談　話　室

標準函数プログラム作製上の問題点について

有山正孝*　吉村一馬**

初等函数の函数近似についてはすでにさまざまな近似公式が発表されており，またこれらの公式を用いて函数ルーチンを作ることは，取り上げ方によっては初心者向きの手頃な演習問題でさえあろう．いずれにせよ，よほど巧妙な方法を考案したというのでもなければ，この問題に関して改まってとやかくいうことはないようにさえ思われる．しかし，このような初等函数でも，FORTRAN, ALGOL などの標準函数* サブプログラムを作ろうとすると，いろいろ面倒なことが起ってくる．

その原因は，標準函数は，文法的に許される数，すなわちその函数の引数の型として認められた型の数であれば，その計算機の取りうる全ての値の数を引数として受付けねばならないという点にある．

実際，この種のサブプログラムを作ろうとすると，いろいろ思いがけない面倒なことが起ってくる．後で考えてみれば当然のことで，はじめから気がつかない方がどうかしているというようなことも多いが，HITAC 5020 FORTRAN の標準函数サブプログラム作製にあたっての経験を中心に，二，三気のついたことを書いてみよう．

引数の還元と区間の分割に伴なう問題

標準函数のサブプログラムに要求されるのは，広い範囲の引数に対し，なるべく一様な速さ，一定の精度で函数値を求める能力であり，しかも当然のこととしてあまり長大なものであってはならない．

近似公式の選択と区間の分割の方法については一松氏の詳細な報告**があるから，ここでは主としてプログラムを作る段階で発生する問題についてふれるにとどめよう．

HITAC 5020 の標準函数サブプログラムでは，加法公式・乗法公式もしくは周期性を利用して引数の範囲を還元し，さらに区間を分割して分割点を中心とする多項式近似を行ない，それと分割点における函数値の表を組み合わせて結果を求める方式を多くの場合に採用した．

このように引数を還元したり，変域を分割するのに伴なって，二つの問題が起きる．一つはそのような操作に際して起きる引数の桁落ちであり，もう一つは，多項式近似の後で加法公式などを使用する際に起きる函数値の桁落ちである．

まず，引数の桁落ちについて考えてみよう．函数ルーチンの引数は通常，正規化された浮動小数点表示で与えられるので，その有効数字の桁数は仮数の桁数に等しいと考えて取り扱うべきであろう．引数の最終桁が誤差を含むものと考えれば，函数の性質によっては，仮に引数が8桁与えられていても函数値が8桁の精度を持つとは限らない．極端な例をあげれば，三角函数は引数が大きくその絶対誤差がπの程度をこえる場合には，函数値はまったく決めることができない．このような場合にも，与えられた引数は誤差を全然含まないもの，いいかえると最終桁の数字の後には無限に0が続いているものと考えるなら，原理的には函数値を欲する桁数まで求められるはずである．もちろん，そのようなプログラムが実用的なものであるか否かは別問題である．

このように引数を還元するためにまるめの誤差が導入されたり，桁落ちがおきて精度が失なわれる場合があるが，それが函数値の精度にどのような影響を及ぼすかを調べておく必要がある．

* 電気通信大学
** 日立製作所中央研究所

* プログラムの中で定義しないでも使用できるようにシステムがサブプログラムを用意している函数という意味でこの言葉を使った．

** 1) たとえば
宇野利雄編：計算機のための函数近似公式
数理科学総合研究 第IV班 第5分科会
1 (1961), 2 (1962), 3 (1963)
一松　信：近似式　竹内書店　1963
2) 一松　信：函数近似について—特に近似公式の撰択について—
第5回プログラミングシンポジウム報告集，N-1 (1964)

次に加法公式等の使用に伴なう函数値の桁落ちについて考えてみよう．たとえば，HITAC 5020 の標準函数では arctan x を次の方法によって求めている（島内氏の方法）．

