数理解析研究所講究録
第253巻 1975年 69-82

Mathematical Software

名古屋大学　工学部　二宮市三

筆者はこの三年来、名古屋大学大型計算機センターにおいて、Mathematical Softwareの改良開発に従事して来たのであるが、ここにその成果の一部を紹介し、あわせて日頃抱懐している意見を述べて、大方の御批判を受けたいと考える。

I. 基本外部関数と特殊関数

FACOM 230－60 のFORTRANシステムの基本外部関数ルーチンの全面的な書き換えを行った。改良案は精度、速度、語数などあらゆる点で旧案にまさり、特に30％－50％の速度向上が著しい。このような優秀性のために、改良案はメーカー富士通社の正式ルーチンとして登録され、同社より各ユーザーに提供されて現在順調に稼動している。各々の関数に対する新旧両案の速度比較テストの結果を表1に掲げる。[2]

改良に着手するにあたって採用した指導原則は、重要度の順に列挙すれば次のようである。

（1）単精度（26ビット、10進8けた弱）、倍精度（61ビット10進18けた強）ともに、けた数一杯の精度の実現を目指す。

（2）速度向上に力を注ぐ。

（3）使用語数を少くする。

表1　基本外部関数の演算速度

関数名（単精度）	平均演算速度 (μs) 新	旧	関数名（倍精度）	平均演算速度 (μs) 新	旧	引数範囲
EXP	71	171	DEXP	146	267	−20 ～ 20
EXP2	63	169	DEXP2	142	260	−20 ～ 20
EXP10	72	170	DEXP10	148	265	−20 ～ 20
ALŌG	78	106	DLŌG	155	201	0 ～ 20
ALŌG2	71	101	DLŌG2	150	195	0 ～ 20
ALŌG10	78	105	DLŌG10	156	200	0 ～ 20
SIN	89	169	DSIN	157	252	−20 ～ 20
CŌS	90	170	DCŌS	153	254	−20 ～ 20
SINHP	83	—	DSINHP	151	—	−20 ～ 20
CŌSHP	84	—	DCŌSHP	147	—	−20 ～ 20
TAN	103	155	DTAN	172	244	−20 ～ 20
CŌT	106	168	DCŌT	168	262	−20 ～ 20
TANHP	97	—	DTANHP	166	—	−20 ～ 20
CŌTHP	100	—	DCŌTHP	162	—	−20 ～ 20
SINH	91	225	DSINH	168	348	−20 ～ 20
CŌSH	95	188	DCŌSH	163	289	−20 ～ 20
TANH	105	190	DTANH	150	313	−2 ～ 2
ARSIN	121	252	DARSIN	210	421	−1 ～ 1
ARCŌS	122	265	DARCŌS	214	434	−1 ～ 1
ATAN	95	134	DATAN	192	276	−20 ～ 20
ATAN2	110	152	DATAN2	208	299	第1変数 −20 ～ 20 第2変数 1
SQRT	67	78	DSQRT	93	100	0 ～ 20
CBRT	112	160	DCBRT	164	214	−20 ～ 20
ERF	109	198	DERF	227	296	0 ～ 1
ERFC	112	187	DERFC	230	283	0 ～ 1
GAMMA	184	239	DGAMMA	357	453	0 ～ 20
ALGAMA	190	245	DLGAMA	370	470	0 ～ 20
CABS	102	161	DCABS	157	195	−100 ～ 100+100*i*
CEXP	249	553	DCEXP	462	826	−100 ～ 100+100*i*
CLŌG	292	459	DCLŌG	573	738	−100 ～ 100+100*i*
CSIN	393	619	DCSIN	669	897	−100 ～ 100+100*i*
CCŌS	395	607	DCCŌS	693	888	−100 ～ 100+100*i*
CSQRT	208	324	DCSQRT	323	393	−100 ～ 100+100*i*

