社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－172 (22)
2007－HPC－109 (22)
2007／3／1

多倍長ﾗｲﾌﾞﾗﾘによる精度評価と改善に関する考察

濱口信行

nobuyuki.hamaguchi.sa@hitachi.com

(株)日立製作所 ｿﾌﾄｳｴｱ事業部

概要

精度評価と改善のため、4倍精度以上の演算を行う多倍長ﾗｲﾌﾞﾗﾘを試作し、幾つかの数値計算問題で検討した結果、加減算で発生する誤差要因を定量化した誤差情報、浮動小数点数を表現する配列整数ﾃｰﾌﾞﾙの使用が非常に有効である事が判明した。

Consideration about precision evaluation and improvement of accuracy on numeric calculation using Multi-precision library

Nobuyuki Hamaguchi

Software Division, Hitachi, Ltd.

Abstract

We have developed the Multi-precision library that calculates floating point data of quadruple or higher precision in order to improve calculation accuracy. We examined the library on several numeric problems and obtained the result that the use of the array integer table, which represents value of floating point number, and the use of error information on addition and subtraction are very effective on precision evaluation and improvement of accuracy.

1. はじめに

数値計算においては、得られた解が想定される解と明らかに異なっていたり、どの程度の精度があるか不明な場合がよく発生する。しかし、現状では、原因を究明する方法が不明であったり、より高精度の演算の使用による演算時間の増大などの問題がある。本論文では、誤差の解析、精度の改善のための、効率の良い方法を検討した。

なお、浮動小数点数の表現形式はIEEE754形式をとり、P≧3 のP倍精度として符号部 1ﾋﾞｯﾄ、指数部 15ﾋﾞｯﾄ、仮数部 32*p-16 ﾋﾞｯﾄを仮定している。

2. 誤差要因

浮動小数点演算で発生する誤差は、

(1) 近接する2つの数A、B の減算による桁落ち
(2) 絶対値の差の大きい2つの数A、Bの加算による情報落ち
(3) 乗算、除算の丸め誤差

があり、(3)は内積演算の様に、演算結果を更に加減算する場合にも誤差が発生する。

これらの事から、加減算による誤差情報、多くの要素を加算する効率の良い総和計算が必要となる。

3. 加減算

浮動小数点数A, B(A≠0、B≠0、A≠B) の加減算は、A>B>0 の A+B、A-B の2ｹｰｽに帰着される。

この事から、数値計算で発生する誤差要因を定量化する。(図1)

無効ﾋﾞｯﾄ数: A+B において、丸めを含め使用されなかったBのﾋﾞｯﾄ数。(情報落ち)

ﾛｽﾄﾋﾞｯﾄ数 : A-B においてAとC=A-Bの指数部の差。(桁落ち)

A=0, B=0, A=B≠0 の場合は、以下の様に定めている。

無効ﾋﾞｯﾄ数: A=0, B=0、 A=B≠0 の場合 0
A>0, B=0 の場合 仮数部のﾋﾞｯﾄ数+1

ﾛｽﾄﾋﾞｯﾄ数: A=0, B=0、 A>0, B=0 の場合 0
A=B≠0 の場合 仮数部のﾋﾞｯﾄ数+2

試作した多倍長ﾗｲﾌﾞﾗﾘでは、これらの値を区別するため、無効ﾋﾞｯﾄ数には、負符号を付けている。

4. 総和演算

```
総和計算
T=0
DO I=1,N
T=T+A(I)
END DO
```

では、A(I)に0があった場合は、加算操作を行わない様にすれば、多倍精度は整数配列IA(1200) ととれば演算中にｵｰﾊﾞｰﾌﾛｰ、ｱﾝﾀﾞｰﾌﾛｰは発生しない。この為、浮動小数点数を整数で表現し、演算も整数演算を使用する様にすれば、総和計算は誤差なく計算出来る事になる。

Pi=Ai*2**(n) Pi:整数 T=ΣAi=(ΣPi)/2**(n)
(n=18000)

5. 数値計算例

数値計算で使用されている幾つかの問題で、誤差情報の適用を検討した。
ここではより一般的で現実的な状況を想定し、個々の数値計算上の技法は使用されていない事を前提としている。

