比較的に容易であるが，たとえば1の附近の対数関数や π の附近の正弦関数の場合には原理的に不可能である．なぜなら，このような零点の附近では，関数値の計算には引数と零点の差が必然的に介入するのであるが，この差は避けられない桁落ちのために貧弱な相対精度をもつ値でしかありえないからである．したがって，こういう場合には高々絶対精度しか保証することはできない．それでは，相対精度もしくは絶対精度のどちらかは常に保証できるかというと，そうでもない．正接関数の極の附近では，近似式の計算での打切り誤差や丸めの誤差が大きな微係数によって拡大され，引数が極に近づくにつれて，相対誤差も絶対

† Present State and Issue of Mathematical Software by Ichizo NINOMIYA (Department of Information Science, Faculty of Engineering, Nagoya University).

†† 名古屋大学工学部情報工学科

誤差もいくらでも——勿論ある範囲内で——大きくなってしまう．

標準関数の精度の原理的な困難について，やや抽象的に述べてきたが，これからは話題を絞ってより現実的に話を進めよう．

(a) 零点としての原点附近での精度

零点の附近では相対精度を保証することはできないと述べたが，原点が零点である場合は例外である．この場合には，零点への近さは引数自身によって純粋に，いわばむき出しの姿で表わされているので，零点の位数を k とするとき $f(x)=x^k\cdot g(x)$ の形の近似式を用いれば，桁数一杯の相対精度を出すことは容易である．標準関数では，正弦，正接，双曲線正弦，双曲線正接及びそれらの逆関数がそういう実例である．平方根や立方根は原点の性質も計算法も他のものとは異質であるから同じ部類には入らない．上の諸関数の中で，三角関数とその逆関数には問題がない．$f(x)=x\cdot g(x)$ という形の近似式を使うことが自然であり，これ以外の近似式を使うきっかけが全くないからである．ところが双曲線関数と逆双曲線関数——これらは標準関数ではないが，ついでに述べておく——は，

$$\sinh x=(e^x-e^{-x})/2,$$
$$\tanh x=(e^x-e^{-x})/(e^x+e^{-x}),$$
$$\sinh^{-1}x=\log(x+\sqrt{1+x^2}),$$
$$\tanh^{-1}x=\log\{(1+x)/(1-x)\}/2$$

というふうに指数関数と対数関数によって簡単に表わされるので，うっかりするとこのような式を原点の近くでもそのまま使おうとする誘惑におち入る危険がある．いうまでもなく上式はすべて原点の附近で，桁落ちによる精度の損失をまぬがれないので，桁落ちが無視できる程度に原点から離れた点以外では使ってはならない．三角関数も含めてこれら奇関数の原点附近での計算に必要な近似式は，たとえばハート[2] の本に，種々の範囲，種々の精度にわたって豊富に載せられているのでほとんどの場合そっくりそのまま利用できる近似式が発見できる．しかし注意しなければならないのは，近似式の中には絶対誤差規準に基づくものも混っていることで，必ず相対誤差規準に基づいたものであることを確認すべきである．

標準関数では余接やガンマ関数が実例であるが，原点が零点ではなくて極である場合にも，その位数を k とするとき $f(x)=g(x)/x^k$ の形の相対誤差規準に基づく近似式を用いれば，極の附近でも十分な相対精度が達成される．

(b) 区間縮小

どんな標準関数でも，これを精度よく能率よく計算するためには，対称性や加法定理などを利用して引数をある一定の狭い標準区間内の値に還元することが必要である．このことを区間縮小 (Range Reduction) という．標準区間は，要求精度，計算速度及びプログラムの大きさを考え合わせて適当に選定されねばならない．標準区間内の計算は，多くは既成の理論や近似式による多項式または有理関数の類型的な計算であってあまり問題はない．これに反して，区間縮小の操作は，実数の表現型式や機械の命令構成などハードウェアに直接依存する部分が多いので一般論が成立せず，困難ではあるが同時にプログラマの腕の見せ所でもある．

さて，区間縮小は関数の精度にどのように影響するであろうか．説明を簡単にするために実数は2進の浮動小数点型式で表わされているものとする．すなわち，実数 x は指数部 e と仮数部 m とによって

$$x=2^e\cdot m \tag{2.1}$$

と表わされる．ただし e は整数，m は $1/2\leq|m|<1$ なる小数である．まず，対数関数を考えよう．自然対数は2底の対数の定数倍

$$\log_e x=\log_e 2\cdot\log_2 x$$

であるから，2底の対数を考えればよい．

$$\log_2 x=e+\log_2 m \tag{2.2}$$

であるから，関数値は標準区間での値 $\log_2 m$ と区間縮小のための附加項 e との和となる．この場合 $0>\log_2 m\geqq-1$，であり，e は整数であるから，一般に，そして特に大きな引数に対しては，附加項の方が大きく，しかもこの項は誤差なしで計算できるので，区間縮小による精度の低下の問題はほとんどない．同じ事情が，やや複雑であるが逆正接の場合にも成り立つ．

