数理解析研究所講究録 1381 巻 2004 年 128-133

高精度内積計算アルゴリズムとその応用

大石 進一[1]
早稲田大学理工学部

1 はじめに

文献 [1] では、構成的実数は、それを計算するプログラムと同一視でき、実数と同一視できるプログラムの四則演算が定義できることを述べた。このように、計算機上でも、構成的実数体上の四則演算が厳密に実行できることはよく知られている。

応用上からは、このような厳密な四則演算を、いかに効率的に実装するかが問題になる。通常、計算機には浮動小数点演算が実装され、倍精度浮動小数点数などがハードウエア的にはサポートされている。したがって、整数配列で整数を定義し、有理数演算を基に構成的実数を構築するような実装では、浮動小数点演算の高速性に対してまったくといってよいほど実行速度が遅くなり、多くの現実的な応用には役に立たなくなる。

そこで、浮動小数点演算を用いて厳密な四則演算ができないかということが考えられる。結果からいうと、ある意味でそれは可能であり、特に加減算と乗算については大変効率的な実装ができる。

加算と乗算を組み合わせると、内積計算ができるが、浮動小数点演算により、効率的に必要な精度での内積計算をする方式を著者と S.M.Rump(ハンブルグ工科大学)、荻田武史（早稲田大学 21COE 客員講師）の共同研究で開発した。この方式によると、任意に高い条件数をもつベクトル間の内積計算が IEEE754 規格に従う浮動小数点演算のみで計算できる。これを利用すると、数値線形代数の問題の多くが、どんなに条件数が大きくても数値計算で精度保証付に解け、計算効率も大変高く保てることがわかる。

構成的実数の理論家から見ると、単なる実装の話であるが、応用の現場から見ると、今まで計算機で解けなかった多くの数値線形代数の問題が効率的にかつ数学的に厳密な意味で解けるようになることになる。

構成的実数論に計算速度や実装容易性などの新しい概念を持ち込んで、応用分野の具体的な問題との関わりを論じる方向性が見えるのではないであろうかと考えている。その意味でこの研究集会でお話をさせていただいた。

以下、具体的例題を扱う。すなわち、本文では、連立一次方程式

$$Ax = b \tag{1}$$

の数値解の精度保証付き数値計算法について考える。ただし、$n \times n$ 行列 A は実行列で、b は実 n 次元ベクトルとする。ここに、行列 A が実とは、その要素が全て実数であることをいう。

本論文では、IEEE754 規格を満たす倍精度浮動小数点数上で高精度に内積計算を行うアルゴリズムを示し、A が密行列でガウスの消去法などの直接解法が適用できような問題のクラスに対しては、式 (1) の残差反復計算が高速に実行できることを述べるとともに、この高精度内積計算法を利用して、式 (1) の数値解の高速で高精度な精度保証法を示す。本論文の議論のポイントは、このような高精度な精度保証法が、丸めのモード制御方式精度保証付き数値計算法の高速性を損なうことなく実装できることを示すことである。

[1]早稲田大学理工学部 〒 169-8555 東京都新宿区大久保 3-4-1

2 高精度内積計算アルゴリズム

2.1 高精度内積計算アルゴリズム

1950年代のRung-Kutta法に対するGillの丸めの誤差対策法の技法に関する議論から出発して、浮動小数点数の加算の誤差に対する深い議論がされるようになった。その大きな成果の一つが次の1969年のKnuthの定理である。

定理 1 (Knuth[3]) $a, b \in \mathbb{F}$ とする。$x \in \mathbb{F}$ と $y \in \mathbb{F}$ を最近点への丸めのモード下で、つぎのアルゴリズム **TwoSum** によって計算すると $a + b = x + y$ が成立する。

function $[x, y] = \mathbf{TwoSum}(a, b)$
$x = a \oplus b;$
$b_V = x \ominus a;$
$a_V = x \ominus b_V;$
$b_R = b \ominus b_V;$
$a_R = a \ominus a_V;$
$y = a_R \oplus b_R;$