5.1 四則演算

たとえば、4倍精度で、f の値を計算すると、[1]
a=77617.0 b=33096.0

f=333.75*(b**6)+(a**2)*{11*(a**2)*(b**2)-
(b**6)-121*(b**4)-2} +5.5*(b**8)+a/(2*b)

1.172603940053178631858834904520 18 の結果が得られる。

ところが、この計算の理論値は、-54767/66192
=-0.827396059946821368141165095497981370
で異なる結果となっている。4倍精度でのﾋﾞｯﾄ消失情報は

```
bit1...          0
bit2...        -23
{11*(a**2)*(b**2)-(b**6)}-121*(b**4)
bit3...        -88
{11*(a**2)*(b**2)-(b**6)-121*(b**4)}-2
bit4...          0
bit5...        114
[333.75*(b**6)+(a**2)*{11*(a**2)*(b**2)-(b*
*6)- 121*(b**4)-2}]+5.5*(b**8)
bit6...          0
```

となっていて、bit5 の演算で、[ ] 内の演算結果と5.5*(b**8)の加算で0 となる事を示している。また、bit6 の a/(2*b) を加える演算での値は0であるため、bit5 まで(bit5 も含む)の演算で発生する丸め誤差に原因がある事がわかる。

また、bit2, bit3 での値が負で、-23, -88 より8倍精度だと情報落ちの影響は防げる事を示している。

試作した多倍長ﾗｲﾌﾞﾗﾘの8倍精度ﾙｰﾁﾝを使用すると、
bit5 121 , answer=
-0.827396059946821368141165095479816
となり、正しい結果となっている。またﾛｽﾄﾋﾞｯﾄ数が121 となっているので、4倍精度では正しく計算出来ない事も示している。

5.2 Hilbert 行列の求解

需要予測、回帰分析などで最小二乗近似法を使用する場合、正規方程式は、Hilbert行列Aとなり、Ax=bの連立一次方程式を解く事が必要になる。ただ、この連立一次方程式は小さな次元数(n=10 程度) でも非常に精度が悪くなる事が知られている。[2]

実際、条件数を幾つか計算すると、

n=5	0.943*1.0q+6
n=10	0.353*1.0q+14
n=20	0.628*1.0q+29
n=30	0.118*1.0q+45
n=40	0.225*1.0q+60
n=80	0.324*1.0q+121

となり、得られた結果の精度が悪くなる事を裏付けている。この方程式の求解に誤差情報を適用する。

例として4倍精度で以下の方程式を解いてみると
Hilbert 行列 A :30*30 matrix:Aij=1.0/(i+j-1)
ﾍﾞｸﾄﾙ b 要素数 30 b(i)= ΣAij (j=1,30)
Ax=b の連立一次方程式 を解くと、理論解は
x(i)=1.0 (i=1,30)

結果は、 GOSAMAX= 29.755
ibit = -6 <= b を作成する場合の無効ﾋﾞｯﾄ数の最大値
ibit1= -11 <= 前進消去、後退代入での無効ﾋﾞｯﾄ数の最大値の和

となり、計算結果は、全く信用出来ないものとなっている。
Aij を求める際、除算の丸め誤差が発生するが、Aijの要素から、解がすべて1になる様にbを生成しているので、Aij を求める際の除算の丸め誤差は結果の誤差には影響していない。この誤差情報から、Aij を仮数部112-6-11-1=94 ﾋﾞｯﾄで作成すると、ﾍﾞｸﾄﾙbの作成時の情報落ちの影響が軽減される事が考えられる。Aij の仮数部を94 ﾋﾞｯﾄで実行すると、
ELAPSE TIME= 2.9280000000000000E-003 SEC
GOSAMAX= 9.1958 E-0005となっている。
Aijを4倍精度で求め、b の作成、連立一次方程式の求解部分を高精度で行うと、ﾌﾟﾛｸﾞﾗﾑ修正に要する負担は少なくて済む。実際この問題で、試作した多倍長ﾗｲﾌﾞﾗﾘの8倍精度ﾙｰﾁﾝを使用すると
ELAPSE TIME= 9.7600000000000000E-003 SEC
GOSAMAX= 2.4428E-0012
となり、実用に充分耐えうる精度となり、演算時間の増大も、4倍精度の3.5倍以内と負担の少ないものとなっている。