ところが，指数関数や三角関数の場合は事情が異なる．まず，区間縮小を能率的に行うためには，たとえば，

$$e^x=2^y,\quad y=x\cdot\log_2 e, \tag{2.3}$$

$$\sin x=\sin\frac{\pi}{2}y,\quad y=x\cdot\frac{2}{\pi} \tag{2.4}$$

というような変形が必須である．今度は y は整数部分 N と小数部分 f とによって

$$y=N+f \tag{2.5}$$

と分解され，それぞれ

$$e^x=2^y=2^N\cdot 2^f, \tag{2.6}$$

$$\sin x=\sin\frac{\pi}{2}y=\pm\begin{cases}\sin\frac{\pi}{2}f\\ \cos\frac{\pi}{2}f\end{cases} \quad (2.7)$$

となる．どちらの場合も，小数部分fを用いて計算される標準区間での関数値が精度を左右する．区間縮小によって抜き取られる引数の上位桁のNは，指数関数の場合は結果の指数部として残るが，三角関数の場合には何らの寄与を残さない．このようにして，区間縮小によって引数の整数部分の桁数程度の精度の損失があるということになる．指数関数の場合はこの損失が問題になり始めるよりも先にあふれの限界——通常数百程度——がきてしまうので表面化しないが，三角関数の場合はそういうことがないので，この悩みは深刻である．実際の計算では引数はほとんど常に誤差をもっているので，大きな引数に対する指数関数や三角関数の計算には原理的な困難があるといわなければならない．それはともかくとして，区間縮小による損失以上の損失を許さないためには，(2.3)，(2.4) での変換定数 $\log_2 e$, $2/\pi$ の表現とその乗算を使用桁数よりも大きな桁数で行わなければならない．単精度については倍精度を使えば問題はないが，倍精度以上の場合にどうするかは，倍精度以上の標準関数の精度と，倍精度よりも多い桁数を使う特別の計算のコストをどう考えるかによって決められるべきである．

上に述べた問題に関連して考えられるのは実数の冪乗 x^y の計算である．これは

$$x^y=2^{y\cdot\log_2 x} \quad (2.8)$$

として計算されるが，肝要なことは (2.2) の加算と，$y\cdot\log_2 x$ の乗算を，要求された精度を超える精度で行うことである．これは既成の2底の対数ルーチンや2底の指数関数ルーチンを引用していたのでは不可能であり，どうしてもこれらのルーチンと同等のものを内蔵した，自己完結的な冪乗専用ルーチンを書かなければならない．

(c) 早期中断

数値計算で，その最終結果は実数の制限範囲内の数であるのに，計算の途中で範囲外の数が生じて計算を中断しなければならなくなることを早期中断 (Premature Breakdown) という．この現象の典型的な例は複素数の除算である．$z=x+iy$ を $w=u+iv$ で割る場合に，常識的に

$$z/w=\{ux+vy+i(uy-vx)\}/(u^2+v^2) \quad (2.9)$$

とすると，たとえば$|u|$と$|v|$の一方または両方が実数の最大値の平方根より大きい場合には，wの如何にかかわらず u^2+v^2 の計算でオーバフローが生ずる．アンダフローの場合もほぼ同様である．これに対しては，たとえば $|u|\geqq|v|$ のときは，

$$z/w=\{x+ry+i(y-rx)\}/(1+r^2),\quad r=v/u \quad (2.10)$$

とすれば，早期中断がほとんど完全に防止できるだけでなく，計算速度の点でも (2.9) に勝るという一石二鳥の効果を挙げることができる[3]．似たようなことが複素数の絶対値の計算で起る．この場合には

$$\sqrt{x^2+y^2}=\begin{cases}|x|\cdot\sqrt{1+(y/x)^2}, & |x|\geqq|y|\\ |y|\cdot\sqrt{1+(x/y)^2}, & |x|<|y|\end{cases} \quad (2.11)$$

