論　考

情報落ち対策用和関数の作成

Making of the summation function for information loss measure

Author : 志多　友史　(Yuji Shida)
Date : 2017/5/11
Keywords : 情報落ち(information loss), 浮動小数点演算(floating point arithmetic),
倍精度浮動小数点数(double precision floating point number),
浮動小数点数演算標準(IEEE 754), C 言語(C programming language),
frexp: A function of C programming language

Abstract:==========

本稿では数値計算の際に生じる誤差（情報落ち）について、これを軽減するための関数を作成し、誤差の低減にどの位寄与するか調査を行った。絶対値の大きな値と小さな値を足し合わせる時に、時々、計算機の有効桁数を超えるような演算が行われる。情報落ちとは、有効桁数の範囲外となって切り捨てられた値の蓄積により生じる誤差の事で、数値積分などで棒大な数の短冊状の領域を足し合わせる場合等で注意しなければならない。

In this report, I examined the influence of computation error (information loss). The content is making of summation function for information loss measure and compared non-measured calculation result and measured one. In summation between large absolute value and small one, sometimes, unfavorable calculations which surpasses effective digit are carried out on computers. Here, "information loss" means a computation error caused by accumulation of values which became outside of the effective digit and were cut off. For example, we have to calculate carefully when calculate numerical integration, like summation of many small value.

==========

１．序論(Introduction)

計算機の発達によって科学技術系の数値計算がよく行われるようになった。また自身も数値計算をテーマに色々と趣味のプログラミングを行っているが、計算機の計算精度には限界がある。その問題の一つに「情報落ち」というものがある。本稿では、この問題を改善する方法について検討し、情報落ちの影響が極力小さくなるような関数を作成し、その効果を評価する。

２．理論(Theory)

まず、「情報落ち」とは何かについて記す。図２．１に計算機のメモリを簡略化した模式図を示す。今、2 つの値の和($5.678\times10^{-5}+1.234\times10^{3}$)を計算する時、この計算機の有効桁（仮数桁）は 9 桁であるため、本来ならば"$1.23400005678\times10^{3}$"となるはずが"$1.23400005\times10^{3}$"となってしまう。結果として"678"は消えて無くなってしまう。このように数値計算に際して、下位の桁の情報が有効に計算に寄与しなくなる事を「情報落ち」という。

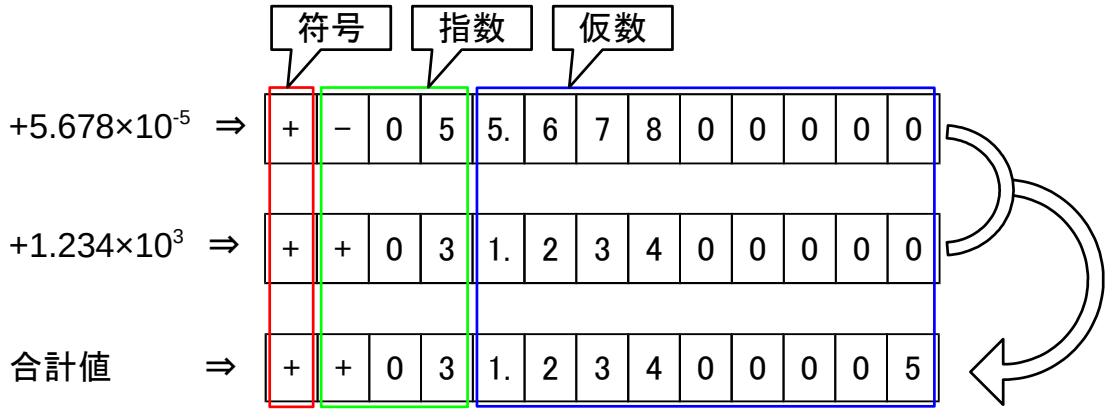

図２．１　情報落ち

一回きりの計算であれば、これも止む無しという事で受け入れる事も可能であるが、図２．２のように数値積分 $\int_{x_s}^{x_e} f(x)dx \simeq \sum_{i=1}^{N} S_i$ のような計算を行う場合、後半になると既に足し合わされた値 $\sum_{i=1}^{n-1} S_i$ と次に足し合わせる値 S_n の絶対値の差は非常に大きなものになっていく。従って、安直に for 文を用いて加算を繰り返すと精密な計算結果を得ることが困難になる。

