数理解析研究所講究録 1320 巻 2003 年 151-161

Strassenのアルゴリズムによる行列乗算の高速精度保証

早稲田大学 教育学部　荻田 武史 (Takeshi Ogita)
School of Education, Waseda University
早稲田大学 理工学部　大石 進一 (Shin'ichi Oishi)
School of Science and Engineering, Waseda University
日立製作所 エンタープライズサーバ事業部　後 保範 (Yasunori Ushiro)
Enterprise Server Division, Hitachi Ltd.

1 はじめに

本論文では、行列乗算

$$C = AB \tag{1}$$

の高速な包み込み方法について考える。ここで、A は $m \times l$、B は $l \times n$ の一般行列である。行列乗算の包み込みとは、A, B が実行列のとき、

$$\underline{C} \leq AB \leq \overline{C} \quad \Longleftrightarrow \quad \text{すべての } (i,j) \text{ に対し } \quad \underline{C}_{i,j} \leq (AB)_{i,j} \leq \overline{C}_{i,j} \tag{2}$$

となるような行列 $\underline{C}, \overline{C}$ を求めることである。ここで、$G = [\underline{C}, \overline{C}]$ を区間行列 (interval matrix) と呼び、それに対し、通常の行列は点行列 (point matrix) と呼ぶ。近年、IEEE 標準 754 に従う浮動小数点演算の丸めモード制御を利用して、A と B の乗算の厳密な包み込みを高速に求める方法が、Oishi-Rump [3, 4] によって提案された。この方法を、丸めモード制御演算方式 (rounding mode controlled computation) と呼ぶ。この高速な方法を使うと、A と B の乗算の厳密な包み込みを、通常の浮動小数点演算による近似計算の手間の 2 倍で計算できる。行列乗算の包み込みは、精度保証付き数値計算において重要な役割を果たすので、その高速なアルゴリズムを開発することは有用である。

1969 年に、Strassen は分割統治法に基づく高速な行列乗算の方法 [6] を提案した。A, B が共に $n \times n$ 行列のとき、A と B の乗算に必要な計算量は、通常の方法では $O(n^3)$ flops であるのに対し、Strassen の方法によると理論的には、$O(n^{\log_2 7}) \approx O(n^{2.807})$ flops となる。

本論文の主要な点は、Strassen の方法に基づき、丸めモード制御演算方式を利用して行列乗算の包み込みをする高速な方法を提案することである。そのために、我々は、区間行列における行列乗算の高速なアルゴリズム [2] を使用する。また、複素行列の場合についても言及し、複素行列における行列乗算の高速な包み込み方法を提案する。

本論文の主要な結果は、Strassen の方法による行列乗算の近似計算の約 2 倍の計算コストで行列乗算の包み込みができることを示す。

2 Strassenのアルゴリズム

A と B を $n \times n$ 実行列とする。以下のような 2×2 ブロックの行列乗算を考える。ここで、$n = 2m$ と仮定すると、各ブロックは $m \times m$ 行列である。

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}. \tag{3}$$

2.1 通常の方法

最初に、次のような 2×2 ブロックの通常の行列乗算を考える。

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}.$$

通常の方法では、$m \times m$ のブロック行列同士の 8 回の乗算と 4 回の加算がある。1 回のブロック行列同士の乗算と加算に必要な計算量はそれぞれ $2m^3 - m^2$ flops と m^2 flops であるから、この通常の方法の計算量は、トータルで $16m^3 - 4m^2$ flops となる。

2.2 Strassenの方法

次に、Strassen の方法による行列乗算を考える。

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ P_2 + P_4 & P_1 - P_2 + P_3 + P_6 \end{bmatrix}.$$

但し、

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), & P_2 &= (A_{21} + A_{22})B_{11}, \\ P_3 &= A_{11}(B_{12} - B_{22}), & P_4 &= A_{22}(B_{21} - B_{11}), \\ P_5 &= (A_{11} + A_{12})B_{22}, & P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}). \end{aligned}$$

Strassen の方法では、$m \times m$ のブロック行列同士の 7 回の乗算と 18 回の加算がある。したがって、この方法の計算量は、トータルで $14m^3 + 11m^2$ flops となるので、ブロックサイズ m が十分に大きければ、Strassen の方法による行列乗算の計算量は通常の方法に比べて 7/8 になる。さらに、Strassen の方法は recursive である。