が解決策である．上の二例よりは些細で影響が小さいが，双曲線正弦（余弦でも同じ）でも早期中断が起る．e^x に比べて e^{-x} が無視できるような十分大きいxに対して

$$\sinh x\fallingdotseq\cosh x\fallingdotseq e^x/2 \quad (2.12)$$

となるが，この式を指数関数ルーチンを引用してこのまま適用すると，指数関数の許容制限 x_m がそのままこれらの関数の許容制限となる．しかし実際は $x_m+\log_e 2$ までは計算できるはずである．それを実行するためには

$$e^x/2=e^{x-\log_e 2} \quad (2.13)$$

とすればよいように考えられるが，x_m は数百のオーダであり，$\log_e 2=0.69314\cdots$ であるから $\log_e 2$ の下位の桁が十分 $x-\log_e 2$ に入り切らず結果の精度が悪くなる．そこで次のようにする．2よりも少し大きく，$\log_e c$ が少ない桁数で正確に表わされ，$x-\log_e c$ が誤差なく計算できるような定数cを使い，

$$e^x/2=k\cdot e^{x-\log_e c},\quad k=2/c \quad (2.14)$$

と計算する[4]．これは実に巧妙な方法である．

2.2 標準関数の速度

標準関数の精度には桁数一杯という客観的な努力目標があるのに対して，速度の方にはそのようなものがないために精度に比して幾分なおざりにされているきらいがある．速度の向上を図るために留意しなければならない事項を列挙してみよう．

最も時間がかかる重要な部分は，標準区間での近似多項式または有理式の計算であろう．このような場合普通ならインデックスループを使うのが常識であるが，速度を生命とする標準関数ではインデックスループは禁物である．多少見掛けのステップ数は増しても，ループは使わないで直線的にコーディングしなければならない．

標準区間の幅を小さくして，その中での近似式の次数を低くすることは速度向上のために非常に効果がある．しかしその反面，区間縮小の手続きが複雑となるという逆効果もあるので無制限というわけにはゆかない．

区間縮小の部分は，引数のエラーチェック，引数の分解，大小比較，索表などの非計算的な処理から成っているので，あらゆるコーディングの技術を動員して極力時間短縮につとめなければならない．

関数コールに伴う，レジスタの退避復元などのオーバヘッドは，普通のサブルーチンコールの場合にはさほど問題にならないが，計算時間の短い標準関数の場合には，そのかなりの部分を占めるので最小限に止めねばならない．

最後に述べた考え方の線を徹底したものに，標準関数のインライン展開がある．これは，最近FORTRANなどのオプション（任意選択事項）として取り入れられたもので，標準関数をあたかも，絶対値や整数の実数化などの組込み関数のように扱い，その引用のたびにその場所に関数の目的プログラムを展開するものである．このようにすれば，プログラムの大きさは大きくなるが，関数コールによるオーバヘッドは完全に除去されるので，計算速度向上のために著効がある．しかしながら，他の点をも含めて，利用者に対する周知徹底は十分でなく，折角の名案もまだ効果を発揮するには到っていない．

2.3 16進計算機の標準関数

現在世界ならびにわが国の計算機で広く用いられている実数の表現法に16進法があることは周知の事実であろう．これは32ビットのデータを，先頭の1ビット（符号桁 s），次の7ビット（指数部 e）及び最後の24ビット（仮数部 m）に分けて，実数

$$x=(-1)^s\cdot 16^{e-64}\cdot m \qquad (2.15)$$

を表現しようとするものである．仮数部 m は指数部 e を調整して，常に

$$1/16\leqq m<1 \qquad (2.16)$$

の範囲にあるように正規化されている．したがって m の先頭の部分のビットパターンは次のようになる．

$1/16\leqq m<1/8$	0. 0 0 0 1 ×××× …
$1/8\leqq m<1/4$	0. 0 0 1 ××××× …
$1/4\leqq m<1/2$	0. 0 1 ×××××× …
$1/2\leqq m<1$	0. 1 ××××××× …

すなわち，実数の正味のビット数は値によって異なり，それぞれ21, 22, 23, 24となる．このような値によって精度が異なる実数を用いた場合の関数の精度をどのように考えるべきであろうか．大まかな考え方として，すべての場合において最悪の場合の21ビット分の精度があればよいとする考え方がある．たしかに普通の関数ならばそれでよいのかも知れないが，標準関数ともなれば話は別であって，関数値の正味ビット数に応ずる精度が要求されてもしかたがない．これは一見何でもないことのように見えるが実はそうではない．

正弦と余弦とを標準区間 $-1\leqq x\leqq 1$ での $\sin(\pi/2)x$ に還元する場合を考えて見よう．この場合近似多項式は

$$\sin\frac{\pi}{2}x=x(a_0+a_1x^2+a_2x^4+\cdots) \qquad (2.17)$$

の形で，a_0 は $\pi/2=1.570796\cdots$ の附近の値である．すなわち a_0 は最悪の21ビット数である．したがってこの式をこのままの形で使う限り，精度はいつも21ビット分しかない．ところが $\sin(\pi/4)x$ の場合には (2.17) と同様な近似式の係数 a_0 は $\pi/4=0.785398\cdots$ に近く，これは最良の24ビット数である．16進法では，このように近似式の良さを測る尺度として，精度と次数の外に重要な係数――多くは最低次項の係数――の正味ビット数という好ましくない因子が介入してくる．