(定理終)

1971年Dekkerはこれと同様な定理が浮動小数点数の乗算についても成立することを示した。まず、準備として、次の定理を示す。

定理 2 (Dekker[4]) $a \in \mathbb{F}$ とする。つぎのアルゴリズム **Split** によって仮数部の非ゼロ要素の数がそれぞれ 26-*bit* である a_H と a_L を計算すると、$a = a_H + a_L$ で $|a_H| \geq |a_L|$ となる。

function $[a_H, a_L] = \mathbf{Split}(a)$
$c = (2^{27} + 1) \otimes a;$
$d = c \ominus a;$
$a_H = c \ominus d;$
$a_L = a \ominus a_H;$

(定理終)

この定理を基に、次の定理が成り立つ。

定理 3 (Dekker[4]) $a, b \in \mathbb{F}$ とする。$x \in \mathbb{F}$ と $y \in \mathbb{F}$ をつぎのアルゴリズム **TwoProduct** で計算すると、計算の途中でアンダーフローが起きなければ $a * b = x + y$ が成り立つ。

function $[x, y] = \mathbf{TwoProduct}(a, b)$
$x = a \otimes b;$
$[a_H, a_L] = \mathbf{Split}(a);$
$[b_H, b_L] = \mathbf{Split}(b);$
$err_1 = x \ominus a_H \otimes b_H;$
$err_2 = err_1 \ominus a_L \otimes b_H;$
$err_3 = err_2 \ominus a_H \otimes b_L;$
$y = a_L \otimes b_L \ominus err_3;$

(定理終)

以上の定理を基に、$\alpha \in \mathbb{F}$ と $p = (p_1, p_2, \cdots, p_n)^T \in \mathbb{F}^n$ に対して、$\beta = \alpha \oplus p_1 \oplus p_2 \oplus \cdots \oplus p_n$ と β の計算誤差を表すベクトル $q = (q_1, q_2, \cdots, q_n)^T$ を計算するアルゴリズム **SumEFT** を導入する。

アルゴリズム 1 [SumEFT]

function $[\beta, q] = \mathbf{SumEFT}(\alpha, p)$

$\beta_0 = \alpha$;

for $= 1 : n$

$[\beta_i, q_i] = \mathbf{TwoSum}(\beta_{i-1}, p_i)$;

(アルゴリズム終)

定理 1 より明らかに $\alpha \in \mathbb{F}$ と $p \in \mathbb{F}^n$ を入力として、**TwoSum** で β と q を計算したとすると

$$\alpha + \sum_{i=1}^{n} p_i = \beta + \sum_{i=1}^{n} q_i \tag{2}$$

が成り立つ。

さらに、つぎの内積の高精度計算アルゴリズム $\mathbf{Dot}(x, y)$ を導入する。

function $s = \mathbf{Dot}(x, y)$
$p_0 = s_0 = 0$;
for $i = 1 : n$
$[h_i, r_i] = \mathbf{TwoProduct}(x_i, y_i)$;
$[p_i, q_i] = \mathbf{TwoSum}(p_{i-1}, h_i)$;
$s_i = s_{i-1} \oplus (q_i \oplus r_i)$;
$s = p_n \oplus s_n$;

2.2 残差反復法

連立一次方程式 (1) を考える。ただし、$A \in \mathbb{F}^{n \times n}$ で $x, b \in \mathbb{F}^n$ とする。$\tilde{x}$ を式 (1) の数値解とする。$A_e \in \mathbb{F}^{(n+1) \times n}$ と $\tilde{x}_e \in \mathbb{F}$ を次のように定義する。

$$A_e = (A|b),\ x_e = \begin{pmatrix} \tilde{x} \\ -1 \end{pmatrix} \tag{3}$$

ただし、$-1 \in \mathbb{F}$ である。このとき

$$A_e \tilde{x}_e = A\tilde{x} - b \tag{4}$$

が成立する。

$$\begin{aligned} s = \quad & (\mathbf{Dot}(A_e^{(1)}, \tilde{x}_e), \mathbf{Dot}(A_e^{(2)}, \tilde{x}_e), \cdots, \\ & \mathbf{Dot}(A_e^{(n)}, \tilde{x}_e))^T \end{aligned} \tag{5}$$