以上が一般原則であるが、これを生かすために個々の関数について、そのアルゴリズムの選定や実際のコーデイングにおいて用いた具体的な方策は次の通りである。

(4) 単精度のルーチンでは、できるだけ固定小数点演算を用いる。

これによって浮動小数点演算よりも9ビットだけ余分のけた数が使えるので、丸めの影響はほとんどなくなり、格段によい精度がえられる。又、浮動小数点方式ではけた落ちのために無効となるような近似式を用いることもできる。一方浮動固定両表示間の変換やスケーリングの必要性は速度を減じ、語数を増す方向に働くマイナス材料であるが、精度第一主義のためやむをえない。

(5) 速度重視の見地から、プログラミングの常道とされているインデックスループの使用を避ける。

著しい速度向上の主要な原因はこの方策の採用にあったと思われる。この方策は一般に語数増加の方向に働くが、インデックスの保存、復元、初期設定、修正、判定がことごとく不必要となる点は語数減少の方向に働くので、反復数が小さい場合には差引き語数が減少することがある。

(6) 単精度の指数関数、三角関数の区間縮小の際の情報の喪失による精度の低下を防ぐために、倍精度演算を混用する。

例えば、指数関数は $e^x=2^{x\log_2 e}$ として計算されるが、たとえXに誤差がなくても、$\log_2 e$に誤差があれば($\log_2 e$を単精度の数として用いたのでは不可避の丸めの誤差がある)$x\log_2 e$にも同程度の誤差がある。この誤差は整数部分が抜き去られると、その分だけ上方のけたにくり上がり、e^xの値に大きな誤差をもたらす。これを防ぐには、$\log_2 e$を倍精度で表わ

し、$x\log_2 e$の計算を倍精度で行えばよい。しかし、これはあくまでもxに誤差がないという前提の上での話で、この前提が成り立たなければ（実際にはほとんどの場合成り立たない）何等解決策とはならないばかりでなく、どんな解決策もありえない。同様な事情はA** Bの形のべき乗計算で起る。$A^{**}B = 2^{B \cdot \log_2 A}$ と計算されるわけだが、2を底とする対数ルーチンで計算された$\log_2 A$ の値（小数部分が先に計算され、これに整数部分が加えられる）を丸めることなく、そのままの姿でこれに倍精度演算でBを乗ずれば、Bに誤差がなければ、大きな精度の損失を防ぐことができる。

（7）原点で0となる関数の原点近傍での精度を確保する。

このような関数は、特殊なSQRT、CBRTをのぞきすべて奇関数で SIN、TAN、ARSIN、ATAN、SINH、TANHがそれである。この種の関数f（x）に対して、上の方針を貫き同時に（4）の方策をも生かすには、f（x）／xの近似式を固定小数点計算で計算し、これに浮動小数点数としてのxを乗ずればよい。

以上で基本外部関数についての説明を終る。筆者は同様な考え方に基いて、応用上重要な諸特殊関数ルーチンの開発を行った。[3] その結果、指数積分、正弦余弦積分、フレネル積分、第一種及び第二種完全楕円積分、ヤコビ楕円関数及び各種の（球）（変形）ベッセル関数などが、単複両精度ともに、ほぼ完備するに至った。この中には、特に倍精度の関数において、不完全な近似式を使っているものがあるので、逐次改善して行くつもりである。これらについての詳細は一切省略し、ここではより基本的な二種類の関数について述べる。

その一つは、SINHP、COSHP、TANHP、COTHP、及びその倍精度版（頭にDをつける）の導入である。これらはxに対して$\sin\frac{\pi}{2}x$、$\cos\frac{\pi}{2}x$などを直接計算する関数である。名前の末尾のHPはHALF PIのつもりだが、双曲線関数の名前とまぎらわしいのは多少気がひける。さて、このようなものをわざわざ特殊関数として設ける理由は二つある。その第一は$\sin\frac{\pi}{2}x$とかcos2πxとかいうような形の計算の必要性が大であることであり、その第二はSINやTANのルーチンでは、区間縮小の必要上SINHPやTANHPが計算されていて、たとえばSIN（X）はむしろSINHP（X／HP）として（HPにはπ／2が入っているものとする）計算されていることである。$\sin\frac{\pi}{2}x$を従来通りSIN（HP＊X）として計算すると、実はSINHP（HP＊X／HP）が計算され、π／2を乗じてすぐその後でπ／2で割るという無益の二重手間をふむことになる。SINHPを用いればこのような無駄がなく、しかもπの値をあらわに書くわずらわしさからも解放される。

