Scilab における 4 倍精度演算環境 Toolbox 「QuPAT」の開発

DEVELOPMENT OF QUADRUPLE PRECISION ARITHMETIC TOOLBOX **QuPAT** ON SCILAB

齊藤翼[1)], 石渡恵美子[2)], 長谷川秀彦[3)]
Tsubasa SAITO, Emiko ISHIWATA and Hidehiko HASEGAWA

[1)]東京理科大学大学院（〒162-8601 東京都新宿区神楽坂 1-3, j1409612@ed.kagu.tus.ac.jp）
[2)]東京理科大学（〒162-8601 東京都新宿区神楽坂 1-3 ）
[3)]筑波大学（〒305-8550 茨城県つくば市春日 1-2）

We develop a quadruple precision arithmetic environment QuPAT (Quadruple Precision Arithmetic Toolbox) using the numerical software package Scilab as a toolbox. Based on Double-Double (DD) arithmetic, QuPAT uses only a combination of double precision arithmetic operations. QuPAT has three main characteristics: (i) the same operator is used for both double and quadruple precision arithmetic; (ii) both double and quadruple precision arithmetic can be used at the same time, and also mixed precision arithmetic is available; (iii) QuPAT is independent of hardware and operating systems. We also show the effectiveness of QuPAT on several numerical examples.

Key Words : *quadruple precision arithmetic, mixed precision, Scilab*

1. はじめに

コンピュータにおける計算では，浮動小数点数による近似計算が主流である．しかし，浮動小数点演算において，桁落ちや丸め誤差，情報落ちなどの影響は避けられない．現在標準的に用いられている，IEEE754 によって規定された倍精度浮動小数点数は，有効桁数が 10 進 16 桁であり，連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の反復解法において，丸め誤差の影響で残差が停滞する場合や，残差は減少しても誤差が減少しない場合などが知られている．このような誤差を少なくするためには多倍長演算が必要であるが，特別なハードウェアのない通常の計算機環境で実現するのは難しい．

本研究では, 4 倍精度演算を対話型数値計算ソフトウェア Scilab[1] 上に，ツールボックス “QuPAT (Quadruple Precision Arithmetic Toolbox)” として実装した．QuPAT は，4 倍精度演算を 2 つの倍精度浮動小数点数を用いて倍精度演算のみで実行する Double-Double（DD）演算に基づいている．QuPAT の特徴は，(i) 倍精度演算と 4 倍精度演算で同じ演算子が利用できること，(ii) 倍精度演算と 4 倍精度演算と同時に実行したり，混合演算も実行できること，(iii) Scilab が動作する環境であれば，ハードウェアや OS に依存しないこと，である．本稿ではまず DD 演算を紹介し，QuPAT の実装方法やその特徴について述べ，利用例を通じて QuPAT の有効性を示す．

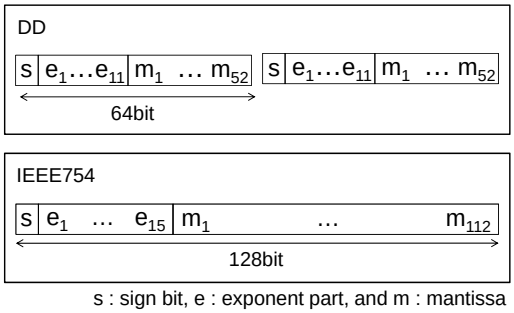
2. DD 演算

QuPAT で用いた DD 演算は，Dekker[2] や Knuth[3] らによる誤差のない厳密な浮動小数点数アルゴリズムに基づいている．DD の数は，2 つの倍精度浮動小数点数で表現する．実数 α を DD の数 A で表現する場合，Ahi（上位）と Alo（下位）を用いて，次のように表す：

$$\mathrm{Ahi} = (\alpha\text{を倍精度に丸めた値}) \tag{1}$$

$$\mathrm{Alo} = ((\alpha - \mathrm{Ahi})\text{ を倍精度に丸めた値}) \tag{2}$$

この方式と，IEEE754 準拠の 4 倍精度浮動小数点数の構成ビット数を図 1 に示す．IEEE754 準拠の 4 倍精度数の有効桁数が 10 進で 34 桁であるのに対し，DD の数は 10 進で 31 桁である．これは IEEE754 準拠の 4 倍精



図–1 DD の数と IEEE754 準拠 4 倍精度数の構成 bit 数

度数よりも短いが，倍精度演算のみで実装できるというメリットがある．

DD 演算を実装するためには，倍精度演算が誤差なしに計算できなければならない．そこで，2 つの倍精度浮動小数点数を用いて倍精度演算の結果を保持する，4 つのアルゴリズムを定義する．

