多倍長計算手法

平成24年度第1四半期

目次

1. *DD*形式の利点

浮動小数点演算では、近接する２つの数a,bに対し$a-b$の演算で桁落ちが発生します。
例えば,$x=1-\varepsilon,(1>>\varepsilon>0)$で$1-x$を計算する場合があげられます。

倍精度演算と単精度変数を２つつなげた擬倍精度演算を比べた場合、有効ビット数は倍精度変数のほうが多いですが,$1-x$を計算した場合の桁落ちも、倍精度変数のほうが大きくなります。これは倍精度変数では$1-2^{-53}$　と１の間の実数xを表す事ができませんが、擬倍精度変数 x は$x=a+b$　$(a,b$単精度$)a=1,b<0$で表す事が可能になる事でわかります。

$x=1-10^{-i}$　の場合の$1-x$の値は以下の様になっています。

倍精度演算

```
i=  11   0.10000000082740371D-10
i=  12    0.99997782798785D-12
i=  13    0.10003109451872 66D-12
i=  14    0.99920072216264 09D-14
i=  15    0.99920072216264 09D-15
i=  16    0.11102230246251 57D-15
i=  17    0.00000000000000 00D+00
```

擬倍精度演算

```
i=  11  0.99999999600419 72D-11
i=  12  0.99999999600419 72D-12
i=  13  0.99999998245167 00D-13
i=  14  0.99999998245167 00D-14
i=  15  0.10000000036274 94D-14
i=  16  0.10000000168623 84D-15
i=  17  0.99999998377515 90D-17
```

同様の事は、多倍長演算でのDD形式と*ieee*形式の演算を比較する場合も同じ様になります。
すなわち、計算において、DD形式で表現できる数値範囲で済む場合は、精度上DD形式が有利になります。
例としては,

$\int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx = B(\alpha,\beta)[\alpha,\beta>0]$　　*ieee*形式が有利

$\int_0^1 \frac{1}{-x^2+x+\lambda^2}dx \fallingdotseq 4\ln(\frac{1}{\lambda})$　$(\lambda=10^{-60})$　*DD*形式が有利

があります。

これをより一般的な多次元数値積分に適用した例を以下に示します。

(0,1)の浮動小数点数xには,$\varepsilon < x < 1-\varepsilon$となる$1 >> \varepsilon > 0$が存在します。変数変換区間[0,1]の二重指数関数型積分法を使用する場合,$x=1-\varepsilon$は$s=\frac{\pi}{2}\sinh(t), x=\frac{e^s}{e^s+e^{-s}}$から

$\frac{1}{\varepsilon} \fallingdotseq e^{2s}$。重み係数は$w=\frac{\cosh(t)\times\pi}{(e^s+e^{-s})^2} \fallingdotseq \frac{\sqrt{4s^2+\pi^2}}{e^{2s}} \fallingdotseq 2s\varepsilon \fallingdotseq \varepsilon\ln(\frac{1}{\varepsilon})$。

tの値は$t=\sinh^{-1}(\frac{1}{\pi}\ln(\frac{1}{\varepsilon}))$となります。具体的な数値としては以下の様になります。

$\varepsilon=10^{-30}$の場合,$t=3.784, \varepsilon=10^{-300}$の場合,$t=6.086$となります。

刻み幅をhとすると分点数Nは$N=\frac{2t}{h}$となり、演算量的に

実際の計算では$h=10^{-3}$より大きな値をとる必要があります。

n次元有界領域Ωにおいて$D>0$,または$D<0$の場合,

$X=(x_1,x_2,...,x_n)\in\Omega$　　$x_i=1-\varepsilon$(iは1からnのなかの数)で

$\frac{w_1w_2...w_n}{h^n}D(X)$の絶対値E,積分値の絶対値Fとすると、

E/Fの値より使用する演算精度で得られる結果の誤差を推定する事ができます。

$INFRA\quad BOX$

$$I=\int_0^1\int_0^{1-x}\int_0^{1-x-y}\frac{1}{D^2}dzdydx$$

$$D=-sxy-tz(1-x-y-z)+(x+y)\lambda^2$$
$$+(1-x-y)(1-x-y-z){m_e}^2+z(1-x-y){m_f}^2$$