その二はlog（1＋x）、$\sinh^{-1}x=\log(x+\sqrt{1+x^2})$、$\tanh^{-1}x=(\frac{1}{2})\cdot\log((1+x)/(1-x))$などの関数である。これらは科学技術計算にしばしば登場する重要な関数で、原点で0となる。ところがこれらを定義式通り対数関数を経由して計算すると、原点附近で著しい精度の低下を招く。したがって、これらの関数に対して、原点附近での特別の措置を含むルーチンを作ることは有意義である。名前は一応ALOG1、ASINH、ATANHなどとつけたが、よりよい案があれば示されたい。

さて、次にPremature overflow（Underflow）について述べる。M（m）を計算機の内部で許される最大（最小）の絶対値としよう。$\sinh x=(e^x-e^{-x})/2$、$\cosh x=(e^x+e^{-x})/2$をこれらの式の通りに計算すると、

xの値が$\log M < x \leq \log 2M$の範囲にあるときには$\sinh x = \cosh x \leq M$であるのにもかかわらず、$e^x > M$であるためにオーバーフローを生じ、計算が無効になってしまう。すなわち、一般的にいうと、計算の最終結果は合法な数であるのに、これを算出する途中の段階で非合法な数が発生するための困難が問題なのである。上例のオーバーフローを防ぐには、e^{-x}がe^xに比べて無視できるようなxに対しては$\sinh x \fallingdotseq \cosh x \fallingdotseq e^x/2 = 2^{x\log_2 e - 1}$と計算すればよい。

類似の現象が複素数の計算では豊富に起る。もっとも典型的なものはCABSの場合で、$|Z|$を単純に$\sqrt{|Z|^2}$と計算すると$|Z| > \sqrt{M}$（$|Z| < \sqrt{m}$）のときオーバー（アンダー）フローとなってしまう。この現象は以前からよく知られており、回避の処置も打たれている。しかしもっと重要と思われる除算の場合は案外見過されているようである。今$p = a + ib$を$q = c + id$で除して$r = e + if$となるものとする。この場合、誰にも明白で常識的なアルゴリズムは、次の通りである。

$\bar{0}1.$ $w \leftarrow c^2 + d^2$

$\bar{0}2.$ $e \leftarrow (ac + bd)/w$

$\bar{0}3.$ $f \leftarrow (bc - ad)/w$

ところが、$|q| > \sqrt{M}$（$|q| < \sqrt{m}$）ならば、wの計算でオーバー（アンダー）フロー、$|pq| > \sqrt{2}M$（$|pq| < m$）ならば、eやfの計算でオーバー（アンダー）フローになる。合法な複素数に一様分布を仮定すると、上の困難は50%以上の確率で発生することになる。これを防ぐには、アルゴリズムを次のように改める。

N1. $|c| \geq d$ ならばN2aへ、さもなければN2bへ.

N2a.	$t \leftarrow d/c$	N2b.	$t \leftarrow c/d$
N3a.	$w \leftarrow dt + c$	N3b.	$w \leftarrow ct + d$
N4a.	$e \leftarrow (bt + a)/w$	N4b.	$e \leftarrow (at + b)/w$
N5a.	$f \leftarrow (b - at)/w$	N5b.	$f \leftarrow (bt - a)/w$