上述の困難をもう少し一般化すると，16進法では計算の重要な段階でビット数の少ない数を出現させてはならないということになる．ニュートン法で平方根を計算する場合を考えよう．R_i を第 i 近似とすると，$\sqrt{x}$ を計算するためのニュートン法は

$$R_{i+1}=(R_i+x/R_i)/2 \qquad (2.18)$$

と書かれる．今，$\sqrt{x}$ はたとえば0.5と1.0の間の24ビット数であり，ニュートン反復の最終段階に入ったものとする．このとき R_i と x/R_i は共に $\sqrt{x}$ に近い24ビット数である．ところが R_i+x/R_i は1.0と2.0の間の21ビット数である．これを2で割った R_{i+1} は一応24ビット数ではあるが，最後の3ビットは常に0になっていて，21ビット分の精度しかない．1.0――一般には，仮数部 m が1/16の数――を通過する際の3ビットの断層がひき起すこの珍現象をそのまま放置している平方根ルーチンもまれではない．しかし多くの良心的な平方根ルーチンでは，最終段だけを (2.18) から

$$R_{i+1}=(x/R_i-R_i)/2+R_i \qquad (2.19)$$

に変更してこの困難を解消している．一般的に，16進法特有のこのような困難は綿密な注意と巧みな技術に

よって何とか克服されることが多い．しかしその際払われる努力は外の場所に応用のきかない偏ったものであって感心できない．要するに16進法は，少ないビット数で実用上差支えない程度に広い範囲の実数を表現しようとする無理から生じた産物であって，数値計算の本来の目的からは無縁の労苦を強いるという点で，ソフトウェア作者にとっても利用者にとっても好ましくない代物である．今後の計算機からは是非追放してもらいたいと切に願うものである．

3. 特殊関数ソフトウェア

特殊関数はその範囲が広汎であり，問題も多種多様であるので標準関数のように組織的に扱うことは困難である．ここでは思いつくままに問題点を拾い上げて行くことにする．

(a) 計算法

特殊関数の計算法としては種々のものが考えられる．理論的な冪級数，漸近級数及び連分数展開の打切りなどの直接的なものから，定積分の計算や逆関数の計算に見られる方程式の反復解法のような間接的なものまであるが，これらは近似式を作ったり，関数ソフトウェアの精度検査をしたりするときのマスタルーチン用の計算法としては許容できるが，生産的な，特に一変数関数のソフトウェア用の算法としては不適当である．どうしても，何らかの意味の最良性をもった多項式または有理式による近似式でなければならない．この点に関して現状はまだ満足すべき段階ではない．

(b) 領域分割

標準関数より以上に，特殊関数では定義領域の分割が必要であり，そして簡単な加法定理が存在しないために各小領域での別々の計算法が必要である．領域分割に対する原則は，部分領域での近似式の精度と計算量の一様性であろう．この中で，精度の一様性は厳格に守られなければならないが，計算量の一様性にはそれほどこだわらなくてもよい．なぜなら，現実の関数の引用での各部分領域の出現頻度は必ずしも一様ではないからである．

一般的な分割のパターンは，原点附近の冪級数領域と無限遠中心の漸近級数領域への分割であり，ときたまこの両者の中間領域がとられることがある．冪級数領域では近似式を作ることは容易であるが，漸近級数領域ではきわめて困難である．この困難を回避するために冪級数領域を不当に広くとり，その結果近似式が，桁落ちの激しい不安定なものになっている例を，ベッセル関数などに見受けることがある．実に，桁落ち防止による数値安定性の確立は，精度と速度に並んで関数ソフトウェアの作成上の要点であって，今後この点の認識を益々強化しなければならない．

(c) 新関数のすすめ

$\sin(\pi/2)x$，$\cos 2\pi x$ などの形は応用上しばしば現れるが，これらを標準関数で計算すると π を陽に書く面倒さとそのための過失の機会がつきまとう．SINHP(X)[5] と書くと $\sin(\pi/2)x$ が計算されるような関数を作る理由がここにある．さらに重要なことは，標準関数の精度についての議論で示したように，$\sin x$ よりは $\sin(\pi/2)x$ の方がむしろ数値計算的には基本的で，$\sin x$ は $\sin(\pi/2)x$ を経由して計算されるということである．したがって $\sin(\pi/2)x$ を計算するのに SIN(HP*X)——HP には $\pi/2$ が入っている——とすると実際は，SINHP(HP*X/HP) という二重手間の計算が行われる．以上の見地から，π を因数とする引数に対する三角関数ルーチンの新設と普及が切に望まれる．