とする。ただし、$A_e^{(i)}$ は行列 A_e の第 i 行目とする。式 (5) の s は残差 $r = A\tilde{x} - b$ を内積の高精度計算アルゴリズム **Dot** によって計算したものとなる。z を $Au = s$ の倍精度浮動小数点数演算で計算した u の近似解として

$$x_{new} = \tilde{x} \ominus z \tag{6}$$

で、残差反復した、式 (1) の新しい近似解 x_{new} が得られる。これが本論文での残差反復法の実装法の提案であり、多倍長浮動小数点数パッケージを利用しないで内積の高精度計算アルゴリズム **Dot** を用いているため、IEEE754 の倍精度浮動小数点数規格に従う計算システムであれば、ポータブルに実装できることがわかる。

3 高精度精度保証法

2002年、Oishi と Rump[2] は式 (1) の数値解に対する高速精度保証を提案した。ここでは、この精度保証法を内積の高精度計算アルゴリズム **Dot** を用いて高速かつ高精度に実装する方法を示す。つぎの定理が基礎となる。

定理 4 (Banach) A を $n \times n$ 実行列、b を n 次元実ベクトルとして連立一次方程式

$$Ax = b \tag{7}$$

を考える。適当な $n \times n$ 実行列 R が存在して[2]

$$\|RA - I\|_\infty < 1 \tag{8}$$

を満たすとき、A は可逆で、任意の n 次元実ベクトル $\tilde{x}$ に対して[3]

$$\|x^* - \tilde{x}\|_\infty \leqq \frac{\|R(A\tilde{x} - b)\|_\infty}{1 - \|RA - I\|_\infty} \tag{9}$$

となる。ただし、I は n 次元単位行列で、$x^* = A^{-1}b$ とする。*(定理終)*

この定理を基に、連立一次方程式の精度保証法を実装法を示す。この定理では $\tilde{x}$ は任意であるが、我々の実装法においては、2.2 で提案した高精度内積を用いた残差反復法により必要な回数だけ反復した式 (1) の近似解を取るものとする。また、式 (9) の右辺の分子に現れる $A\tilde{x} - b$ を前節で提案した内積の高精度計算アルゴリズム **Dot** を用いて計算することを提案する。したがって、IEEE754 の倍精度浮動小数点数規格に従う計算システムであれば、ポータブルに実装できることがわかる。

4 数値例

ここでは、提案したアルゴリズムの有効性を示すために行った数値実験の中から幾つかを紹介する。行列 A の条件数は $O(10^{14})$ に比較して小さいが問題の次元が $n = 1000$ と前節の問題に比して大きな問題を例として取り上げる。すなわち、A は 1000×1000 実行列でその要素は乱数であるとする。

```
1> b=A*ones(n,1);R=inv(A);
2> norm(R)*norm(A)
ans=
    1.156215743354834267e+05
```

より、A の条件数は $O(10^5)$ である。1>行目からわかるように $b = A(1, 1, \cdots, 1)^T$ として式 (1) を考える。ただし、丸め誤差の存在により、この場合の式 (1) の厳密解は $x = (1, 1, \cdots, 1)^T$ と若干異なることを注意する。

```
3> x=A\b;x[999]
ans=
    9.999999999989218624e-01
```

[2]実際の応用では R としては A の近似逆行列を取る。
[3]やはり実際の応用では $Ax = b$ の近似解を $\tilde{x}$ とする。

```
4> e=vale(A,b,x,R)
ans=
    2.673299304758226751e-12
5> x=ite(A,b,x,R);x[999]
ans=
    1.000000000000095923e+00
6> e=vale(A,b,x,R)
ans=
    1.108664021230798898e-16
```