5.3 特殊関数値の計算

Li3(z)=Σ(1/n**3)*z**n の様に、特殊関数は、無限級数の和で表される事が多い。[3]
この場合、必要精度でどこまでの項を加えて良いかが問題となる。これを、無効ﾋﾞｯﾄを使用して判定する方法を以下に示した。
4倍精度の範囲で、Li3(1/2) の計算例では、

i, ibit=	90	-108
i, ibit=	91	-109
i, ibit=	92	-110
i, ibit=	93	-111
i, ibit=	94	-112
i, ibit=	95	-113

res=0.53721319360804358712084833232241 4
となり、n=95 まで加えると、それ以降は加えても4倍精度では値は変化しない事
がわかる。またn=95 までにも、情報落ちが発生している事がわかるので、4倍精度の結果を得るには、8倍精度ﾙｰﾁﾝを使用する必要がある事がわかる。試作した8倍精度ﾙｰﾁﾝでの結果は、以下の様になっている。

結果

i, ibit=	90	-108
i, ibit=	91	-109
i, ibit=	92	-110
i, ibit=	93	-111
i, ibit=	94	-112
i, ibit=	95	-113

res=0.53721319360804020094062322559496 6
上記2ｹｰｽの結果より、4倍精度のまま計算すると10進14桁までしか精度が保証出来ない事がわかる。

5.4 反復計算

数値解析では、反復計算が多く見られ、この処理では、値が収束したかどうか判定するのに、相対誤差、絶対誤差[4]で評価する為、浮動小数点除算がたびたび現れる。
ここでは、絶対値最小の固有値を算出する場合を例にとる。[5]

∂2u/∂x2=-λu (0<u<1) u(0)=0, u(1)=0 を4倍精度で計算する。この場合、
理論解 π**2
N等分離散化した時の解 2*N**2*(1-COS(PI/N))
(計算機理論解) が知られている。

通常収束判定処理は以下の様になる。

```
ERROR= ABS((ENEW-EVAL)/ENEW)    <= 除算を使用
       EVAL=ENEW
       NITER=NITER+1
       IF(NITER.EQ.M) GO TO 5
       IF(ERROR.GT.EPS) GO TO 2
   5 CONTINUE
```

これを、ﾛｽﾄﾋﾞｯﾄ数で収束判定処理をするには、

```
CALL Q4ADDSUB(ENEW, EVAL, WORK, 2, ibit)
```

<= ENEW-EVAL の値をWORK, 誤差情報をibit にいれる。

```
EVAL=ENEW
```

```
   NITER=NITER+1
   IF(NITER.EQ.M) GO TO 5
   IF(ibit.le.IEPS) GO TO 2      <= IEPSはEPSをﾛｽﾄﾋﾞｯﾄ数で表現したもの
 5 CONTINUE
```

と、ﾌﾟﾛｸﾞﾗﾑの修正も少なく、演算時間に負担のかかる、除算も必要がない。

またこの場合の4倍精度での結果は、

```
   もとのﾌﾟﾛｸﾞﾗﾑ
 SIZE=       1000000
  EPS=
9.99999999999999999999999999999999E-0025
 ITERATION=              17
 EIGEN VALUE=
9.86960440108124119458166029137638      計算解
       RAMDA=
9.86960440108935861883449099987615      理論解
       RAMDA2=
9.86960440108124119458166013396532
       計算機理論解

   ﾛｽﾄﾋﾞｯﾄ使用ﾌﾟﾛｸﾞﾗﾑ
 SIZE=       1000000
  EPS=
9.99999999999999999999999999999999E-0025
 ITERATION=              17
 EIGEN VALUE=
9.86960440108124119458166029137638   ibit= 40 86
理論解、計算機理論解との差
       RAMDA=
9.86960440108935861883449099987615
       RAMDA2=
9.86960440108124119458166013396532   ibit= 40
理論解と計算機理論解の差
```