原点附近で，定義式通り計算すれば桁落ちを免れないような特殊関数は多数存在する．既述の逆双曲線関数のほかに，$\log(1+x)$[5]，e^x-1[6] などはその典型的な例である．原点附近で特別の近似式を使って桁落ちを防止した，この種の関数ルーチンは十分にその存在理由がある．

階乗 $n!$ は n の増加と共に激しく増大しすぐにオーバフローとなってしまうので，有効な関数値は高々100個の程度である．したがってあらかじめ数表にしておいても差支えない[5]．このような数表関数の他の可能性としては，ゼータ関数 $\zeta(n)$，ベルヌーイ数 B_{2n} などが考えられる[6]．

特殊関数の中で最も需要の高いベッセル関数関係のソフトウェアはかなりよく整備され利用されている．しかし現実の利用の情況にはまだまだ問題がある．たとえば $J_0(x)$ は，単独で計算されることはあまりなく，$Y_0(x)$ が同時に必要であったり，$J_0(x)$，$J_1(x)$，$Y_0(x)$，$Y_1(x)$ が同時に求められたりする．このような要求に対して，これらの関数単独のための関数ルーチンを別々に使用していたのでは，その計算式の多くの共通部分が利用できないという無駄がある．同様な状況が楕円関数の計算にも見られる．利用者がその利用の特殊な情況に即した合理的なソフトウェアを注文し，作者がこれに即応するという態勢の確立は，現在の所まだ遠い夢でしかない．

4. 線型計算ソフトウェア

関数ソフトウェアとその他のソフトウェアを分ける特質はその確定性である．関数ソフトウェアではその引数のあらゆる場合に対する処置があらかじめ定められ，その結果の精度について断定的な保証が与えられる．これに反してその他のソフトウェアはその対象が，予見できない不確定因子を多く含んだ不特定多数であり，結果の精度についてせいぜいのところ希望的な粗い評価がえられるにすぎない．それでは，不確定ソフトウェアの出力結果の正しさをどのようにして確かめたらよいのであろうか．連立一次方程式などの特別の場合には，事前のあるいは事後の誤差評価法が研究され，実際にも役立つ場合があるようである．しかし，次のようなより一般的でより実際的な方法の方が効果がある．すなわち，同一の計算問題を，データを故意に少しずらして反復して結果をつき合せ，一致した桁の部分だけを信頼するという方法である．この方法は一部ソフトウェアとして自動化されている．しかしながら，どの方法にもせよ，絶えず精度検査を行うのは不経済であるので，そのような精度検査は間欠的に行うこととし，経常的には結果の，他の同様な結果との連続性による，経験的な検証に頼るより仕方がなさそうである．

さて，不確定ソフトウェアの中では最も確定度の高い線型計算ソフトウェアから始めよう．

4.1 連立一次方程式関係のソフトウェア

(a) ガウスの消去法と *LU*-分解法

連立一次方程式の解法は数値計算の中で学問的に最も成熟した分野であって種々の方法に対する判然とした評価が定着している．すなわち，一般の係数行列の場合には，部分軸選択——行交換による軸選択——を伴うガウスの消去法あるいはそれと数学的に同等な LU-分解法が最良の方法とされている．ここに LU-分解法というのは，係数行列 A——正確には A に軸選択のために必要な行交換を行ったもの——を，下三角行列 L と上三角行列 U によって

$$A=L\cdot U \tag{4.1}$$

と分解する方法をいう．ガウスの消去法は結果においては LU-分解法と同じであるが，ここで LU-分解法というのは始めから (4.1) の分解の存在を仮定した上でえられる方法のことであって，ガウスの消去法とは実際面でかなりの差がある．すなわち，ガウスの消去法では，係数行列の要素は消去の各段階ごとに少しずつ変形を受けるが，LU-分解法では，ガウスの消去法での変形を一つにまとめた積和の形の変形を一度受けるだけである．そのために，積和の計算を倍精度で行うことにより丸めの誤差を最小限に抑えることができるが，その反面ガウスの消去法に比べて，プログラムの局所性が悪くなるという弱点もある．LU-分解法は，L を単位下三角行列とするドゥリトル法と，U を単位上三角行列とするクラウト法に分けられるが，これらの名称は必ずしも正確に使い分けられていない．ともかくも，(4.1) の分解が行われれば，連立一次方程式 $Ax=b$ の解は

$$x=A^{-1}b=U^{-1}L^{-1}b \tag{4.2}$$

で与えられる．(4.2) の右辺は，中間変数 y を用いて

$$y=L^{-1}b,\quad x=U^{-1}y \tag{4.3}$$

と書かれ，前者は下三角行列 L による前進代入，後者は上三角行列 U による後退代入により容易に計算することができる．

一方ソフトウェアの方では，これらの方法に基づき，複数右辺の同時処理，LU-分解成分の再利用，行列式の計算などの有用な機能を具えた優秀なサブルーチンが開発整備されている．