まず $0\leq x\leq 1$ の範囲に引数を還元したのち

$$x=u+v$$
$$u=\frac{2k+1}{16}\quad k=0,\ 1,\ 2,\ \cdots\cdots,\ 7$$
$$|v|\leq 2^{-4}$$
$$t=(x-u)/(1+x\cdot u)$$

とおき，$\tan^{-1}t$ を多項式近似で計算し，$\tan^{-1}u$ はあらかじめ計算して表にしておけば $\tan^{-1}x=\tan^{-1}u+\tan^{-1}t$ である．これをそのまま用いると $x=0$ の近傍では $\tan^{-1}x\fallingdotseq\tan^{-1}\frac{1}{16}-\tan^{-1}\left(\frac{1}{16}-x\right)$ となり，x が 0 に近ずくとこの引算で桁落ちが起きる．対策は，0 の近くでは多重精度演算をするか，あるいは $x=0$ のまわりの展開を別に行なうかで，いうまでもなく後者の方が容易である．このような場合にはじめから $u=\frac{k}{16}$，$k=0,\ 1,\ 2,\ \cdots,\ 15$ として必ず $v\geq 0$ にとるようにすれば引算による桁落ちの心配はなくなるが，それだけ分割点を増すなら，$|t|\leq 2^{-5}$ にして多項式の項数を一つでも減らす工夫をした方がよさそうである．

$\tan x$ では $x=0$ の近傍，$\log x$ では $x=1$ の近傍で同様のことが起きる．$\exp x$ は加法公式の適用の結果は乗算になるから，桁落ちの心配は起らない．

加法公式ではないけれど，双曲線函数を

$$\sinh x=\frac{e^x-e^{-x}}{2},\quad \tanh x=\frac{e^x-e^{-x}}{e^x+e^{-x}}$$

によって計算すると，$x=0$ の近傍ではひどい桁落ちが起きるから，直接 $\sinh x$，$\tanh x$ を展開によって求めるようにせねばならない．

函数サブプログラムは常に最終桁まで正しく函数値を求められるのが理想であろう．しかし，近似公式の打切り誤差を見積ることは容易であるが，以上に述べたように，引数の還元や加法公式等の使用によりまるめの誤差や桁落ちが起るために，総合的な誤差の見積りは面倒になる．最終桁まで求めるためには，要所要所に多倍長の計算をとり入れ，最後に下半分を捨ててまるめる位にすればよいかもしれないが，それでは時間がかかって話にならない．利用できる所では固定小数点演算を行なって指数部の何桁かを guard figure として用いるということも考えられるけれど，$\sqrt{x}$ を上から1桁ずつ決めて行くというような計算法の場合を除いては，面倒なわりに効果は少ない．標準函数サブプログラムとしてはそのようにしてまで 1 bit，2 bit の精度を争うより，その分だけでも step 数の少ないサブプログラムに仕立てておく方がよさそうである．したがって函数値の精度も，2進では最終の 2 bit くらい，10進では最終の1桁には誤差が含まれる場合があるものと考えて必要ならば2倍長，4倍長の高精度のサブプログラムを使用してもらう方がよい．もちろん，これは標準函数のサブプログラムは精度をおろそかにしてもよいという意味ではない．ただそれは常に仮数の桁数だけの有効数字を持つ函数表を打ち出すように作られてはいないし，またそのように作られている必要も多分ないだろうということである．

なお，区間の分割をどこまで細かくしたらよいかというのも一つの問題である．よく知られているように2進の計算機では $1/2^n$ の整数倍を分割点にするのがよい．特に 5020 のように指定された桁に対する可変長演算が金物で容易にできる計算機では，引数がどの区間に入るかを調べるのに要する時間が区間の数，引数の値によらず，ほとんど一定になるという利点がある．もっとも，区間の幅を半分にすれば分割点の函数値の表は倍になるけれど，多項式の項数はせいぜい1，2項減るだけであるから，むやみに細かく分割しても利益は少ない．明確な根拠があるわけではないが，区間の幅は 2^{-3}～2^{-4} くらい，その中点のまわりの展開式を用いて多項式の変数は 2^{-4}～2^{-5} 以下におさえるのが HITAC 5020 では最適の見当のように思われた．