となっていて、理論解と計算機理論解の差は、計算解と計算機理論解の差より大きいので、精度向上を行うには、演算をより高精度で行うよりも、同じ演算精度で分割数Nを大きくすれば良い事を示している。このため、精度向上策として、ﾌﾟﾛｸﾞﾗﾑの修正も少なく、演算時間に負担のかからない方法がわかる。

5.5 Gauss-Legendre による数値積分

多重積分を数値計算で行う場合、直積法[2] などを使用して行うが、変数変換等で精度の向上が行えない場合があり、1次元Gauss-Legendre 積分を繰り返す事がある。ここでは、変数変換が使用できない事を想定し、1次元Gauss-Legendre 積分で精度の出にくい場合に誤差情報をどの様に活用するかを説明する.

例として図2の積分区間[0,1]での1/sqrt(1-x*x) の積分[6]を考える。倍精度での測定結果と解析は図2の様になっている。

通常、倍精度演算で求められる精度は誤差1.0d-6から1.0d-12 程度である。この結果を見ると、次元数をあげて、精度を向上させるのは得策ではない。

対策としては、

(1)4倍精度、8倍精度などの、より高精度な演算を行う

(2) 積分区間を分割して計算するがあげられる。

この場合、誤差曲線から見て、精度が出ないのは、演算精度ではなく、積分手法の問題と想定される。

実際n=1000 で計算して見ると、誤差は、倍精度、4倍精度、8倍精度= 6.1536q-4 で予想された結果といえる。

(2)の方式をとる場合、誤差1.0d-6 を考えた場合、次元数 n, 分割数m n*m=643029とする事も考えられるが、異なる次元数n1, n2 をとり、区間を2分割して誤差の大きい、区間を積分していく方法が効果的といえる。[7]

この積分の場合、誤差の出るのは、1-x*xの計算部分になる為、この最大ﾛｽﾄﾋﾞｯﾄ数は、n1=30 で9, n2=61 で11 となっている。すなわち倍精度の場合、積分区間、[1-2**(-53-11),1] での値は、精度がない事と、この区間に入る値で被積分関数を計算すると、ｾﾞﾛ割りが発生する可能性がある事がわかる。倍精度での計算結果2分割数 42回 で誤差0.64402d-08 43回でｾﾞﾛ割り発生。すなわちﾛｽﾄﾋﾞｯﾄ数により、その演算精度での誤差の最小値が推定できる事になる。

この精度で不足の場合、4倍精度で 2分割数 100回で、0.11565q-16、8倍精度で2分割数 200回で、0.11168q-31 で、必要とする精度におおじて、演算精度を選択すれば良い。またGauss-Legendre 積分で1-x*xのﾛｽﾄﾋﾞｯﾄ数が最大となるxは、事前にわかる為、より高精度演算を使用して、演算時間の増大なく、精度向上ができる。

5.6 総和計算

5.1 四則演算で使用した計算[1]

a=77617.0 b=33096.0

f=333.75*(b**6)+(a**2)*{11*(a**2)*(b**2)-(b**6)-121*(b**4)-2} +5.5*(b**8)+a/(2*b)

を4倍精度で要素数7の配列に分解し、

```
A(1)=333.75*(b**6)
A(2)=(a**2)*11*(a**2)*(b**2)
A(3)=-(a**2)*(b**6)
A(4)=-(a**2)*(121*(b**4))
A(5)=-(a**2)*2
A(6)=5.5*(b**8)
A(7)=a/(2*b)
```

精度を改善した総和計算を行うと、

answer=
-0.827396059946821368141165095479816

となり、正しい結果となる。

性能面でも、

Intel Xeon 3.06GHz (2cpu L2ｷｬｼｭ (512KB/cpu),
Os: Red Hat Enterprose Linux ES release 3 (taroon)
Kernel: 2.4.21-4 Elsmp on an i686
Intel Fortran V8.0 -O2 (注-1)で 1 cpu で実行

した結果、要素数 N=10,000,000 で通常 の4倍精度演算 0.7 秒、精度改善4倍精度0.41 秒

要素数 N=1,000,000 で仮数部の長さを変化させた場合、(図3) と、演算時間の増大という問題も解消されている。

(注－1)
IntelおよびIntel Xeonは, アメリカ合衆国およびその他の国におけるｲﾝﾃﾙｺｰﾎﾟﾚｰｼｮﾝまたはその子会社の商標または登録商標です。
Red Hatは, 米国およびその他の国でRed Hat, Inc. の登録商標若しくは商標です。
Linuxは, Linus Torvaldsの米国およびその他の国における登録商標あるいは商標です。