(b) 掃出し法の誤用

ところが計算の現場では，誤ったソフトウェアの選択が大規模かつ集団的に行われていて，一向に改善の兆しが見られない．第一の問題は掃出し法——正式にはガウス・ジョルダンの消去法——の誤用である．掃出し法はガウスの消去法の一変形で，ガウスの消去法では軸要素より下の方程式の係数だけを消去するのに対して，軸要素より上の方程式の係数をも消去する．そのために，消去の段階が終了したときには，係数行列は対角行列もしくは単位行列となって，ガウスの消去法のように後退代入の段階を全く必要とせず，プログラムは非常に簡単なすっきりしたものになる．これは一見計算量の少なさを物語るように見えるが，事実は逆である．N 元の連立一次方程式を解くために必要な計算量の主要項は，ガウスの消去法が $N^3/3$ であるのに対して，掃出し法は $N^3/2$ である．すなわち掃出し法の方が 50% も余分な計算を必要とする．精度の面で両者の間には有意な差は認められないので，50% の計算量の差は決定的である．

(c) 逆行列の濫用

掃出し法の誤用は，しかしながら，逆行列の濫用に比べれば物の数ではない．$Ax=b$ の解が $x=A^{-1}b$ で表わされることが明らかにその理由である．この理

由は一見自然に見えるが果してそうであろうか．一次方程式 $ax=b$ の解は $x=a^{-1}b$ と書かれるからといって，わざわざ a の逆数を計算し，これに b を乗ずる必要があるだろうか．b を a で割って $x=b/a$ とすればよいのである．$A^{-1}b$ は解を表わす単なる記号と見るべきで，$A^{-1}b$ を計算する方法はその形と関係なく，もっと別の実際的な考慮から定められなければならない．計算量を調べて見よう．逆行列を計算する方法は多数存在するが，その中で最も計算量の少ない方法はどれでも N^3 の計算量を要する．これはガウス法の3倍に当る．精度についてもガウス法の方が断然優れているので逆行列法の不合理性は明白である．

"同じ係数行列で右辺ベクトルを変化させて反復的に求解する場合には，一度だけ A^{-1} を求めて置き，$x_i=A^{-1}b_i$, $i=1,2,\cdots$ とするのは合理的である"．というのが逆行列法に対する最も強力な弁護である．しかしこの場合ですら逆行列は計算すべきでないというのが現在の一致した見解である．ガウス法で $x_1=A^{-1}b_1$ を求めるときに自然に作られる LU-分解成分を再利用して，$x_i=U^{-1}L^{-1}b_i$, $i=2,3,\cdots$ と計算すると，一つの解を求めるのに必要な計算量は A^{-1} とベクトル b_i の乗算に必要な N^2 と同じで，したがって最初の方程式を解くときの手間 $N^3/3$ と A^{-1} を求めるときの手間 N^3 及び精度の差だけガウス法の方が有利である．このように見てくると，逆行列の計算の正当性は逆行列の要素自体の必要性以外には見出されない．

(d) コレスキー法と改訂コレスキー法

係数行列が正値対称の場合の最良の方法はコレスキー分解法及び改訂コレスキー分解法である．コレスキー法は LU-分解法で L と U とが互いに転置行列の関係にある特別の場合

$$A=LL^T \qquad (4.4)$$

であり，改訂コレスキー法は L を単位下三角行列とし対角行列 D を加えた

$$A=LDL^T \qquad (4.5)$$

という場合である．どちらもガウス法などの半分の計算量 $N^3/6$ で実現でき，しかも軸選択なしで数値的安定性が保証されるという結構づくめの方法である．両者の比較では，N 回の平方根の計算だけコレスキー法が不利であるが，その他の点は簡単であってそれほど優劣の差はない．しかも，どうしてもコレスキー法でなくてはならない場合が固有値解析などにあるので，一方だけで済ますことはできない．

係数行列が対称ではあるが正値ではない場合には，(4.5) の D を 2×2 の小行列を含めた対角ブロック行列に拡張した，バンチの分解法[7] が有効であることが知られている．コレスキー法とバンチの方法はソフトウェア化が順調に行われているが，バンチの方法はまだあまり利用されていないようである．