Strassen の方法による行列乗算のアルゴリズムは以下のようである。本論文では、アルゴリズムを MATLAB ライクに表現する。

アルゴリズム 1 ([1, 6]) *Strassen* の方法による行列乗算：

function $\widetilde{C} = \mathbf{StrassMM}(A, B)$
$[m, l] = \mathbf{size}(A)$;

$n = \mathbf{size}(B, 2);$
if $(l * m * n) < N_{\min}$
  $\widetilde{C} = A * B;$
else
  $T_1 = A_{11} + A_{22};\ T_2 = B_{11} + B_{22};\ T_3 = A_{21} + A_{22};\ T_4 = B_{12} - B_{22};$
  $T_5 = B_{21} - B_{11};\ T_6 = A_{11} + A_{12};\ T_7 = A_{21} - A_{11};\ T_8 = B_{11} + B_{12};$
  $T_9 = A_{12} - A_{22};\ T_{10} = B_{21} + B_{22};$
  $P_1 = \mathbf{StrassMM}(T_1, T_2);\ P_2 = \mathbf{StrassMM}(T_3, B_{11});$
  $P_3 = \mathbf{StrassMM}(A_{11}, T_4);\ P_4 = \mathbf{StrassMM}(A_{22}, T_5);$
  $P_5 = \mathbf{StrassMM}(T_6, B_{22});\ P_6 = \mathbf{StrassMM}(T_7, T_8);$
  $P_7 = \mathbf{StrassMM}(T_9, T_{10});$
  $\widetilde{C}_{11} = P_1 + P_4 - P_5 + P_7;$
  $\widetilde{C}_{12} = P_3 + P_5;$
  $\widetilde{C}_{21} = P_2 + P_4;$
  $\widetilde{C}_{22} = P_1 - P_2 + P_3 + P_6;$
end

3 基本アルゴリズム

本章では、区間行列における行列乗算の包み込みを計算する方法を概説する。これらの基本的なアイディアとアルゴリズムは大石 [8] などによって既に提案されている。

最初に、行列 A, B の乗算の包み込みを計算するアルゴリズムを以下に示す。

アルゴリズム 2 ([8]) 通常の行列乗算の包み込み：

function $[\underline{C}, \overline{C}] = \mathbf{RealMMIn}(A, B)$
setround(*down*);
$\underline{C} = A * B;$
setround(*up*);
$\overline{C} = A * B;$

ここで、命令 **setround**(*down*) は丸めモードを下への丸めに変更し、**setround**(*up*) は丸めモードを上への丸めに変更する。丸めモードが一度変更された場合、次に丸めモードを変更する **setround** 命令が現れるまで変更された丸めモードが持続する。丸めモード制御演算方式の詳細については、文献 [3, 4, 8] を参照されたい。

次に、区間行列 $[\underline{A}, \overline{A}]$ の中心と半径を計算するアルゴリズムを示す。ここで、$m \times n$ 行列 $X = (x_{i,j}), Y = (y_{i,j})$ に対し $X \leq Y$ は

$$X \leq Y \Longleftrightarrow \text{すべての } (i,j) \text{ に対して } x_{i,j} \leq y_{i,j}$$

を意味し、$m \times n$ 区間行列は

$$[\underline{A}, \overline{A}] = \{X : m \times n \text{ 行列} \mid \underline{A} \leq X \leq \overline{A}\}$$

で定義される。

アルゴリズム 3 ([8]) 区間行列 $[\underline{A}, \overline{A}]$ に対して $[M - R, M + R] \supseteq [\underline{A}, \overline{A}]$ となるような中心 M と半径 R の計算:

function $[M, R] = \mathbf{CenterRadius}(\underline{A}, \overline{A})$
setround(up);
$M = \underline{A} + 0.5 * (\overline{A} - \underline{A})$;
$R = M - \underline{A}$;

以下に、点行列と区間行列の乗算を包み込む方法および区間行列と点行列の乗算を包み込む方法をそれぞれ示す。

アルゴリズム 4 ([8]) 点行列と区間行列の乗算：

function $[\underline{P}, \overline{P}] = \mathbf{PIMult}(A, \underline{B}, \overline{B})$
$[B_M, B_R] = \mathbf{CenterRadius}(\underline{B}, \overline{B})$;
$[\underline{T}, \overline{T}] = \mathbf{RealMMIn}(A, B_M)$;
setround(up);
$P_M = \underline{T} + 0.5 * (\overline{T} - \underline{T})$;
$P_R = (P_M - \underline{T}) + \mathbf{abs}(A) * B_R$;
$\overline{P} = P_M + P_R$;
setround($down$);
$\underline{P} = P_M - P_R$;