⟨**1**⟩ **two_ sum**：倍精度加算と倍精度に収まりきらない誤差分を計算する

```
[s,e] = two_sum(a,b)
  s = a + b;
  v = s - a;
  e = (a - (s - v))+ (b - v);
```

⟨**2**⟩ **fast_ two_ sum**：two_ sum と同様（ただし $|a| \geq |b|$ を仮定）

```
[s,e] = fast_two_sum(a,b)
  s = a + b;
  e = b - (s - a);
```

⟨**3**⟩ **split**：倍精度浮動小数点数の仮数部を上位 26bit と下位 26bit に分割する

```
[h,l] = split(a)
  t = 134217729 * a;
  h = t - (t - a);
  l = a - h;
```

⟨**4**⟩ **two_ prod**：倍精度乗算と倍精度に収まりきらない誤差分を計算する

```
[p,e] = two_prod(a,b)
  p = a * b;
  (ah, al) = split(a);
  (bh, bl) = split(b);
  e = ((ah * bh - p)
        + ah * bl + al * bh) + al * bl;
```

DD 演算は上記のアルゴリズムを用いて，倍精度演算の組み合わせで次の ⟨5⟩,⟨6⟩ のように定義される．ただし A=Ahi+Alo，B=Bhi+Blo，C=Chi+Clo とする．

⟨**5**⟩ **DD 加算**：

```
C = add(A,B)
  [sh,eh] = two_sum(Ahi,Bhi);
  [sl,el] = two_sum(Alo,Blo);
  se = eh + sl;
  [sh',se'] = fast_two_sum(sh,se);
  see = se' + el;
  [Chi,Clo] = fast_two_sum(sh',eh');
```

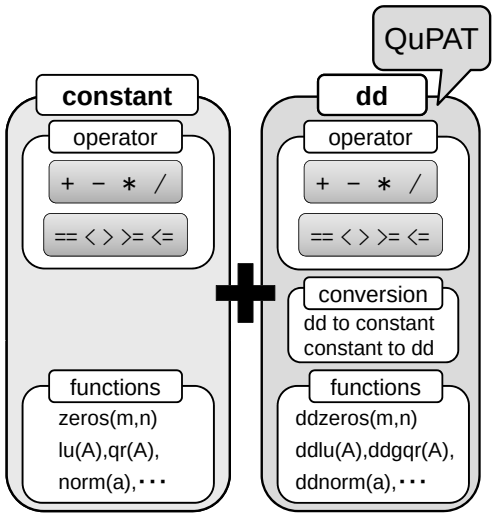
⟨**6**⟩ **DD 乗算**：

```
C = mul(A,B)
    [p1,p2] = two_prod(Ahi,Ahi);
    p2 = p2 + Ahi * Blo;
    p2 = p2 + Alo * Bhi;
    [Chi,Clo] = fast_two_sum(p1,p2);
```

たとえば，DD 加算には 20 回，DD 乗算には 24 回の倍精度演算が必要となる．このようにして，既存の倍精度演算のみの環境でも手軽に 4 倍精度演算を扱える．実装の詳細は [4] にある．

3. QuPAT の実装

Scilab はフランスの INRIA (Institut National de Recherche en Informatique et en Automatique) で開発されているオープンソースでフリーの数値計算ソフトウェアである．本節では，QuPAT の Scilab への実装について述べる．QuPAT では 4 倍精度数に該当する dd 型を新たに定義し，既存の倍精度数に該当する constant 型と同時に利用できるようにした．また，演算子を多重定義し，倍精度，4 倍精度演算に加え，混合精度演算も全て同じ演算子 $(+, -, *, /)$ が使えるようにした．QuPAT を利用することで，通常の Scilab の倍精度演算とほぼ同様かつ同時に，4 倍精度演算を扱うことができる（図 2）．なお，QuPAT は Scilab の言語機能のみで作成されているため，Scilab が動作する環境であればハードウェアや OS には依存しない．



図–2 Scilab と QuPAT の関係

(1) DD の数の導入

Scilab では関数 “tlist” を用いて新しいデータ型を定義できる．関数 tlist は Scilab においてオブジェクトを生成するもので，次のように記述する．

```
tlist(typ, a1, ..., an)
```

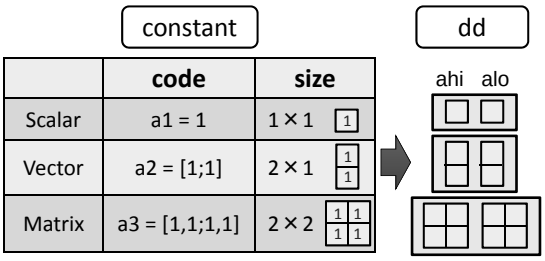
ここで typ は型の名前で，ユーザが自由に設定することができる．a1, ..., an は含まれる要素である．このように作成した型は，C++ におけるクラスのように，複数のデータをひとつのデータとみなす．