$s<0,t<0,|s|,|t|>>{m_e}^2>>\lambda^2$

変数変換をして整理すると

$$I=\int_0^1\int_0^1\int_0^1\frac{x(1-x)}{D^2}dzdydx$$

$$D=-sx^2y(1-y)+x\lambda^2+(1-x)^2[tz^2+(-t+{m_f}^2-{m_e}^2)z+{m_e}^2]$$

$$P=-sx^2y(1-y)+x\lambda^2+(1-x)^2{m_e}^2$$

$$Q=-sx^2y(1-y)+x\lambda^2+(1-x)^2[{m_e}^2-t(\frac{-t+{m_f}^2-{m_e}^2}{2t})^2]$$

とすると,$\frac{x(1-x)}{P^2}\geq\frac{x(1-x)}{D^2}\geq\frac{x(1-x)}{Q^2}$。以上から、

$\int_0^1\int_0^1\frac{x(1-x)}{P^2}dydx,\int_0^1\int_0^1\frac{x(1-x)}{Q^2}dydx$での分点での値と解析近似解を比較すれば誤差が推定できます。二重指数関数型積分法ではある分点を境に急速に分点での絶対値が減少しますので誤差推定は少ない点で計算できるという利点もあります。

いま、刻み幅$h,x=1-\varepsilon,y=\frac{\lambda^2}{-s}$として二重指数関数型での値は,$\frac{h^2\varepsilon^2\ln(\frac{1}{\varepsilon})(\frac{\lambda^2}{-s})\ln(\frac{-s}{\lambda^2})}{P^2}$　　$P=2\lambda^2$

これと,$I=\frac{1}{2(-s)(-t+{m_f}^2)}\ln(\frac{-s}{\lambda^2})\ln(\frac{(-t+{m_f}^2)^2}{{m_e}^2{m_f}^2})$

との比(相対誤差,Iを２で割っているのは,二重指数関数型積分の値の対称性より)は

$s=-500^2,t=-150^2,{m_f}=150,{m_e}=0.0005,\lambda=10^{-30}$

$h=0.5^6\times596/1024$

$\varepsilon=10^{-34}(ieee4倍精度)$で$6.6207\times10^{-8}$

$\varepsilon=10^{-32}(DD4倍精度)$で$6.2313\times10^{-4}$

$A(z)={m_f}^2$とし,

$$I\geq\int_0^1\int_0^1\frac{x(1-x)}{Q^2}dydx\qquad Q=-sx^2y(1-y)+x\lambda^2+{m_f}^2$$

で計算しても,$\varepsilon=10^{-32}(DD4倍精度)$で$1.3799\times10^{-4}$

となり10進6桁の精度を得るにはDD形式の４倍精度では困難な事がわかります。

実測結果は以下の様になりました。

DD形式４倍精度演算結果

```
h=0.5q0**6*593.0q0/1024.0q0
do  i=1,n
x=-595+i
t=x*h
y=pi*0.5q0*sinh(t)
x30(i)=exp(y)/(exp(y)+exp(-y))
gw30(i)=cosh(t)*pi/(exp(y)+exp(-y))**2
end do
```

```
   s=  -250000.00000000000000000000000000
   t=  -22500.00000000000000000000000000
  me=  0.49999999999999999999999999999978E-003
  mf=   150.00000000000000000000000000000
ramda=  0.99999999999999999999999999999605E-030


 result=  0.35615337207937480457621671164174208E-006
```

これを改善するために、

$x30(i) = \exp(y)/(\exp(y)+\exp(-y))$を$x30(i) = 1.0q0/(1.0q0+\exp(-2.0q0*y))$

とプログラムを変更します。

ieee形式４倍精度

```
s=   -250000.00000000000000000000000000
t=   -22500.000000000000000000000000000
fme=   5.0000000000000000000000000000000E-0004
fmf=    150.00000000000000000000000000000
ramda=   1.0000000000000000000000000000000E-0030
size=          1024
result=   3.5617329059598581079806543315774 47E-0007
```

DD形式４倍精度

```
 size=          1024
   s=   -250000.000000000
   t=   -22500.0000000000
mf**2=    22500.0000000000
me**2=   2.50000000000000E-007
ramda**2=   1.00000000000000E-060
 result=    0.35617366244969523 3D-06
```