ここで、実行列 $X = (x_{ij})$ に対し、命令 $\mathbf{abs}(X)$ は $|X| = (|x_{ij}|)$ を意味する。アルゴリズム 4 では 3 回の行列乗算を必要とする。すなわち、$\mathbf{RealMMIn}(A, B_M)$ で 2 回の行列乗算と $\mathbf{abs}(A) * B_R$ で 1 回の行列乗算が必要である。

アルゴリズム 5 ([8]) 区間行列と点行列の乗算：

function $[\underline{P}, \overline{P}] = \mathbf{IPMult}(\underline{A}, \overline{A}, B)$
$[A_M, A_R] = \mathbf{CenterRadius}(\underline{A}, \overline{A})$;
$[\underline{T}, \overline{T}] = \mathbf{RealMMIn}(A_M, B)$;
setround(up);
$P_M = \underline{T} + 0.5 * (\overline{T} - \underline{T})$;
$P_R = (P_M - \underline{T}) + A_R * \mathbf{abs}(B)$;
$\overline{P} = P_M + P_R$;
setround($down$);
$\underline{P} = P_M - P_R$;

アルゴリズム 5 もアルゴリズム 4 と同様に 3 回の行列乗算を必要とする。

さらに、区間行列同士の乗算を包み込むアルゴリズムを以下に示す。

アルゴリズム 6 ([8]) 区間行列同士の乗算:

function $[\underline{P}, \overline{P}] = \mathbf{IIMult}(\underline{A}, \overline{A}, \underline{B}, \overline{B})$
$[A_M, A_R] = \mathbf{CenterRadius}(\underline{A}, \overline{A})$;
$[B_M, B_R] = \mathbf{CenterRadius}(\underline{B}, \overline{B})$;
$[\underline{T}, \overline{T}] = \mathbf{RealMMIn}(A_M, B_M)$;
setround(up);
$P_M = \underline{T} + 0.5 * (\overline{T} - \underline{T})$;
$\overline{S} = \mathbf{abs}(B_M) + B_R$;
$P_R = (P_M - \underline{T}) + A_R * \overline{S} + \mathbf{abs}(A_M) * B_R$;
$\overline{P} = P_M + P_R$;

setround($down$);
$\underline{P} = P_M - P_R$;

ここで、アルゴリズム 6 では 4 回の行列乗算を必要とする。

Strassen の方法に、丸めモード制御演算方式による従来の行列乗算の包み込み方法をそのまま適用しようとすると問題が発生する。例えば、P_1 の計算

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

を考えると、丸めモード制御演算によって $A_{11} + A_{22}$ と $B_{11} + B_{22}$ の計算結果は区間行列で包み込むことができるが、そのため $(A_{11} + A_{22})(B_{11} + B_{22})$ は、区間行列同士の乗算となる。従来の方式では、区間行列同士の乗算の包み込みに必要な計算量は、点行列同士の乗算の近似計算に必要な計算量の約 4 倍（$8m^3 + O(m^2)$ flops）であり、同様に、区間行列と点行列の乗算あるいは点行列と区間行列の乗算の包み込みに必要な計算量は、点行列同士の乗算の近似計算に必要な計算量の約 3 倍（$6m^3 + O(m^2)$ flops）である。したがって、Strassen の方法を用いると、従来の方式では、行列乗算 AB の包み込みに必要な計算量は $48m^3 + O(m^2) \approx 6n^3$ flops となる。通常の方法では、$4n^3$ flops で行列乗算の包み込みが計算できるので、Strassen の方法を用いると逆に計算量が多くなってしまう。

そこで、次の章では、区間行列と点行列の乗算の包み込み、点行列と区間行列の乗算の包み込み、そして区間行列同士の乗算の包み込みを高速に計算する新しい方法を提案する。

4 高速アルゴリズム

本章では、Strassen の方法に基づいた行列乗算の包み込みを高速に計算する方法を提案する。

X を $m \times l$ 非負行列、Y を $l \times n$ 非負行列とすると、以下の不等式が成り立つ：

$$(XY)_{i,j} = \sum_{k=1}^{l} X_{i,k}Y_{k,j} \le \sum_{k=1}^{l} X_{i,k} \left(\max_{1 \le p \le n} Y_{k,p} \right) = P_{i,j} \tag{4}$$