このようにすれば、$|t| \leq 1$ であるので、wは $|q|$ の程度の数、e、f の分子は $|p|$ の程度の数となり困難は大いに緩和される。もっとも、困難が完全に除去されるわけではないが、改良の意図は十分達成されたものと考えられる。より重要なことは、改良案が速度と精度の上で僅かに改良になっている点である。それは、新（旧）アルゴリズムに要する演算が加減算3（3）、乗算3（6）、除算3（2）であることを見ればわかる。筆者は最近このことに気付き、複素除算ルーチンを書き改めて実験したところ、10%－20%の速度向上が見られた。改良案は語数の点では改悪であるが、改良点はこれを埋めて余りあるものと思う。[8]

II. 四倍精度計算システム

多重精度計算システムを作成するのに二つの立場がある。その一 は一般性や互換性を尊ぶ立場であり、その二は実用性を尊ぶ立場である。筆者は後者の立場に立ち、四倍精度専門の計算システムを作った。効率や計算速度を重視して、すべてのサブルーチンをアセンブリ言語FASPで書いた。[3,8]

このシステムでの四倍精度の数（以下Q型という）は連続する5語で表わされ、始めの4語に仮数部（35×4＝140ビット）、最後の1語に指数部

（35ビット）を入れる。したがって精度は10進42けた強となり、単精度の5倍強に達する。指数部にとられたビット数が多いので、オーバーフローやアンダーフローの心配はほとんどない。このシステムを使って四倍精度の計算をするときには、特別なコール文を必要とせず、230－60FŌRTRAN独特のTYPE文を加えるだけで、普通の型の計算と同様にFŌRTRANで気楽にプログラムできる。たとえば、

```
TYPE Q*5, X, Y, A, B, QEXP, QABS
DIMENSIŌN A(10), B(10,10)
```

と書けば、変数X、Y、一次元配列Aの各要素、二次元配列Bの各要素、基本外部関数QEXP、QABSの値はQ型として宣言され、以後これらをFŌRTRANの代入文の中に自由に使うことができる。整数型、実数型、倍精度実数型との、あらゆる混合をゆるす、加減乗除とべき乗が可能である。論理IF文は働かないが、算術IF文は利用できる。入力というか、データから四倍精度の数を作ることはや、複雑だが、実用上ほとんどその必要はない。出力は特別なサブルーチンコールによらなければならないが、若干の書式制御を伴う、5けた区切りのE型とF型のプリント出力が可能である。また基本外部関数としては、表1から複素数関係のものを除いてほとんど全部のものが用意されており、QABS、QFLŌAT、IQFIXなどもそなえられている。表2に各種演算ならびに主な関数の速度を掲げる。これらの値は多重精度演算の速度としては出色のものと思う。筆者はこのシステムを倍精度関数の近似式の係数の計算や精度のチエックに使い、非常に利便を感じている。又、一般ユーザーにも開放され、一部ではかなり使用されて好評のようである。

表2. 四倍精度の速さ

演算の種類	平均速度 (ms)
代　入	0.04
符号変化	0.05
加　法	0.2～0.3
減　法	0.2～0.3
乗　法	0.4～0.5
除　法	0.4～0.5
QSQRT	2.0
QEXP	10.0
QLŌG	13.0
QSIN	12.0
QCŌS	12.0
QATAN	13.0
QTANH	11.0

III. 線型計算ライブラリプログラム

ライブラリプログラムの中で最も基本的で汎用性の高い線型計算のライブラリプログラムの開発を行った。すべてのプログラムは（前節の特殊関数ルーチンも含めて）アセンブリ言語FASPでコーデイングした。高級プログラミング言語の発達した今日、ことさらアセンブリ言語を用いる理由は、言うまでもなくでき上ったプログラムの品質の高さ、密度の濃さである。機械の特質を生かした、融通性のある、無駄のないプログラムは短かくて速い。これに対して難点は、プログラムの互換性の欠如、プログラム作成、解読、変更の困難さにある。一般的な風潮は、この難

点を過大視して、数値計算のライブラリルーチンをアセンブリ言語で書くことは途方もないことと考える傾向がある。しかし、筆者はこの風潮こそ打破すべきであって、基本外部関数と同様に、少なくとも特殊関数や線形計算などの基本的なプログラムだけはアセンブリー言語で書くべきものと確信する。