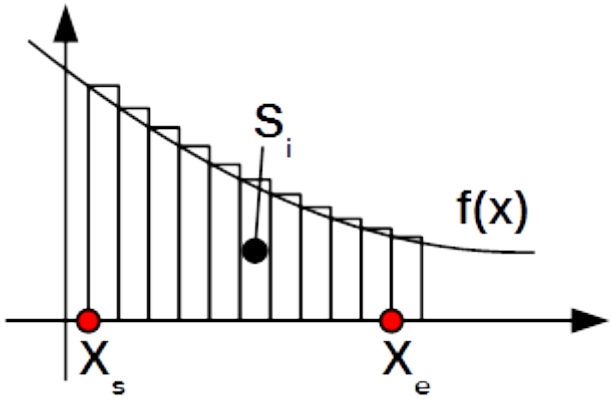

図２．２　数値積分

このような問題を解決（誤差の低減）するため、本稿では特別な加算操作を行う関数の作成を考える。具体的には、与えられた数値を指数毎に振り分けて対応する配列に足し合わせていき、最後に絶対値の小さい値が入った配列から順番に足し上げていくというものである。この操作のフローチャートを図２．３に示す。

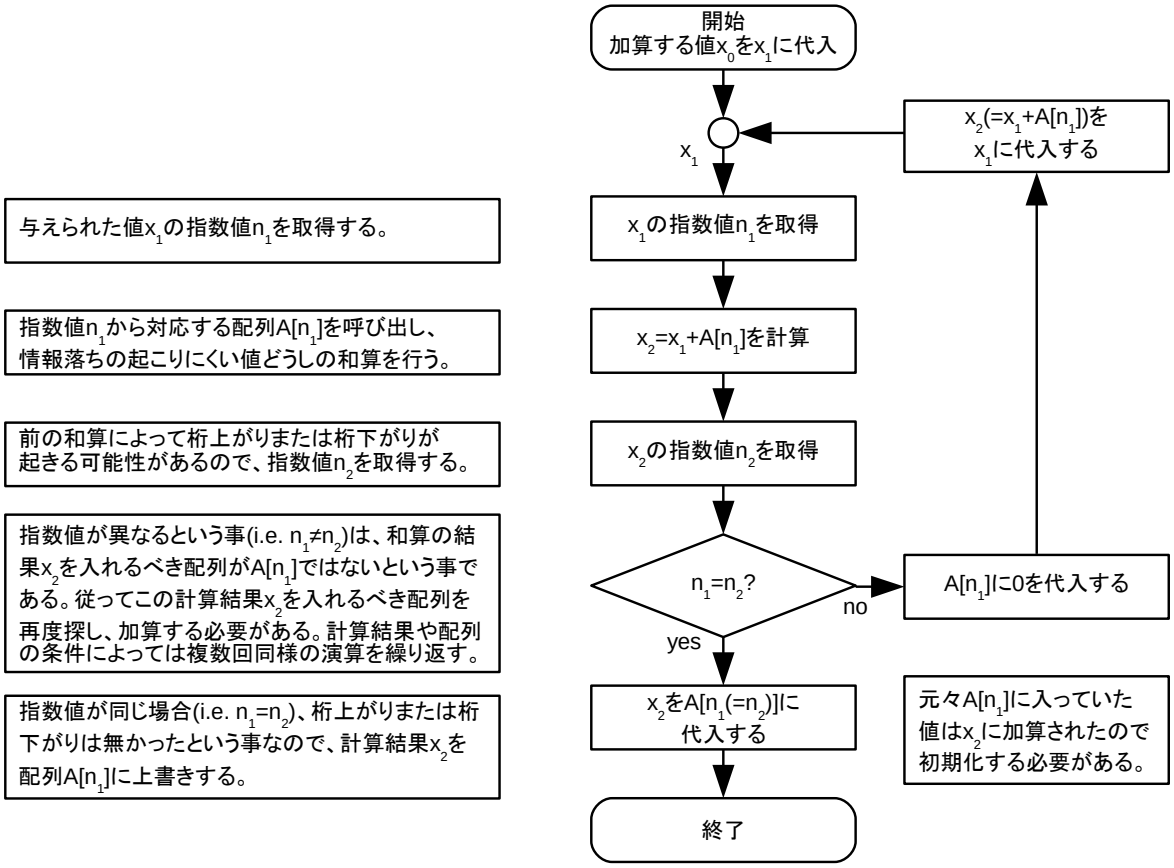

図２．３　値の振り分けと加算

次に上図の演算フロー実現する上で必要となる情報を以下に記す。

（１）浮動小数点数演算標準(IEEE 754)による倍精度浮動小数点数（double 型）の定義

倍精度浮動小数点数の計算機内部における表現は下図の通りである。

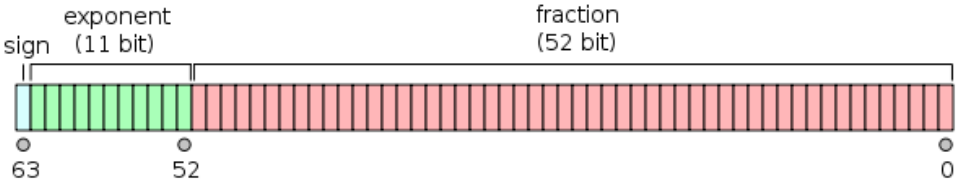

図２．４　計算機内部における double 型の表現（ビットレイアウト）
（wikipedia「倍精度浮動小数点数」より引用）

図中の色付き部分の意味は次の通りである。

・sign（符号ビット）："0"で正の値、"1"で負の値

・exponent（指数部）：2 の指数値"-1022～+1023(1～2046,bias:1023)"の他に、
"-1023(0,bias:1023)"で非正規化数、"1024(2047,bias:1023)"で無限大を表す。

・fraction（仮数部）：仮数部であり、けち表現で値が格納される。
※けち表現とは、"X.YYY…"の"X"が常に"1"になるように桁を揃え、
"YYY…"の部分を格納する事をいう。

また最終的に計算機上において格納された値は下式のように表される。

$$(-1)^{sign}\,2^{(exponent-1023)}\left(1+\sum_{i=1}^{52} b_{52-i}2^{-i}\right) \quad \text{or} \quad (-1)^{sign}\,2^{exponent-1023}\left(1.b_{51}b_{50}b_{49}\cdots b_2b_1b_0\right)$$

（２）frexp 関数の機能

