B-007

MATLAB 上での高精度演算の実装について
Implementation of high-precision arithmetic onto MATLAB

長谷川秀彦† 椎葉 健‡ 石渡 恵美子‡
Hidehiko Hasegawa Takeru Shiiba Emiko Ishiwata

1. はじめに

数値計算アルゴリズムには丸め誤差に敏感なものも少なくない。しかし、演算精度を改善できれば、より効果的に利用できることも多い。演算精度の改善には数式処理システム（ほぼ無限桁だが、対象が有理数）、GMP などの可変多倍長演算（ほぼ無限桁）、Double-double 演算を使用した４倍精度演算[1]などが利用できるが、一般の倍精度演算に比べれば大きな演算コストがかかる。最近の著しいコンピュータ性能向上により、倍精度、演算の組合せで４倍精度演算を実行する Double-double 演算ならば、合理的な演算時間で実行できる（倍精度演算を基準にした演算量は、加減算は 11 倍、乗算は 24 倍）[2]。

GMP を利用するにせよ、Double-double 演算を利用するにせよ、高精度演算を利用するには演算型を明示したプログラムを書く必要がある。まったく同じアルゴリズムであっても、字面から見れば倍精度用のプログラムと４倍精度用のプログラムはまったく別物であり、ここにバグが入り込む危険性がある。アルゴリズムをテストする観点からは、プログラムがほぼそのままで異なる演算精度で使えることは利便性につながる。

2. オブジェクト指向

たとえば、Fortran の言語仕様は４倍精度のデータ型を持ち、何も意識せずに倍精度と４倍精度を混在させられるが、一般のコンパイラでは整数演算の組合せとして実装されているため、高速化は期待しにくい。最近のプログラミング言語であれば、オブジェクト指向として、新たなデータ型とそのデータ型に対する演算子を定義できるため、倍精度演算と４倍精度演算に対してもオブジェクト指向的なアプローチが可能である。実際、数値計算プログラムに現れる四則演算の式、たとえば

z = a*x +y

を、適当な関数を導入して

axpy(z, a, x, y)

と書くようにすれば演算精度によらずに使えるだろう。しかし、四則演算をこのように書くのは抵抗があるだろうし、可読性も大幅に低下する。残念ながら、C++、 Java などのオブジェクト指向言語でも、四則演算子＋、－、＊、／を用いた四則演算で、データ型によらず演算子を共用することは困難であり、四則演算の部分でオブジェクト指向のメリットを享受することは難しい。プログラミング言語を用いる方法では、コードチューニング、コンパイラによる最適化、並列化・ベクトル化など、実プログラムに対する性能面のメリットは大きいが[3]、アルゴリズムのテストという観点ではハードルが高い。そこで、オブジェクト指向の機能を持つ汎用的な数値処理環境上に Double-double 演算の実装を試みる。

```
function [x, e] = dd_cg(a, b, x0)
    [m, n] = size(a);
    x = DD(zeros(m,1));
    r = b - a * x0;  p = r;
    rn = norm(r, 2);  rr0 = r' * r;
    nMax = max(m, n); i = 1;
    e = DD(zeros(nMax,1));
    while(i < nMax)
        y = a * p;
        alpha = rr0 / (p' * y);
        x = x + alpha * p;
        r = r - alpha * y;
        e(i) = norm(r,2) / rn;
        if(e(i) < 1.0E-15)
            break;
        end
        rr1 = r' * r;
        beta = rr1 / rr0;
        p = r + beta * p;
        rr0 = rr1; i = i + 1;
    end
end
```

図 1　Matlab 版 CG 法コード

2.1 Matlab と Scilab

Matlab は Mathworks が提供する広く一般に使われている行列ベースの汎用的な数値処理環境である。Scilab は INRIA が開発したフリーの汎用的な数値処理環境であり、Matlab とほぼ同等の記法と機能を有する。Matlab、Scilab とも、Mathematica や Maple のような数式処理システムと同様に、それぞれの環境を起動し、その中で行列ベースの記法により処理を記述する（Fortran や C++をより簡単に書いているイメージ、図 1 に CG 法のサンプルコードを示す）。Matlab と Scilab はオブジェクト指向の機能を備え、新しいデータ型の定義、新しい演算の定義ができ、四則演算子＋、－、＊、／についても再定義ができる。すなわち、４倍精度のデータ型を定義し、４倍精度のデータ型に対する＋（加算）の場合は、Double-double 演算の加算が実行されるようなシステム構築ができる。外部関数の呼び出しも可能なので、Matlab や Scilab の命令を組み合わせて Double-

† 筑波大学図書館情報メディア系
‡ 東京理科大学 理学部 数理情報科学科（現、応用数学科）

double 演算を実現してもよいし、Double-double 演算を実行する外部関数（実行形式）を用意してもよい。

2.1.1 Scilab での実装

われわれは、Scilab 向けに Double-double 演算を用いた４倍精度演算と、Quad-double 演算を用いた８倍精度を実装した[2]。行列演算に関する基本機能、疎行列（行列の非ゼロ要素だけ保持）、C 言語による外部関数などを実装し、Windows と Linux 上で性能評価を行った。