ともかく、筆者は所信を実行に移し、行列正規化、行列式、逆行列、連立一次方程式、対称行列固有値問題、FFTなど多数のプログラムを作った。[2,4,6,7,8] ここではその中の二三の重要なものについて、や、詳細に説明する。

(1) 連立一次方程式

連立一次方程式解法ルーチンの中の主要なものとして、一般の方程式を、行交換を伴うLU一分解法で解くためのLEQLUS、LEQLUD、対称正値な係数行列をもつ方程式を、コレスキー法で解く、フルマトリックス用のCHŌLFS、CHŌLFD、及びバンドマトリックス用のCHŌLBS、CHŌLBDがある。名前の末尾にS(D)があるのが単(倍)精度用である。以下単精度ルーチンに話を限定する。これらのルーチンに共通する特徴は次の通りである。

(a) オプションとして、行列式の値が計算出来る。

(b) 係数行列を共有する、複数個の方程式を同時に解くことができる。複数の中には、0も入っており、この場合は係数行列の分解だけを行う。

(c) オプションとして、係数行列の分解成分の再利用が可能である。この機能のため、これらのルーチンは逆行列ルーチンの働きをする。

(d) 積和計算などに倍精度計算が混用されていて精度がよい。しかも

この機能を入れることによりスピードは全くそこなわれない。

(e) コーデングに工夫がこらしてあって、スピードが非常に速い。同じアルゴリズムをFORTRANで書き、最適化の入ったコンパイラーでコンパイルしたプログラムに比べて、1.3～3.0の速度比が見られる。表3に6個のサブルーチンの速度テストの結果を示す。LEQLUS、LEQLUD、CHOLFS、CHOLFDには100元のフランクの行列（i, j要素が101－max（i, j））を係数とする方程式を用い、CHOLBS、CHOLBDには1000元で半帯幅21のある行列を係数とする方程式を用いた。

表 3

ルーチン名	所要時間	ルーチン名	所要時間	ルーチン名	所要時間
LEQLUS (FORTRAN)	9.8秒	CHOLFS (FORTRAN)	3.8秒	CHOLBS (FORTRAN)	15.0秒
LEQLUS (FASP)	5.4秒	CHOLFS (FASP)	3.0秒	CHOLBS (FASP)	5.2秒
LEQLUD (FASP)	6.8秒	CHOLFD (FASP)	3.7秒	CHOLBD (FASP)	5.9秒

(f) できる限り行列要素を列方向にたどるようなアルゴリズムが採用されているので、仮想記憶向きである。

以上の特徴の中で（c）についてさらに説明しよう。「LU－分解法」やこれに数学的に同等な「ガウス消去法」では、係数行列Aを単位下三角行列Lと上三角行列UによってA＝LUと分解し、これを用いて方程式Ax＝bを、Lについては前進代入、Uについては後退代入により、$x=U^{-1}(L^{-1}b)$として解く。もしもL、U両成分が保存してあれば、同じAを係数にもつ別の方程式$Ax'=b'$を解く必要を生じたとき、LU－分解を省略して、ただちに

$x' = U^{-1}(L^{-1}b')$ と計算できる。これがLU一分解成分の再利用ということである。実際は、分解の際のピボット選択のために必要な行交換の情報を保存しておいて、これを考慮しながら方程式を解かねばならないので、プログラムは幾分複雑となる。同じことは逆行列A^{-1}を作っておいて$x' = A^{-1}b'$という風にもできるわけで、この場合には行交換の効果はすでにA^{-1}の中に織りこまれているので、これを考慮する必要がなく、$A^{-1}b'$は行列とベクトルの積であるのでプログラミングが簡単である。しかし、計算量をしらべて見ると、LU一分解に$n^3/3$、$x = U^{-1}(L^{-1}b)$にn^2に対して、A^{-1}にはn^3、$x = A^{-1}b$にn^2である。すなわち、LU一分解の方が計算量の少なさで、したがって精度の良さですぐれている。又Aが正値対称帯行列のときには、そのコレスキー分解成分もAと同じ大きさの帯行列であるのに、A^{-1}をとるとフルマトリックスに拡がってしまい、折角のスパース性がこわれてしまう。このように考えると、逆行列を計算することを良しとする根拠は、プログラミングの単純さによる心理的な気安さだけのように思われる。（c）の機能をもつサブルーチンを用いることにすれば、プログラミングの簡単さは問題にならないので、このような場合に逆行列を用いることは合理的でない。一般的にいって、行列の積の因子としての逆行列を計算することは必要でもないし、得策でもない。ところが、最近いくつかの共同利用大型計算機センターで、相前後して実施された（北大、名大[5]、九大、東大）、ライブラリルーチン使用頻度調査の結果によると、どのセンターでも申し合わせたように逆行列ルーチンが高位を占めている。逆行列がそれ自体のために計算されねばならないという場合は非常に少ないので、このように大量に行われる逆行列の計算の