Scilab では倍精度浮動小数点数がデータ型 “constant” によって定められている．そこで関数 tlist を利用して，DD の数のために新しいデータ型 “dd” を定義した．QuPAT で dd 型の変数を生成するために，以下の関数を定めている．

```
function a = dd(ahi,alo)
  a = tlist(['dd','hi','lo'],ahi,alo);
endfunction
```

たとえば，a=dd(1,0) のように入力すると，値 1.0 を持つ dd 型変数 a が生成される．ここで [‘dd’,‘hi’,‘lo’] は，データ型と各要素の名前を表し，ahi, alo はそれぞれ倍精度浮動小数点数を持つ constant 型の変数を表している．dd 型変数の上位部 ahi（下位部 alo）を参照する際には，a.hi（a.lo）と入力すれば良い．上位部を参照することは，dd 型から constant 型に変換することと同義である．また QuPAT では，constant 型の変数を dd 型に変換する関数 d2dd を定めている．

Scilab ではスカラー, ベクトル, 行列がすべて constant 型として扱われているため，われわれは新しく dd 型を定義するだけでスカラー，ベクトル，行列のすべてを DD の数へ自然に拡張できる（図 3）．



図–3　constant 型から dd 型への拡張

(2)　演算の定義

Scilab では新しいデータ型に対して，四則演算などの演算子を多重定義でき，新しいデータ型に対する演算では，予め定められた関数が呼ばれる．dd 型に対する四則演算として DD 演算を演算子多重定義し，constant 型，すなわち倍精度数と同じ演算子で四則演算を行えるようにした．

(3)　関数の定義

既存の Scilab 関数は，新しいデータ型を引数に持つことができない．そこで，既存の関数に対応する dd 型変数用の関数を，dd<function_name> という名前で定めた．たとえば，平方根を求める Scilab 関数 ‘sqrt’ に対して dd 型変数の平方根を求める ‘ddsqrt’ という関数を作成した．他にも絶対値を求める関数や，内積，ノルムを計算する関数など，20 個程度の関数を作成した．

4.　QuPAT の使用例

QuPAT を用いれば，プログラムをほとんど変更することなく DD 演算を利用でき，倍精度演算では理論どおりにならない場合などで，簡単により高精度な演算を試行できる．本節では倍精度演算と DD 演算を比較し，演算精度に対する依存性を検証する．Scilab は 5.1.1 を用いている．

(1)　2 次方程式の求解

2 次方程式 $ax^2 + bx + c = 0$ の解を求める際には，理論的には解の公式

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{3}$$

を用いればよい．しかし係数が $a = c = 1, b = -10^6$ のような場合には，分子の計算で桁落ちを引き起こす可能性があり，倍精度演算では計算に工夫が必要となる．

まず，Scilab でのプログラムを示す．2 つの解 x_1, x_2 と，それぞれの残差を計算する．倍精度演算の場合は以下のとおりである．

```
a = 1;b = -1000000;c = 1;
D = b*b - 4.0*a*c;
x1 = (-b+sqrt(D))/(2.0*a);
x2 = (-b-sqrt(D))/(2.0*a);
res_x1 = abs(a*x1*x1 + b*x1 + c);
res_x2 = abs(a*x2*x2 + b*x2 + c);
```

DD 演算の場合は変数の型を dd 型で定義し，平方根や絶対値の関数を少し変更する必要があるが，単純な修正でよい．

```
a = dd(1,0);b = dd(-1000000,0);c = dd(1,0);
D = b*b - 4.0*a*c;
x1 = (-b+ddsqrt(D))/(2.0*a);
x2 = (-b-ddsqrt(D))/(2.0*a);
res_x1 = ddabs(a*x1*x1 + b*x1 + c);
res_x2 = ddabs(a*x2*x2 + b*x2 + c);
```

計算結果を表 1 に示す．x_1 はほぼ同じだが，x_2 は倍精度演算と DD 演算で大きく異なっている．解 x_2 の残差は，倍精度演算の場合 10^{-6} 程度であるが，DD 演算では 10^{-22} と大きく改善されている．Mathematica の NSolve で計算すると，

```
{x -> 1.00000000000100000000002*10^-6},
{x -> 999999.9999989999999999990}
```

表–1　解の公式による計算結果の比較

	double	DD
$x_1(\times 10^5)$	9.999999999989	9.999999999989
$ax_1^2+bx_1+c$	0.0	0.0
$x_2(\times 10^{-6})$	1.000007614493	1.000000000001
$ax_2^2+bx_2+c$	7.614×10^{-6}	3.329×10^{-22}

となる．倍精度演算に比べ，DD 演算では精度良く解が求まっていることがわかる．このようにプログラムに工夫が必要となる場合でも，より理論に近い形でプログラムが記述できる．