frexp 関数は次のように使用した場合、戻り値（double 型）として"frac"に仮数を返し、"exp"に指数を格納する。ここで、戻り値である仮数の範囲は"0.5≦frac<1"となる事、また格納される指数の基数は"2"であることに注意する必要がある。

frac = frexp (value, &exp)

３．方法(Method)

前章で示した計算方法をC言語のプログラムで具体的に作り上げる。作成した関数は下記に記すもので、引数は動作モード、使用配列（2048個のdouble型配列）のアドレス及び足し合わせる値（double型）である。

高精度和関数(High Precision Summation – Double type)

```
double HPS_D2048(int mod, double buf[], double value)
{
	int i,bn1,bn2;
	double bx1,bx2,ans;

	switch(mod){
		case 0:		※配列の初期化
			for(i=0;i<2048;i++) buf[i]=0.0e0;
			break;
		case 1:		※値の振り分け及び計算
			bx1=value;
			while(1){
				frexp(bx1,&bn1);		※指数値を取得する。
				bn1+=1023;			※配列番号に変換する。
				bx2=bx1+buf[bn1];		※対応する配列との和を計算する。
				frexp(bx2,&bn2);		※再度、指数値を取得する。
				bn2+=1023;			※配列番号に変換する。
				if(bn1==bn2){			※計算前後の配列番号を比較する。
					buf[bn1]=bx2;	※一致したので現配列に代入する。
					break;			※処理を終了する。
				}else{
					buf[bn1]=0.0e0; ※現対応配列を初期化する。
					bx1=bx2;		※計算値を代入し、再度ループを回す。
				}
			}
			break;
		case 2:		※全数値の合計と返却
			ans=0.0e0;
			for(i=0;i<2048;i++) ans+=buf[i];		※小さな方から計算を行う。
			return ans;
		default:
			break;
	}

	return 0.0e0;
}
```

４．結果(Results)