Error の処理

函数サブプログラムで error というのは，与えられた引数に対する函数値が求められない場合であるが，その原因を大別すると次の三つに分類することができよう．

（a）函数値が overflow する．

（b）与えられた引数が函数の定義される範囲外にある．たとえば $\log x$，$\sqrt{x}$ で $x<0$ の場合など．

（c）与えられた引数の精度が不十分で函数値を決められない．振動的な函数で引数が非常に大きい場合に起きる．たとえば $\sin x$ で，x の仮数が 119 bit 与えられていても，指数が 121 以上であれば函数値は 1 と -1 の間にあることしかいえない．

標準函数といえども，以上のことに変りはない．た

だ，通常のライブラリー・ルーチンの場合は使用者が一応注意書を読んで選択した上で用いるはずで，引数の値もある程度は check されているものと考えられるのに対して，標準函数の場合はあらためて定義せずに使用できるところから，不適当な使い方をされる危険が非常に大きいものと覚悟せねばならない．いずれにせよ，標準函数のサブプログラムは，金物の演算機能の拡張という意味で，どのような使い方をされてもよいように十分な error 処理対策をほどこし，fool-proof にしておく必要がある．

一般に函数サブプログラムで error が発生した場合の処理としては，原因発見の手がかりとなる情報を register に保持した状態で一旦停止して manual の処置を待つこと，または error message を印刷したのち，使用者の用意した error 処理プログラムへ jump するか，サブプログラムの側で一律の処置をとったのち calling program へ帰ることなどが考えられるが，どれが一番よいかはサブプログラムの使用されるシステムによって異なる．

一昔前の計算機で，プログラムを書いた本人が機械の前に坐りこんで計算をする使用法の場合にはmanual な処置でもまにあうし，time sharing system の場合もそれが可能であろう．しかし machine time の有効な利用のためには，一時停止をせずに演算を続行すべきであり，特に batch process の場合には job をなるべく打ち切らないようにしておかねばならない．標準函数は ALGOL では function designator，FORTRAN では FUNCTION statement の形式で call されるたてまえであるから，calling program へ戻る場所は一ヵ所に限られるので，job を打ち切らないとすれば error の場合には函数に特定の値を与えておくほかなく，その値を用いて以後の計算が続行されることになる．

HITAC 5020 FORTRAN の標準函数サブプログラムでは，error の処置はまずプログラムのどこで，前述の三つのうちのどの型の error が発生したかを示す error message を印刷し，次にそれぞれの場合に応じて次のような方針で函数値を与えて calling program へ戻ることを原則とした．

（a） 絶対値最大の数に，引数の値に応じて適正な符号を付けたものとする．

（b） $f(|x|)$ を計算しておく．

（c） $f(x)=0$ とする．

もちろん，このような処置が最良のものであるとはいいきれない．異なる処理方法はいくらでも考えられる．たとえば，error message の中に，問題を引き起こした引数の値も示すようにしておけば一層親切であったかもしれない．また，函数値の与え方にしても（b）の型のものは，$f(|x|)$ を求めてもまったく意味がないという考え方もあろう．むしろ後々まで error の痕跡を残すように極端な値，たとえば0とか，絶対値の大きい数を与えておく方がよいかもしれない．しかし，どのような値を与えておけばよいかはあとでどう使われているかということによるから，一概にどうすればよいといいきれない．（c）の型の場合も同様である．函数値を含む項の影響を無視した場合の結果を求めておくという意味で $f(x)=0$ としたのであるが，函数値をそのまま乗数または除数に使ってあるとぐあいが悪い．

計算を続行することにも問題はある．error が出るかもしれぬことを承知の上で作られたプログラムの場合は当然続行しなければならないが，同じ式を parameter だけ変えて繰り返えし計算するようなプログラムの中で函数サブプログラムの error が続出するようなら，計算を続けても意味のない場合がある．もちろん，あまり多く error message が出るようなら，時間切れもしくは印刷行数の check にひっかかることになるが，その前に job 打ち切りの処置をとるようにしておくことも考えられる．