設計どおり、型宣言以外の変更なしに、異なる演算精度を組み合わせてアルゴリズムの確認ができるようにできたが、Scilab がインタプリタ形式の実行であるため性能面での問題が残った。すべてを外部関数に任せるようにして、外部関数を高速化すれば問題を解消できる可能性があるが、システムメンテナンスの労力が大きくなる。

2.1.2 Matlab での実装

Matlab においても Scilab と同様の実装が可能であることと、Matlab では Matlab の関数をコンパイルして実行する機能があるため、Scilab と Matlab での性能面の違いを検証する。そのために、まずは Matlab 上に Double-double 演算を実行できる環境を構築する。

Double-double では２つの浮動小数点数を用いてデータを扱うので、２つの数値を持つ型として DD を定義する。Matlab の浮動小数点数は数値型の値なので、DD 型では２つの数値型の値をもつ型として定義し、演算子をオーバーロードし、数値型の値と同様に演算が行えるように定義する（表１参照）。

表１　演算の定義

算術演算	関数	説明
a + b	plus(a, b)	加算
a – b	minus(a, b)	減算
-a	uminus(a)	単項マイナス
+a	uplus(a)	単項プラス
a .* b	times(a, b)	要素単位の乗算
a * b	mtimes(a, b)	行列乗算
a ./ b	rdivide(a, b)	要素単位の右除算
a .\ b	ldivide(a, b)	要素単位の左除算
a / b	mrdivide(a, b)	行列の右除算
a \ b	mldivide(a, b)	行列の左除算

たとえば、+（加算）の場合、Matlab では

```
a+b
```

というコードに対して、内部的に

```
plus( a, b )
```

という関数が呼ばれるようになっているので、a, b の型に合った演算が動作するようしてやればよい。全体的な方針として、まず引数の型が double 型か DD 型かを判別し、double 型と DD 型との演算、DD 型と double 型との演算、DD 型と DD 型との演算についての処理を定義している。Double-double 演算の核となる部分、四則演算子は Calc クラスで定義した「厳密な演算」を実現する関数を組み合わせて実装した。Matlab の場合、データ型の基本は行列（要素が１個ならスカラー、１次元ならベクトル）なので、a, b は行列と仮定し、行列の要素分だけ演算を繰り返す。関係演算子、論理演算子、行列に対する関数、行列を定義する関数などに対して、DD 型が使えるように拡張を行う。

この結果、CG 法のコードは図１になる。x, e, a, b, x0 の型宣言に関係した部分だけがデータの型に依存した部分となる。それ以外の部分は、型によらず、倍精度と４倍精度まったく同じコードとなる。

3. Matlab での高速化

DD 型の演算を高速化するには、DD 型で繰り返して実行される Calc クラスで定義した「厳密な演算」を高速化する必要がある。そこで、この部分を C 言語で記述し、Matlab の外部関数呼び出しを用いて実装した。Matlab の外部関数内では、C 言語の動的メモリの割り当てに用いる new や malloc を使用せず、mex ファイルにある動的メモリ割り当て関数 mxCreateDoubleMatix や mxMalloc などを使用し、mxDestroyArray や mxFree などで動的メモリの解放を行う。Matlab の場合、Scilab と異なり、特別な C コンパイラを用意することなくコンパイルが可能である。現時点では、大きな性能向上にはなっていない。

原因は、外部関数の呼び出しコストは少なくないためと考えられる。1 回の呼び出しで、（大きな）行列・ベクトル演算の場合に十分な高速化効果が得られる可能性がある。Matlab の基本データ構造は行列であるため、要素の数に応じた呼び出しの実装はしやすいが、高速化のためには１回の呼び出しでより大きな単位の処理ができるような設計が必要になる。高速化のために、処理量（行列サイズと演算精度、必要ならば格納形式）に応じた制御を導入すると、システムメンテナンスの労力が大きくなる。

4. おわりに

Scilab や Matlab のような汎用的な数値処理環境を用いて、複数の演算精度に対して、同じ変数名、四則演算子＋、－、＊、／が使える高精度演算環境を構築した。これによって、利用者の利便性は各段に向上し、コードの変更に対するバグが入りにくくなり、コードの可読性も悪化しない。

一方、高速化には外部関数を用いる必然性が生じるが、一般に外部関数の呼び出しコストは小さくないため、外部関数の処理量を増やすための工夫が必要となって、システム内のコードが複雑化する。

このトレードオフをどうするかが大きな問題である。

謝辞

本研究の実施には、JSPS 科研費 JP25330141 の助成を受けた。

参考文献

[1] D.H. Bailey, QD (C++ /Fortran-90 double-double and quad-double package), Available at http://crd.lbl.gov/~dhbailey/mpdist/

[2] 吉川 慧子, 斉藤 翼, 石渡 恵美子, 長谷川 秀彦, “Scilab における高精度演算環境 MuPAT の実装”, 図書館情報メディア研究, Vol. 11, No. 1, pp. 23-46 (2013).

[3] 菱沼 利彰, 藤井 昭宏, 田中 輝雄, 長谷川 秀彦, “AVX2 を用いた倍精度 BCRS 形式疎行列と倍々精度ベクトル積の高速化”, 情報処理学会論文誌コンピューティングシステム(ACS), Vol.7, No.4, pp.25-33 (2014).