帯行列あるいはより一般的に疎行列のための方法，さらには共役傾斜法で代表される反復法などについては紙数の関係で省略しなければならない．

4.2 固有値解析ソフトウェア

この分野は連立一次方程式と並んで学問的にもソフトウェアの面でも最も高い水準にある分野である．対称行列については，ハウスホルダ変換による三重対角化，固有値全体を求めるための QR あるいは QL 法，一部分の固有値を求めるためのスツルム二分法，固有ベクトルのための逆反復法，そして非対称行列についてはヘッセンベルグ行列への変換，固有値のためのダブル QR 法，固有ベクトルのための逆反復法などの優れた方法が確立され，ソフトウェア化されている．さらに強調すべきは，この分野では，連立一次方程式などの分野のような利用者側の誤った選択がなく，優れたソフトウェアが実力にふさわしく大いに活用されていることである．その理由は，固有値問題自体が比較的専門的である上に，優れた計算法がどれも素人の安易な理解を超える高度なものであることによる．そのために，多数の利用者は先入観に妨げられることなく専門家や識者の意見に素直に追随したものと思われる．

この分野での比較的新しい話題は，特異値分解法[8] やそれを応用した一般連立一次方程式の最小ノルム・最小二乗解の計算法[8] で，速やかなソフトウェア化とその普及が期待されている．

5. その他のソフトウェア

(a) 数値積分

この分野の現場の状況はシンプソン法一辺倒であるといっても過言ではない．ガウス法やロンバーグ法なども使われているが，学問の第一線では二重指数関数法[9]，改良クレンショー・カーチス法[10]，適応型ニュートン・コーツ法[11] などの自動積分法が盛んに研究され，ソフトウェア化も大いに進行している．積極的な普及の推進が望ましい．多重積分関係はまだ緒についたばかりの段階である．

（b） 常微分方程式

ここは，ルンゲ・クッタ・ギル法の一人舞台である．ここでも学問の最尖端と現場との落差が著しい．誤差評価能力をもつ数多くのルンゲ・クッタ法，有理補外法，可変次数多段法，スティフ方程式法[12]などの優れた業績はまだソフトウェア化が進んでいないが，そろそろ突破口が開かれようとしている．

（c） 代数方程式

カルダノ法，フェラリ法，ニュートン・ベァストー法などの古典的な方法と並んで，ジャラット法[13]，ジェンキンス・トラウブ法[14]などの新しい方法が人気がある．最近では *DKA* 法[15]がもてはやされているが，時間がかかり過ぎるという難点がある．固有値解析の *QR* 法に匹敵するような決定的方法はまだあらわれていない．

（d） フーリエ解析

2の冪乗サンプルの複素 *FFT* と実数 *FFT* は現場までよく浸透していて喜ばしい．偶奇の対称性を利用した余弦あるいは正弦 *FFT* はすでに一部でソフトウェア化[16]されている．画期的な方法として注目されるビノグラードの方法[17]の実用価値についてはまだ定説がない．

（e） その他の分野

近年進歩の著しい分野に，多変数関数の最適化，非線型連立方程式，スプライン補間などの重要な分野があるが，これらについて論ずることは著者の任ではないので省略する．

6. 数学ソフトウェアの開発と流通

数学ソフトウェアの迅速な開発と円滑な流通は，電子計算機による数値計算の合理化にとって欠くことのできない要件である．しかしながら現実はあまり満足できる状況ではない．論文の形で発表された多くの優秀な計算法が，ソフトウェアとなって具現され，末端の利用者にまで普及するためには想像以上の長い時間がかかるのである．ソフトウェアのこのような流れを阻害している諸要因を分析し，これを克服する方策を考えるのが本章の目的である．

ソフトウェアの開発を妨げている直接の原因は，これに携わる人材の層の薄さである．しかし，その奥にはより深い原因として，ソフトウェア作りに対する社会の評価の低さがある．“ソフトウェアはハードウェアの添え物にすぎないし，ソフトウェア作りは計算法の引き写し作業にすぎない”．このような考えが誤りであることは今や明白である．ソフトウェアはハードウェアと同等あるいは，より重要な計算機の要素であり，数学ソフトウェア作りは数値解析とプログラミングの有機的な総合によって始めて可能な，創造的活動である．しかしながら事態は徐々に好転しつつある．ソフトウェアの有償化が定着し始め，書籍に準ずる形での著作権制定への胎動が始まっているのである．

ソフトウェアの流通をおくらせている因子としては三つのものが考えられる．その第一は発表機関の少なさである．しかしこれは，ASM の専門誌 Transaction on Mathematical Software の発刊や，本学会の資料的論文の制定によって大いに緩和された．第二の因子は，流通機関の欠如である．これに対して，一部の計算機メーカーに，専門家と提携して数学ソフトウェアの収集流通機関の役目を果そうという動きがあるのは喜ばしいことである．また，全国共同利用大型計算機センター間において，ソフトウェアの相互融通の動きが活発化しようとしているのも見逃せない．第三の因子は利用者の保守性である．利用者はその通性としてプログラムの改良にはきわめて消極的である．しばしば彼らの使用しているプログラムは，彼ら自身の作品ではなくて，多数の人の手を経た伝来物であり，その間に加えられた無原則な変更が，プログラムを読解不能な迷路と化し，改良への意欲を喪失させている．案外これが最も根強い難敵なのであって，これを打破することは至難のわざである．プログラムの書き方と維持管理についての長年月にわたる忍耐強い啓蒙教育活動の積み重ねによって一歩一歩改善して行く以外に方法がない．