この場合には一回の残差反復でほぼ最終桁まで正しい結果が得られていることがわかる。また、本論文で提案している高精度精度保証プログラムがシャープな誤差評価を与えていることがわかる。この点に関し、残差を高精度に計算しない従来の精度保証プログラムで精度を計算すると

```
7> e2=f(A,b,x,R)
ans=
    1.679971748267317365e-10
```

となって、本来、近似解が持っている精度をシャープに評価できていないことがわかる。

ここで、式 (1) の近似解を求める時間 (A の LU 分解にかかる時間) と残差を高精度に計算しない従来の精度保証プログラムで精度保証する時間 (本論文ではこれを標準精度保証法の計算時間という) と残差を高精度に計算する本論文で提案した方法 (新法の計算時間と呼ぶ) を表 1 に示す。表中に示した数字は各次元とも、異なった 10 のランダム行列 A を生成し、$b = A(1, 1, \cdots, 1)^T$ として、式 (1) を解いた時間を平均したものである。尚、精度保証の時間には A の近似逆行列 R を計算する時間を含めていない。計算に用いたコンピュータは Mac Power Book G4 (1.25GHz CPU) である。

表 1 高精度の精度保証に要する時間

n	LU 分解	標準法	新精度保証法
200	0.02	0.04	0.04
400	0.07	0.31	0.35
600	0.21	0.87	1.01
800	0.45	2.23	2.37
1000	0.81	3.84	4.31

近似解 (LU 分解) を計算する計算のため浮動小数点数演算の回数 (flops という単位が標準で用いられる。floating point operations の略である。) は四則演算をすべて一回と数えると $2/3n^3$flops である。一方、標準精度保証法も新精度保証法も近似逆行列 R を計算する時間は含めていないので、行列の積を 2 回計算する手間の $4n^3$flops である。よって、浮動小数点演算の回数の比でいえば、精度保証にかかる計算時間は標準法も新しい方法もともに LU 分解法の計算時間の 6 倍となる。4 に示した実際の数値計算例においてはこの比は 2 つの精度保証法とも 6 倍より少なくなって

いる。これは行列の積の計算がATLASによって生成された最適化BLASを基に行われていることに起因する。すなわち、最適化BLASはキャッシュヒット率を最大にするように行列の積を計算するように生成されている。ガウスの消去法の中にもこの最適化は生かされているが、行列の積の方が単純なアルゴリズムで計算できるので、この最適化の効果がより生かされる。したがって、精度保証に必要な主計算が行列の積であることから、実際の計算時間が浮動小数点演算の回数の比よりは短縮されているのである。

また、標準精度保証法の計算時間と新しい精度保証法の計算時間は、後者が高精度な内積計算を残差の計算に用いてるので、よりかかっているが、ほぼ同じオーダーであることが確認された。すなわち、残差の計算において高精度な内積計算を用いてもその演算回数は$O(n^2)$であり、しかも高精度な内積計算法に要する時間が浮動小数点数演算での内積計算に要する時間の一定の倍率でしか時間がかからないことからこれは理論的にも理解される。すなわち、新しい精度保証法は標準精度保証法の高速性を損なうことなく、高精度性が達成されていることがわかる。

参考文献

[1] 大石進一:精度保証付き数値計算, コロナ社 (2000)

[2] Shin'ichi Oishi and Siegfried M. Rump:"Fast verification of solutions of matrix equations", *Numer. Math.* vol. **90** (2002) pp.755-773

[3] D. E. Knuth:"*T*he Art of Computer Programming, Volume 2: Seminumerical Algorithms"., Reading, Massachusetts: Addison-Wesley (1969).

[4] T. J. Dekker:"A Floating-point Technique for Extending the Available Precision", *N*umer. Math. **18** (1971), 224–242.

[5] T. Ogita, S. Oishi and Y. Ushiro:"Computation of Sharp Rigorous Componentwise Error Bounds for the Approximate Solutions of System of Linear Equations", *R*eliable Computing, **9** (2003), 229–239.