(2)　連立一次方程式の反復解法

連立一次方程式 $A\boldsymbol{x}=\boldsymbol{b}$ の反復解法である GCR(m) 法は，理論上は各反復で残差ノルムが減少し，高々次元数の反復で解に収束するが，浮動小数点演算では残差が停滞する場合や，誤差が減少しない場合などが知られている．ここでは MatrixMarket[5] の arc130 に対して，倍精度演算と DD 演算の結果を比較する．プログラムの変更点は変数の定義，内積，ノルム計算である．arc130 の次元数は 130 で，Scilab 関数 'cond' を用いて条件数を求めると 6.05×10^{10} である．最大反復回数は 1000，リスタート周期は $m=50$，右辺ベクトルは解 $\boldsymbol{x}^*$ の値がすべて 1 となるように定め，$\boldsymbol{x_0}=\boldsymbol{0}$ とした．停止条件は以下のように定めた．ただし，$\boldsymbol{r}_k=\boldsymbol{b}-A\boldsymbol{x}_k$ である．

$$\|\boldsymbol{r}_k\|_2 \le 10^{-12}\ \|\boldsymbol{r}_0\|_2 \qquad \text{(double)} \qquad (4)$$

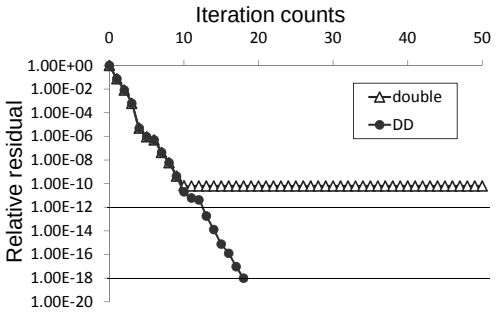
$$\|\boldsymbol{r}_k\|_2 \le 10^{-18}\ \|\boldsymbol{r}_0\|_2 \qquad \text{(DD)} \qquad (5)$$

結果を表 2 に示す．理論的には最大反復 130 回で解に収束するはずが，倍精度演算の場合は相対残差が 1.0×10^{-10} 程度で停滞し収束しない（図 4）．DD 演算の場合，反復 18 回で相対残差が 9.89×10^{-19} となる．これらの違いは計算精度の影響である．倍精度浮動小数点数の有効桁数は 10 進で 16 桁のため，条件数が 10^{10} であるこの問題に対して，十分な精度の解を得ることができなかった．しかし DD 演算を用いることで，理論どおりのふるまいを確認できた．

QuPAT は少量のプログラム変更で DD 演算へ移行できるため，倍精度演算で理論とは異なる結果となった場合にも，簡単に DD 演算で誤差の影響を確認できる．

表–2　反復回数，相対残差ノルム，相対誤差ノルムの比較

	反復回数	$\|\boldsymbol{r}\|_2/\|\boldsymbol{r}_0\|_2$	$\|\boldsymbol{x}-\boldsymbol{x}^*\|_\infty/\|\boldsymbol{x}^*\|_\infty$
double	1000	6.28e-11	9.27e+00
DD	18	9.89e-19	2.74e-08



図–4　収束履歴

5.　まとめ

倍精度演算での計算結果を検証するためには，より高精度な演算環境が必要となる．本研究では，Scilab 上で手軽に利用できる 4 倍精度（DD）演算ツールボックス "QuPAT(Quadruple Precision Arithmetic Toolbox)" を作成した．QuPAT は web サイト [6] からダウンロード可能である．QuPAT を利用すれば，Scilab の対話型プログラムの特徴を活かしつつ，わずかなプログラムの変更で 4 倍精度での検証を行える．なお，QuPAT は Scilab が動作する環境であればハードウェアや OS には依存しないが，演算を高速化するには Scilab の機能である C 言語とのリンクを利用すればよい．しかし C 言語のプログラムを導入することで，ツールボックスが計算機や OS に依存するようになる．

現在の QuPAT には基本的な四則演算や，主に線形計算に必要な関数を実装している．しかし，三角関数などはまだ実装していないため，今後はより便利なツールボックスとして機能を充実させていきたい．

参考文献

1) Scilab, http://www.scilab.org/
2) T. J. Dekker, A Floating-Point Technique for Extending the Available Precision, Numer. Math. Vol. 18, 224-242(1971) .
3) D. E. Knuth, The Art of Computer Programming, vol. 2. Addison Wesley (1969).
4) T. Saito, E. Ishiwata and H. Hasegawa, Development of Quadruple Precision Arithmetic Toolbox QuPAT on Scilab, ICCSA2010, Proceedings Part II, LNCS 6017, 60-70 (2010).
5) MatrixMarket, http://math.nist.gov/MatrixMarket/
6) QuPAT, http://www.mi.kagu.tus.ac.jp/qupat.html