前章で作成したプログラムの実行結果を示す。なお計算環境は次の通りである。

CPU：Intel Core i5-4570 3.20GHz

MEM：DDR3 Dual Channel 32GB(8GB*4) 1333MHz

OS：Windows8 64bit

（１）代数関数の場合

次のような代数関数の定積分(1≦x≦11)を考える。

$$f(x)=x^3+x^2+\sqrt{(x)}+\frac{1}{x}-\frac{1}{x^2}$$

理論解を求めると

$$F(x)=\frac{1}{4}x^4+\frac{1}{3}x^3+\frac{2}{3}x^{\frac{3}{2}}+\ln x+\frac{1}{x}+Const$$

$$\left[F(x)\right]_{x=1}^{x=11}=\frac{135358}{33}+\frac{22}{3}\sqrt{11}+\ln 11$$

となり、計算結果は 4.1284773861596477e3 となるはずである。

表４．１　代数関数の場合の結果

分割数	通常和算	高精度和算	計算結果の差
1.0×10^{1}	3.4338474787090063e3	3.4338474787090058e3	0.0000000000000005e3
5.0×10^{1}	3.9844996454876396e3	3.9844996454876391e3	0.0000000000000005e3
1.0×10^{2}	4.0561729618825830e3	4.0561729618825834e3	0.0000000000000004e3
5.0×10^{2}	4.1139660149360034e3	4.1139660149360061e3	0.0000000000000027e3
1.0×10^{3}	4.1212185451794285e3	4.1212185451794257e3	0.0000000000000028e3
1.0×10^{4}	4.1277512180786071e3	4.1277512180786043e3	0.0000000000000028e3
1.0×10^{5}	4.1284047665117059e3	4.1284047665117141e3	0.0000000000000082e3
1.0×10^{6}	4.1284701241664588e3	4.1284701241664570e3	0.0000000000000018e3
1.0×10^{7}	4.1284766599598397e3	4.1284766599600425e3	0.0000000000002028e3
1.0×10^{8}	4.1284773135393734e3	4.1284773135396827e3	0.0000000000003093e3
1.0×10^{9}	4.1284773788980219e3	4.1284773788976509e3	0.0000000000003710e3

（２）三角関数の場合

次のような三角関数の定積分(-5≦x≦5)を考える。

$$f(x)=-\cos 3x+5\sin x$$

理論解を求めると

$$F(x)=-\frac{1}{3}\sin 3x-5\cos x+Const$$

$$\left[F(x)\right]_{x=-5}^{x=5}=-\frac{2}{3}\sin 15$$

となり、計算結果は-4.3352522677141120e-1 となるはずである。

表４．２　三角関数の場合の結果

分割数	通常和算	高精度和算	計算結果の差
1.0×10^{1}	4.7485063123790416e0	4.7485063123790408e0	0.0000000000000008e0
5.0×10^{1}	5.3848351417821738e-1	5.3848351417821638e-1	0.0000000000000100e-1
1.0×10^{2}	4.9193237394367151e-2	4.9193237394367595e-2	0.0000000000000444e-2
5.0×10^{2}	-3.3750273393294167e-1	-3.3750273393294350e-1	0.0000000000000183e-1
1.0×10^{3}	-3.8554649815853609e-1	-3.8554649815852038e-1	0.0000000000001571e-1
1.0×10^{4}	-4.2873028025411780e-1	-4.2873028025412774e-1	0.0000000000000994e-1
1.0×10^{5}	-4.3304576138264489e-1	-4.3304576138264239e-1	0.0000000000000250e-1
1.0×10^{6}	-4.3347728052541934e-1	-4.3347728052516743e-1	0.0000000000025191e-1
1.0×10^{7}	-4.3352043215011565e-1	-4.3352043214971148e-1	0.0000000000040417e-1
1.0×10^{8}	-4.3352474730812268e-1	-4.3352474730926760e-1	0.0000000000114492e-1
1.0×10^{9}	-4.33525178825864 55e-1	-4.33525178825198 59e-1	0.0000000000066596e-1