ともかく，多くの使用者の多種多様の要求をすべて満足させることは不可能である．一番よいのは calling program の中で処理方法を選択できるか，もしくは使用者が用意する error 処理プログラムへ移れるようになっていることである．そのためには function designator あるいは FUNCTION statement で error の場合の行先きを指定できるように文法を改めるのも一つの方法であろう．特に指定がなければ自動的に一定の処置をとることにしておけばよい．あるいは，標準函数のサブプログラムに error があったことを calling program の方で知ることができるように目印しを残しておいて，サブプログラムから戻った直後にこれを検査するようにしてもよい．これなら文法を改めなくとも，compiler を作る段階で打てる手である．もっともこのような‘親切’のためにプログラムを書くのが面倒になるとすれば一般にはかえって歓迎されないであろうし，標準函数の主旨に反するかもしれない．

なおサブプログラムの中で発生する error につい

て一言しておく．厄介なのは計算の途中で overflow がおこるが函数値は overflow しないという場合である．たとえば複素数の平方根を求めるのに，$R_e(z)$, $I_m(z)$ がそれぞれ overflow 寸前の場合 $z^{\frac{1}{2}}$ は求められるはずであるが，途中で下手に $|z|$ を計算すると overflow してしまう．このような場合も救える限りは函数値を求めるようにしておくべきであろう．しかしそのためにはいちいち引数の大きさを検査せねばならず，普通の大きさの引数の場合でもそれだけ余分に時間がかかってしまうということになる．overflow したら対策を講ずるようにすれば簡単になるが，そのときモニタが介入すると時間がかかってぐあいが悪い．標準函数サブプログラムには絶対ムシがいないはずだから，その中ではモニタの割込みを禁止しておくのも一方法であろう．

プログラムの検査方式について

プログラムを作成したあとで，それが正しく計算してくれるものかどうか検査しなければならないが，これは重大な仕事であるにかかわらず，あまり問題にされないのは‘これだけやれば絶対大丈夫である’と保証できる実用的方式が見つかっていないためであろう．検査方式の問題点をあげてみると，検査用の入力データの選び方とその個数の評価，いかなる方法で誤差を求めるか，があげられる．

検査用の入力データの選び方として二つの方法が考えられる．一つはプログラムの流れの全ての組み合わせを通るデータを選ぶ方法で，プログラムの作成者が debug の時に採用するものである．この方法のよい所は全ての流れを通ること，調べなければならないデータの個数としては最小であることである．その反面，作成した人でないと，フローチャートをよく見ないと判らないこと，またプログラムは必ずしもフローチャートのとおりに作成されているとは限らない，などの欠点を持っている．第2の方法はプログラムの作成方法には無関係に変域の全てについて調べるもので，乱数によるもの，適当なきざみの網の目を調べるなどの方法がある．この方法の長所は画一的である点にあるが，いくら網の目を細かくしてもこれでプログラムの全ての流れを通っているとの保証のない欠点がある．

HITHC 5020 FORTRAN の標準函数プログラムでは，この二つの方法の併用によって検査を行ない，一様乱数によるものは平均3万個行なってある．

次にいかなる方法で誤差を知るかであるが，大きく分けて3種の方法があげられる．数表と照し合わせるという原始的方法，同じプログラムで精度のよいプログラムがあるとそれと照し合わせる方法，もう一つは函数の間の関係（たとえば加法定理）を使って自己診断させる方法である．

同じ函数を一重精度，二重精度，四重精度にわたって作るときは2番目の方法が便利である．3番目の方法は区間を分けて函数近似する所でふれたような加算による桁落ちの現象への注意と，$\sin^{-1}x$ のような多価函数のとき定義域についての注意をしないと見かけの誤差がでて，驚かされる．