大部分が上述の議論の意味で不経済な計算であることは疑いない。この現状を打開するには、LU一分解（コレスキー分解）成分の再利用の機能をもつ連立一次方程式ルーチンを早急に整備して、その利用をすすめるとともに、初心者の数値解析、プログラミング教育にあたって、この点を大いに強調し正しい常識を植えつけることが必要である。

関連事項としてもう一つ言いたいことは、「掃き出し法」についてである。この名前は正式名「ガウスジョルダンの消去法」の別名であるが、世間では消去法一般あるいは、場合によっては、「ガウスの消去法」を意味するものとして誤用され、混乱の種となっている。又方法自体も、逆行列の算法としては良い方法であるが、連立一次方程式の解法としては、ガウスの消去法やLU一分解法に比べて50%も余分な計算量を必要とするので、使わない方がよいと思うが、どんなものであろうか。

（2）FFT

複素入力[2]及び実入力[4]のFFTルーチンを作成した。複素入力の方はビット逆転をしないCooley- Tukeyの算法を用い、実入力の方は、最初にビット逆転をやる方法を独自に考案して採用した。三角関数の値は、初めにまとめて計算しないで、必要に応じてその都度計算する。このため余分のメモリを必要としない。又、加法定理を使わず、標準関数をコールすることもなく、ルーチンの内部で特別に作った短い近似多項式で直接計算する。このため精度がよく、スピードも速い。1024分点の場合の所要時間は、複素入力で0.52秒、実入力で0.18秒である。このFFTこそは、スピードが売りものであり、そのアルゴリズムの本性から見て、ぜひともアセンブリ言語でプログラムすべきものと思う。

IV. 結言

現在の計算社会におけるMathematical Softwareの質と流通の状態は決して満足すべきものではない。時代おくれのソフトウエアがいつまでも幅をきかしているかと思うと、最新の研究の成果を盛りこんだソフトウエアは一向に現われて来ない。大衆は一度使ったソフトウエアに固執して、新しい良いソフトウエアが現われてもこれを敬遠してよりつかない。

このような状態を打開するにはどうしたらよいか。数値解析の知見とプログラミングの技術を兼備した人材を養成すること、Mathematical Softwareを作るという生産性の高い知的創造活動に対する評価の水準を高めることなどが必要であることはいうまでもない。しかし、その前に、数値計算と電子計算機の利用に対して、合理性をきびしく追及する「道徳」の養成が肝要である。この道徳の確立なくして、計算文化の開花はありえない。

参考文献

[1] Mathematical Software, J.R.Rice edit. Academic Press, 1971.
[2] 名古屋大学大型計算機センターニュース，第10号，1972.
[3] 名古屋大学大型計算機センターニュース，第14号，1973.
[4] 名古屋大学大型計算機センターニュース，第17号，1973.
[5] 名古屋大学大型計算機センターニュース，第18号，1973.
[6] 名古屋大学大型計算機センターニュース，第20号，1974.
[7] 名古屋大学大型計算機センターニュース，第21号，1974.
[8] 名古屋大楽大型計算機センターニュース，第23号，1974.