変数変換区間を[−1,1]で行う必要がある場合は

$x30(i) = (\exp(y) - \exp(-y))/(\exp(y) + \exp(-y))$を

$y < 0$なら

$x30(i) = -(1.0q0 - \exp(-2.0q0 * y))/(1.0q0 + \exp(-2.0q0 * y))$

$y > 0$なら

$x30(i) = (1.0q0 - \exp(-2.0q0 * y))/(1.0q0 + \exp(-2.0q0 * y))$

と修正するだけで,精度改善が実施できます。

$N=4096$まで使用すると,$\lambda=10^{-100}$でも
解析近似解　　$=0.111864900351\times10^{-5}$に対し
変数変換区間$[0,1]=0.111864900547\times10^{-5}$
変数変換区間$[-1,1]=0.111864900549\times10^{-5}$
と非常に良い結果となり,$N=8192$まで使用
すると,$\lambda=10^{-150}$でも(λ^2の計算があるので*DD*形式
では限界ともいえます。）
解析近似解$=0.166327423367\times10^{-5}$に対し,
変数変換区間[0,1]で$0.166327137485\times10^{-5}$
と*ieee*形式では32倍精度でないと良い精度
の計算ができないものまで計算できます。

ただし、コーデイングでは,アンダーフロー,オーバーフローを防ぐ必要があります。

gw30(i1)*gw30(i2)*gw30(i3)/d**2

を

(gw30(i1)/d)*(gw30(i2)/d)*gw30(i3)

に修正が必要です。1に近いxを取る様にすると、gw30(i1)*gw30(i2)*gw30(i3）、d**2 **でアンダーフローが発生する場合があるためです。**

2.　精度保証

誤差を含まない変数配列a,bに対する,総和、内積

$s=s+\sum_{k=1}^{n}a(i), s=s+\sum_{k=1}^{n}a(i)*b(i)$　の計算でsの精度を

a,bと同じ精度を保証する方式を考えます。一般に
浮動小数点演算をマイクロ命令レベルで見ると、
*FPGA*などでわかるように,整数演算、シフト演算、
論理演算、マスク演算で行われています。これを
*FORTRAN*や*C*で行う(整数演算方式）事を考えます。
通常の演算では、浮動小数点演算方式と整数演算
方式を比べると,乗算処理はあまり差がなく、加減算の
処理に幾分差がでます。精度を保証する場合は、
乗算処理は、整数演算方式が有利となり,加減算処理
では,浮動小数点演算方式で行う事が非常に困難に
なります。

一般に0以外の浮動小数点数は$2^n(1.F)\ (0\le F<1)$で表せます
ので,0以外の浮動小数点数の絶対値最小の数2^mに対し
2^{2m}を単位とし,絶対値最大の数の二乗分の整数テーブル
(内積演算途中でのアンダーフロー,オーバーフローを
防ぐため)を作成しそれを使って加減算処理をします。
テーブルは正の数値用IPと負の数値用IMの２つを
持ち,$a(i), a(i)*b(i)$が正の場合は*IP*に,負の場合は絶対値を
IMに加算します。0の場合は何もする必要がありません。
すべてのデータの処理が終了した後,$IP\ge IM$なら$IP-IM$,
$IP<IM$なら,$IM-IP$を計算し,所要の精度の浮動小数点数
に変換します。($IP<IM$の場合は先頭ビットを１にします。）
これを,*T2K*で6倍精度,8倍精度の通常の内積演算と精度
保証付きの内積演算を行った結果は以下の様になっています。
ここで,通常の内積演算は浮動小数点演算方式で,精度保証
付きの内積演算は整数演算方式で行っています。

T2k 内積演算性能比較

内積演算	N=4,000,000*MPI数		演算量	8*MPI数	MFLOP

精度	MPI数	DD形式(通常演算)		IEEE形式(精度保証)	
		実行時間	性能	実行時間	性能
		(秒)	(MFLOPs)	(秒)	(MFLOPs)
6倍精度	64	0.909	563.26	0.807	634.45
	128	0.902	1135.25	0.812	1261.08
	256	0.904	2265.49	0.848	2415.09
	512	0.904	4338.75	0.811	5050.55
8倍精度	64	1.85	276.76	1.312	390.24
	128	1.85	553.51	1.344	761.9
	256	1.862	1099.89	1.331	1538.69
	512	1.867	2193.89	1.331	3077.39