あるいは

$$(XY)_{i,j} = \sum_{k=1}^{l} X_{i,k}Y_{k,j} \le \sum_{k=1}^{l} \left(\max_{1 \le p \le m} X_{p,k} \right) Y_{k,j} = Q_{i,j}. \tag{5}$$

式 (4), (5) から、

$$(XY)_{i,j} \le \min\{P_{i,j}, Q_{i,j}\}. \tag{6}$$

以下に、非負行列同士の乗算の上限を高速に求めるアルゴリズムを示す。

アルゴリズム 7 非負行列同士の乗算の上限：

function $\overline{G}$ = **FastMultNN**(X, Y)
$[m, l]$ = **size**(X);　　n = **size**(Y, 2);
P = **zeros**(m, n);　　Q = **zeros**(m, n);
u = **zeros**(1, l);　　v = **zeros**(l, 1);
for $k = 1 : l$,

$u(1,k) = \mathbf{max}(X(1:m,k));$
$v(k,1) = \mathbf{max}(Y(k,1:n));$
end
setround$(up);$
$\boldsymbol{p} = \boldsymbol{u} * Y;$
for $i = 1:m,$
$P(i,1:n) = \boldsymbol{p};$
end
$\boldsymbol{q} = X * \boldsymbol{v};$
for $j = 1:n,$
$Q(1:m,j) = \boldsymbol{q};$
end
$\overline{G} = \mathbf{min}(P,Q)$

このアルゴリズムでは、行列-ベクトル積とベクトル-行列積を1回ずつ計算するだけでよい。よって、非負行列同士の乗算の上限を計算するのに必要な計算量は、X, Y が $n \times n$ 行列のときは、$O(n^2)$ で済む。

アルゴリズム7を用いて、区間行列における行列乗算を包み込む高速な方法を以下に示す。

アルゴリズム 8 点行列と区間行列の高速乗算：

function $[\underline{P}, \overline{P}] = \mathbf{PIMultF}(A, \underline{B}, \overline{B})$
$[B_M, B_R] = \mathbf{CenterRadius}(\underline{B}, \overline{B});$
$[\underline{T}, \overline{T}] = \mathbf{RealMMIn}(A, B_M);$
setround$(up);$
$P_M = \underline{T} + 0.5 * (\overline{T} - \underline{T});$
$P_R = (P_M - \underline{T}) + \mathbf{FastMultNN}(\mathbf{abs}(A), B_R);$
$\overline{P} = P_M + P_R;$
setround$(down);$
$\underline{P} = P_M - P_R;$

アルゴリズム 9 区間行列と点行列の高速乗算：

function $[\underline{P}, \overline{P}] = \mathbf{IPMultF}(\underline{A}, \overline{A}, B)$
$[A_M, A_R] = \mathbf{CenterRadius}(\underline{A}, \overline{A});$
$[\underline{T}, \overline{T}] = \mathbf{RealMMIn}(A_M, B);$
setround$(up);$
$P_M = \underline{T} + 0.5 * (\overline{T} - \underline{T});$
$P_R = (P_M - \underline{T}) + \mathbf{FastMultNN}(A_R, \mathbf{abs}(B));$
$\overline{P} = P_M + P_R;$
setround$(down);$
$\underline{P} = P_M - P_R;$

アルゴリズム 10 区間行列同士の高速乗算：

function $[\underline{P}, \overline{P}] = \mathbf{IIMultF}(\underline{A}, \overline{A}, \underline{B}, \overline{B})$
$[A_M, A_R] = \mathbf{CenterRadius}(\underline{A}, \overline{A});$
$[B_M, B_R] = \mathbf{CenterRadius}(\underline{B}, \overline{B});$
$[\underline{T}, \overline{T}] = \mathbf{RealMMIn}(A_M, B_M);$
setround$(up);$
$P_M = \underline{T} + 0.5 * (\overline{T} - \underline{T});$

$\overline{S} = \mathbf{abs}(B_M) + B_R;$
$P_R = (P_M - \underline{T}) + \mathbf{FastMultNN}(A_R, \overline{S}) + \mathbf{FastMultNN}(\mathbf{abs}(A_M), B_R);$
$\overline{P} = P_M + P_R;$
$\mathbf{setround}(down);$
$\underline{P} = P_M - P_R;$