6. まとめ

以上の検討により、試作した多倍長ﾗｲﾌﾞﾗﾘ、および含まれる誤差情報、配列整数ﾃｰﾌﾞﾙの使用は、誤差の解析、精度改善を効率的に行うのに有効な事がわかったが、今後は、さらに高精度のﾙｰﾁﾝの追加、誤差解析、移行の労力の削減を検討して行く。

7. 参考文献

[1] A review on Interval Computation -Software and Applications
Chenyi Hu, Shanying Xu, and Xiaoguang Yang

[2] 杉原正顯、室田一雄　数値計算法の数理
岩波書店 2003

[3] J. Fujimoto, M. Igarashi, N. Nakazawa, Y. Shimizu and K. Tobimatsu,
Progress of Theoretical Physics, Supplement No. 100 (1990) 1-379

[4] 伊理正夫　数値計算　朝倉書店 1981

[5] 菊地文雄、山本昌宏 微分方程式と計算機演習
山海堂 1991

[6] 森正武 数値解析 共立出版　2002

[7] R. Piessens E. de Doncker-Kapenga
C. W. Uberhuber D. K. Kahaner
QUADPACK A Subroutine Package for Automatic Integration
Springer-Verlag Berlin Heidelberg New York
Tokyo 1983

8. 謝辞

精度検証にあたり御指導いただきました 高ｴﾈﾙｷﾞｰ加速器研究機構 藤本、石川、金子先生、に感謝致します。

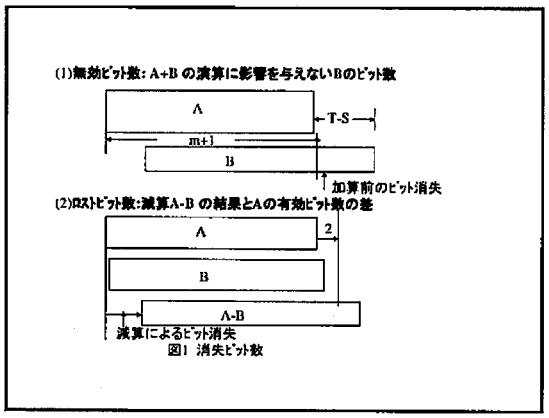


図1 消失ビット数

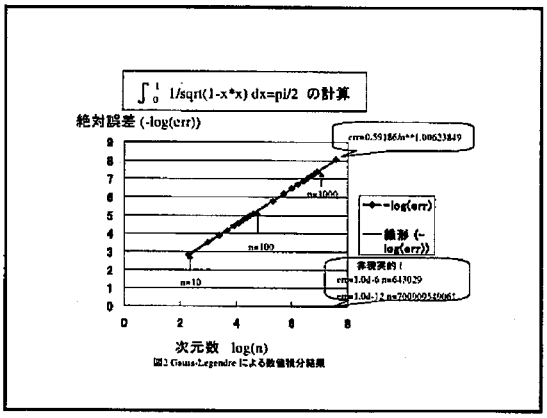
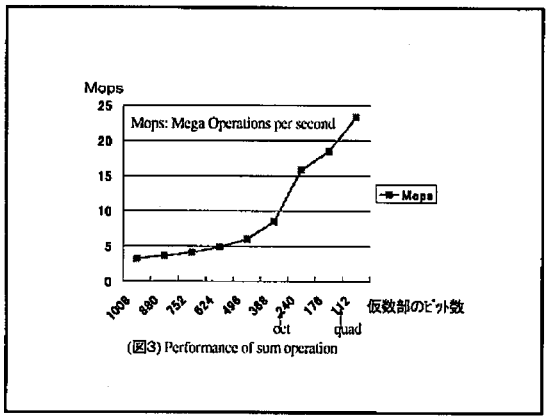


図2 Gauss-Legendre による数値積分結果



(図3) Performance of sum operation