プログラミング教育においては，“自分のプログラムは自分で作れ”というのが基調であったし，現在もそうである．これは健全な一面をもっていて一概に否定できない．しかし数学ソフトウェアがかなりの充実を見せている今日，その利用によって効果的に行うことのできる定形的な計算にまでこの原則を適用するのは誤りである．教育者は単に計算法やプログラミングを教えるばかりでなく，優良なソフトウェアを紹介しその利用を奨励すべきである．できれば適切な例題を与えて，ソフトウェアを実際に使用させることがこの上なく効果的である．

同じような意味で問題なのは，プログラミングや計算法の教科書での安易な例題の選択である．好ましくないとされている方法，たとえばソーティングにおけるバブルソートとか連立一次方程式における掃出し法

などを，他の良い方法との対比もなく例題に使用することは慎むべきである．このようにして初心のうちに植えつけられた先入主が正しいソフトウェアの選択を妨害する大きな原因となっている．

以上の分析によって明らかにされた諸条件を抜本的に是正するためには，計算機社会にみなぎっている誤謬，“プログラムは正しければよい”を一掃しなければならない．そして，それに代って，“プログラムにとって正しさは必要条件にすぎない．合理性が加わって始めて十分条件となりうる”という命題を打ち建てなければならない．

7. おわりに

数学ソフトウェアについての解説を書けという編集局からの勧誘に従って筆をとったが，著者の個人的な好みにより幾分偏った内容になったことは否めない．ここにご寛容を乞うとともに，数学ソフトウェアの発展を希求する者の一人として同志諸賢のご理解とご協力を切にお願いする次第である．

参考文献

1) Rice, J. R.: Mathematical Software, Academic Press, New York (1971).
2) Hart, J. F.: Computer Approximations, John Wiley, New York (1968).
3) 二宮市三: Mathematical Software, 数理解析研究所講究録 253, p. 75 (1975).
4) Cody, W. J.: Software for the Elementary Functions, Mathematical Software, Academic Press, New York pp. 171-186 (1971).
5) 名古屋大学大型計算機センター: ライブラリ・プログラム利用の手引 (改訂版) (1978).
6) 二宮市三: ライブラリ・プログラムの新登録と改訂について, 名古屋大学大型計算機センターニュース, Vol. 12, No. 3, pp. 352-395 (1981).
7) Bunch, J. R., et al.: Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems, Math. Comp., Vol. 31, No. 137, pp. 163-179 (1977).
8) Golub, G. H. and Reinsch, C.: Singular Value Decomposition and Least Squares Solutions, Linear Algebra, Handbook for Automatic Computation, Vol. 2, Springer, Berlin, pp. 134-151 (1971).
9) Takahashi, H. and Mori, M.: Double Exponential Formulas for Numerical Integration, Bull. R. I. M. S., Kyoto Univ., Vol. 9, pp. 721-741 (1974).
10) 鳥居達生, 長谷川武光, 二宮市三: 等差数列的に標本数を増す補間的自動積分法, 情報処理, Vol. 19, No. 3, pp. 248-255 (1978).
11) 二宮市三: 適応型ニュートン・コーツ積分法の改良, 情報処理, Vol. 21, No. 5, pp. 504-512 (1980).
12) Gear, C. W.: Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, N. J. (1971).
13) Garside, G. R., Jaratt, P. and Mack, C.: A New Method for Solving Polynomial Equations, Computer Journal, Vol. 11, pp. 87-90 (1968).
14) Jenkins, M. A. and Traub, J. F.: A Three-stage Algorithm for Real Polynomials Using Quadratic Iteration, SIAM J. Numer. Anal., Vol. 17, pp. 545-566 (1970).
15) 山本哲朗, 古金卯太郎, 野倉久美: 代数方程式を解く Durand-Kerner 法と Aberth 法, 情報処理, Vol. 18, No. 6, pp. 566-571 (1977).
16) 鳥居達生: フーリエ変換サブルーチン・パッケージの作成 (その1), 名古屋大学大型計算機センターニュース, Vol. 10, No. 1, pp. 24-53 (1979).
17) Winograd, S.: On Computing the Discrete Fourier Transform, Math. Comp., Vol. 32, pp. 179-199 (1978).

(昭和56年9月16日受付)