これら3つのアルゴリズムにそれぞれ必要な計算量は、点行列同士の乗算の近似計算を2回実行するのに必要な計算量とほとんど同じである。

以上の議論を用いて、Strassenの方法に基づく行列乗算を包み込む高速なアルゴリズムを以下に示す。

アルゴリズム 11 *Strassen*の方法による行列乗算の高速な包み込み：

function $[\underline{C}, \overline{C}] = \mathbf{StrassMMIn}(A, B)$
$[m, l] = \mathbf{size}(A);$
$n = \mathbf{size}(B, 2);$
if $(l * m * n) < N_{\min}$
  $[\underline{C}, \overline{C}] = \mathbf{RealMMIn}(A, B);$
else
  $\mathbf{setround}(down);$
  $\underline{T_1} = A_{11} + A_{22};\ \underline{T_2} = B_{11} + B_{22};\ \underline{T_3} = A_{21} + A_{22};\ \underline{T_4} = B_{12} - B_{22};$
  $\underline{T_5} = B_{21} - B_{11};\ \underline{T_6} = A_{11} + A_{12};\ \underline{T_7} = A_{21} - A_{11};\ \underline{T_8} = B_{11} + B_{12};$
  $\underline{T_9} = A_{12} - A_{22};\ \underline{T_{10}} = B_{21} + B_{22};$
  $\mathbf{setround}(up);$
  $\overline{T_1} = A_{11} + A_{22};\ \overline{T_2} = B_{11} + B_{22};\ \overline{T_3} = A_{21} + A_{22};\ \overline{T_4} = B_{12} - B_{22};$
  $\overline{T_5} = B_{21} - B_{11};\ \overline{T_6} = A_{11} + A_{12};\ \overline{T_7} = A_{21} - A_{11};\ \overline{T_8} = B_{11} + B_{12};$
  $\overline{T_9} = A_{12} - A_{22};\ \overline{T_{10}} = B_{21} + B_{22};$
  $[\underline{P_1}, \overline{P_1}] = \mathbf{IIMultF}(\underline{T_1}, \overline{T_1}, \underline{T_2}, \overline{T_2});\ [\underline{P_2}, \overline{P_2}] = \mathbf{IPMultF}(\underline{T_3}, \overline{T_3}, B_{11});$
  $[\underline{P_3}, \overline{P_3}] = \mathbf{PIMultF}(A_{11}, \underline{T_4}, \overline{T_4});\ [\underline{P_4}, \overline{P_4}] = \mathbf{PIMultF}(A_{22}, \underline{T_5}, \overline{T_5});$
  $[\underline{P_5}, \overline{P_5}] = \mathbf{IPMultF}(\underline{T_6}, \overline{T_6}, B_{22});\ [\underline{P_6}, \overline{P_6}] = \mathbf{IIMultF}(\underline{T_7}, \overline{T_7}, \underline{T_8}, \overline{T_8});$
  $[\underline{P_7}, \overline{P_7}] = \mathbf{IIMultF}(\underline{T_9}, \overline{T_9}, \underline{T_{10}}, \overline{T_{10}});$
  $\mathbf{setround}(down);$
  $\underline{C_{11}} = \underline{P_1} + \underline{P_4} - \overline{P_5} + \underline{P_7};$
  $\underline{C_{12}} = \underline{P_3} + \underline{P_5};$
  $\underline{C_{21}} = \underline{P_2} + \underline{P_4};$
  $\underline{C_{22}} = \underline{P_1} - \overline{P_2} + \underline{P_3} + \underline{P_6};$
  $\mathbf{setround}(up);$
  $\overline{C_{11}} = \overline{P_1} + \overline{P_4} - \underline{P_5} + \overline{P_7};$
  $\overline{C_{12}} = \overline{P_3} + \overline{P_5};$
  $\overline{C_{21}} = \overline{P_2} + \overline{P_4};$
  $\overline{C_{22}} = \overline{P_1} - \underline{P_2} + \overline{P_3} + \overline{P_6};$
end

もちろん、このアルゴリズムはアルゴリズム1と同様にrecursiveである。すなわち、アルゴリズム8, 9, 10においてそれぞれ **RealMMIn** を **StrassMMIn** に置き換えればよい。