（３）指数関数の場合

次のような三角関数の定積分(0≦x≦10)を考える。

$$f(x)=\exp(-6x)$$

理論解を求めると

$$F(x)=-\frac{1}{6}\exp(-6x)+Const$$

$$\left[F(x)\right]_{x=0}^{x=10}=\frac{1}{6}\left\{1-\exp(-60)\right\}$$

となり、計算結果は 1.6666666666666666e-1（e^{-60}<<1 のため、ほぼ 1/6）となるはずである。

表４．３　指数関数の場合の結果

分割数	通常和算	高精度和算	計算結果の差
1.0×10^{1}	1.0024849116568444e0	1.0024849116568446e0	0.0000000000000002e0
5.0×10^{1}	2.8620255213866663e-1	2.8620255213866663e-1	0.0000000000000000e-1
1.0×10^{2}	2.2163692151608705e-1	2.2163692151608708e-1	0.0000000000000003e-1
5.0×10^{2}	1.7686661868311773e-1	1.7686661868311790e-1	0.0000000000000017e-1
1.0×10^{3}	1.7171666366692381e-1	1.7171666366692381e-1	0.0000000000000000e-1
1.0×10^{4}	1.6716716666636447e-1	1.6716716666636666e-1	0.0000000000000219e-1
1.0×10^{5}	1.6671667166664883e-1	1.6671667166666662e-1	0.0000000000001779e-1
1.0×10^{6}	1.6667166671649825e-1	1.6667166671666667e-1	0.0000000000016842e-1
1.0×10^{7}	1.6666716666557124e-1	1.6666716666716669e-1	0.0000000000159545e-1
1.0×10^{8}	1.6666671665067528e-1	1.6666671666667165e-1	0.0000000001599637e-1
1.0×10^{9}	1.6666667150634734e-1	1.6666667166666671e-1	0.0000000016031937e-1

（４）立体角の計算アルゴリズムに適用した場合（自著「立体角の計算アルゴリズムの作成」参照）

立体角の計算アルゴリズムには、原点を中心とした正四面体の 4 つの頂点の内の 3 つを指すベクトルを設定し、計算結果がπすなわち 3.1415926535897932e0 となるよう計算を行った。

なお立体角の計算アルゴリズム(SoST)における分割数とは、三角形の一辺の分割数を指す事から、足し合わせが行われる要素の数は分割数の二乗となる事に注意されたい。

表４．４　立体角の計算結果

分割数	通常和算	高精度和算	計算結果の差
5.0×10^{0}	3.1823874204863389e0	3.1823874204863389e0	0.0000000000000000e0
1.0×10^{1}	3.1505195258187841e0	3.1505195258187841e0	0.0000000000000000e0
5.0×10^{1}	3.1419518032526397e0	3.1419518032526343e0	0.0000000000000054e0
1.0×10^{2}	3.1416824579788818e0	3.1416824579788734e0	0.0000000000000084e0
5.0×10^{2}	3.1415962459824383e0	3.1415962459824249e0	0.0000000000000134e0
1.0×10^{3}	3.1415935516896081e0	3.1415935516896467e0	0.0000000000000386e0
5.0×10^{3}	3.1415926895136161e0	3.1415926895138093e0	0.0000000000001932e0
1.0×10^{4}	3.1415926625708259e0	3.1415926625707971e0	0.0000000000000288e0
5.0×10^{4}	3.1415926539507661e0	3.1415926539490333e0	0.0000000000017328e0

５．考察(Discussion)

それぞれの計算結果の差を見ると、（1 つの要素の大きさ）<<（足し合わせる要素の数）となるにつれて差が大きくなる事が分かる。しかし、それでも有効数字の小数第 10 位程度までは情報落ち対策の有無に関わらず同じ結果であった。

６．結言(Summary)

思っていたほど効果が無かったのには驚いた。

７．文献(References)

［１］：http://www.cc.kyoto-su.ac.jp/~yamada/programming/float.html
［２］：https://en.wikipedia.org/wiki/Double-precision_floating-point_format
［３］：https://ja.wikipedia.org/wiki/IEEE_754#.E6.8B.A1.E5.BC.B5.E7.B2.BE.E5.BA.A6.E5.BD.A2.E5.BC.8F
［４］：http://mimosa-pudica.net/denorm.html
［５］：https://cpprefjp.github.io/reference/cmath/frexp.html

８．著者(Author)

氏名：志多　友史（工学修士）
略歴：
2011 年：下位国立大学　工学部電気系学科卒業
2013 年：同大学大学院　工学研究科修了
2013 年：研究開発機関へ就職
興味：物理・数学・コンピュータ・電気電子工作

９．備考(Notes)

特になし。