アルゴリズム11は、14回のブロック行列同士の乗算と、ブロック行列同士の和や非負行列同士の高速乗算を含めたそれより低次のオーダの計算を必要とする。それゆえ、アルゴリズム11に必要な計算量は、行列サイズが十分に大きい場合、アルゴリズム1に必要な計算量の約2倍で済む。

5 複素行列における行列乗算の包み込み

以下のような複素行列の乗算を考える：

$$(P+iQ)=(A+iB)(C+iD). \tag{7}$$

ここで、A,B は共に $m\times l$ 実行列であり、C,D は共に $l\times n$ 実行列である。式 (7) から、

$$P = AC-BD, \tag{8}$$
$$Q = AD+BC. \tag{9}$$

ところが、$W=(A+B)(C+D)$ とすると、

$$P = AC-BD, \tag{10}$$
$$Q = W-AC-BD \tag{11}$$

が成り立つことが分かる ([1] p.15 などを参照)。それゆえ、複素行列の乗算は 3 回の実行列同士の乗算によって計算することができる。この事実に基づき、Strassen の方法を適用した複素行列における乗算の近似計算の高速なアルゴリズムを以下に示す。

アルゴリズム 12 *Strassen* の方法を用いた複素行列乗算*:*

function $\tilde{G}=$ **ComplexStrassMM**(Z_1,Z_2)
$A=$ **real**(Z_1); $B=$ **imag**(Z_1);
$C=$ **real**(Z_2); $D=$ **imag**(Z_2);
$T_1=A+B$;
$T_2=C+D$;
$W_1=$ **StrassMM**(T_1,T_2)
$W_2=$ **StrassMM**(A,C);
$W_3=$ **StrassMM**(B,D);
$P=W_2-W_3$;
$Q=W_1-W_2-W_3$;
$\tilde{G}=$ **complex**(P,Q);

ここで、**real**(Z) と **imag**(Z) は、それぞれ複素行列 Z の実部と虚部を表す命令である。さらに、命令 **complex**(P,Q) は、2 つの実行列 P,Q から $P+iQ$ となる複素行列を作成する。

アルゴリズム 11 を用いて、複素行列 Z_1,Z_2 の乗算 Z_1Z_2 の包み込みを計算する高速アルゴリズムを以下に示す。

アルゴリズム 13 *Strassen* の方法を用いた複素行列乗算の包み込み*:*

function $[\underline{G},\overline{G}]=$ **ComplexStrassMMIn**(Z_1,Z_2)
$A=$ **real**(Z_1); $B=$ **imag**(Z_1);
$C=$ **real**(Z_2); $D=$ **imag**(Z_2);
setround$(down)$;
$\underline{T_1}=A+B$; $\underline{T_2}=C+D$;
setround(up);
$\overline{T_1}=A+B$; $\overline{T_2}=C+D$;

$[W_1, \overline{W_1}] = \mathbf{IIMultF}(\underline{T_1}, \overline{T_1}, \underline{T_2}, \overline{T_2})$;
$[W_2, \overline{W_2}] = \mathbf{StrassMMIn}(A, C)$;
$[W_3, \overline{W_3}] = \mathbf{StrassMMIn}(B, D)$;
$\mathbf{setround}(down)$;
$\underline{P} = \underline{W_2} - \overline{W_3}$;
$\underline{Q} = \underline{W_1} - \overline{W_2} - \overline{W_3}$;
$\underline{G} = \mathbf{complex}(\underline{P}, \underline{Q})$;
$\mathbf{setround}(up)$;
$\overline{P} = \overline{W_2} - \underline{W_3}$;
$\overline{Q} = \overline{W_1} - \underline{W_2} - \underline{W_3}$;
$\overline{G} = \mathbf{complex}(\overline{P}, \overline{Q})$;

したがって、アルゴリズム 13 の計算量は、アルゴリズム 12 の計算量の約 2 倍である。

6 数値実験

提案方式の性能評価のため数値実験を行い、その結果を報告する。

A, B を $n \times n$ ランダム行列とする。それぞれ要素は区間 $[-1, 1]$ に一様分布する乱数である。アルゴリズム 2 の **RealMMIn** に基づく方式とアルゴリズム 11 の **StrassMMIn** に基づく方式の 2 つによって包み込み $[\underline{C}, \overline{C}] \supseteq AB$ を計算する。さらに、AB の近似計算として、通常の計算（これを **RealMM** とする）とアルゴリズム 1 の **StrassMM** の 2 つの方式を用いる。まとめると以下のようになる。

	通常の方法	Strassen の方法
近似計算	**RealMM**	**StrassMM**
包み込み	**RealMMIn**	**StrassMMIn**

計算は、HITACHI SR8000（1 ノード, 14.4GFLOPS）で実行した。行列乗算などに BLAS は使用していない。Strassen の方法は recursive であるが、今回は行列サイズに限らず 1 回だけ適用した。

表 14 は、各計算の CPU time[sec] と GFLOPS（括弧内）を表している。ここで、GFLOPS は以下で定義する：

$$\text{GFLOPS} = \begin{cases} 2n^3/(\text{CPU time})/10^9 & \text{(近似計算)} \\ 4n^3/(\text{CPU time})/10^9 & \text{(包み込み)}. \end{cases}$$

表 14 は以下の事実を示している：

1. **StrassMM** は **RealMM** より明らかに高速である。
2. **StrassMMIn** は、行列サイズが小さいときは **RealMMIn** より低速であるが、行列サイズが大きいときは **RealMMIn** よりも高速であり、その計算コストも行列サイズが大きくなるにつれて **StrassMM** の 2 倍に近づいている。

表 14: CPU time[sec] and GFLOPS on HITACHI SR8000

Dimension (n)	**RealMM**	**StrassMM**	**RealMMIn**	**StrassMMIn**
1000	0.22 (9.09)	0.22 (9.26)	0.44 (9.00)	0.45 (8.87)
2000	1.68 (9.52)	1.55 (10.30)	3.42 (9.35)	3.27 (9.80)
3000	5.59 (9.67)	5.13 (10.52)	11.31 (9.55)	10.64 (10.15)
4000	13.34 (9.60)	12.14 (10.54)	27.13 (9.44)	24.77 (10.34)
5000	26.41 (9.47)	23.45 (10.66)	53.83 (9.29)	48.02 (10.41)

表 15: $\max_{i,j}\{\overline{C}_{ij} - \underline{C}_{ij}\}$

Dimension (n)	**RealMMIn**	**StrassMMIn**
1000	2.5×10^{-12}	3.7×10^{-12}
2000	6.8×10^{-12}	1.1×10^{-11}
3000	1.4×10^{-11}	1.9×10^{-11}
4000	2.1×10^{-11}	3.1×10^{-11}
5000	2.8×10^{-11}	4.3×10^{-11}

次に、表 15 は $C = AB$ の下限 $\underline{C}$ と上限 $\overline{C}$ の差の最大値 $\max_{i,j}\{\overline{C}_{ij} - \underline{C}_{ij}\}$ を表している。すなわち、この値は小さいほうがよりシャープな包み込みができているということになる。

表 15 は、**StrassMMIn** は **RealMMIn** よりも包み込みがシャープではないが、区間幅の増大は 2 倍以内に抑えられていることを示している（もちろん、もっと悪くなるケースもあると考えられるが)。

最後にまとめると、Strassen の方法を用いた提案方式は、包み込みの精度については従来方式と比べてそれほど悪くなく、行列サイズがある程度大きいときは、従来方式よりも高速であり、Strassen の方法を用いた行列乗算の近似計算の約 2 倍の計算コストで行列乗算の包み込みができるということが数値実験によって示された。

また、本提案方式は、後によって提案された Strassen の方法の拡張方式 [7] にも適用可能である。

参考文献

[1] Golub, G. H., Van Loan, C. F.: *Matrix Computations, 3rd ed.*, The Johns Hopkins University Press: Baltimore and London, 1996.

[2] Ogita, T., Oishi, S.: Fast inclusion of interval matrix multiplication, submitted to BIT, 2002.

[3] Oishi, S.: Fast enclosure of matrix eigenvalues and singular values via rounding mode controlled computation, Linear Alg. Appl. **324** (2001), 133–146.

[4] Oishi, S., Rump, S. M.: Fast verification of solutions of matrix equations, Numer. Math. **90** (2002), 755–773.

[5] Rump, S. M.: INTLAB - INTerval LABoratory, Version 3.1, 2002.

[6] Strassen, V.: Gaussian elimination is not optimal, Numer. Math. **13** (1969), 354–356.

[7] 後 保範: 行列乗算におけるストラッセンの方法の拡張, 京大数理研講究録 **1040** (1998), 61–69.

[8] 大石 進一: 精度保証付き数値計算, コロナ社: 東京, 2000.