平成15年度

卒業論文

行列演算における
ストラッセン法の有効性

平成１６年２月５日

指導教授：山本　哲朗　教授

早稲田大学　理工学部　情報学科

1g00p062-4　嶋田　陽介

目 次

1 序論 **6**

1.1 はじめに 7

1.2 本論文の目的 7

1.3 本論文の構成 7

2 準備 **8**

2.1 はじめに 9

2.2 行列の表記法 9

2.3 行列演算 9

2.4 ベクトルの表記法 10

2.5 ベクトルの演算 10

2.6 内積の計算と saxpy 11

2.7 行列×ベクトルの乗算と gaxpy 12

2.8 行と列への行列分割 13

2.9 コロンの表記法 14

2.10 外積による更新 15

2.11 行列と行列の乗算 16

2.12 むすび 17

3 ブロック行列 **18**

3.1 はじめに 19

3.2 ブロック行列の表記法 19
3.3 ブロック行列の操作 20
3.4 部分行列の表記法 22
3.5 ブロック行列×ベクトル 23
3.6 ブロック行列の乗算 24
3.7 むすび 26

4 ストラッセン法 **27**
4.1 はじめに 28
4.2 ストラッセン法の概要 28
4.3 計算量 30
4.4 むすび 30

5 結果 **31**
5.1 はじめに 32
5.2 実行環境 32
5.3 プログラムについて 32
5.4 実行結果 33
5.5 むすび 33

6 まとめ **34**
6.1 はじめに 35
6.2 まとめ 35
6.3 考察 35
6.4 むすび 36

謝辞 **37**

A 付録 **39**

A.1 行列乗算における一般的なアルゴリズムのプログラム 40
A.2 ストラッセン法のプログラム 42

参考文献 **50**

表 目 次

5.1　行列演算における一般的なアルゴリズムとストラッセン法の実行結果 . . .　33

図 目 次

第 1 章

序論

1.1 はじめに

数値計算において行列の乗算 ($C = AB$) は, 逆行列を求める計算と共に頻繁に現れる. そのため実行時間を短縮することが重要な問題になってくる. しかし, 行列の乗算は行列の次数が大きくなるにつれて, 実行時間が加速度的に長くなる. なぜなら計算量が $O(n^3)$ だからである. つまり, 一般的な行列乗算の解法よりも計算量の少ないアルゴリズムを使うことで, 実行時間の短縮を図ることが可能と考えることが出来る.

1.2 本論文の目的

行列の乗算においてストラッセン法は理論的には一般的な方法よりも速いことが分かっている. しかし実際に使用されている例が少ないことから, 理論値と実行値に相違があることが推測される. 本論文ではC言語によるプログラムを作成し, ストラッセン法と一般的な方法との実行結果を比較し, 考察する.

1.3 本論文の構成

本論文の構成は以下の通りである.

第 2 章で, 行列の基礎的な知識, それに付随する表記法について説明する.

第 3 章では, ストラッセン法の基礎となるブロック行列について説明する.

第 4 章では, 行列乗算におけるストラッセン法について説明する.

第 5 章では, 行列演算における一般的なアルゴリズムとストラッセン法を, 実際にプログラムして実行結果を比較する.

第 6 章では, 結論について説明する.

第 2 章

準備

2.1 はじめに

本章では行列の基礎的な知識, それに付随する表記法について説明する.

2.2 行列の表記法

$\mathbf{R}$ を実数の集合とする. 実行列 $m \times n$ のベクトル空間を $\mathbf{R}^{m\times n}$ によって表す.

$$A \in \mathbf{R}^{m\times n} \qquad \Longleftrightarrow \qquad A = (a_{ij}) = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \qquad a_{ij} \in \mathbf{R}. \tag{2.1}$$

大文字が行列を表すなら（e.g. A, B, Δ), 下つき文字 ij によって符合される小文字は (i, j) 要素であることを示す.(e.g. $a_{ij}, b_{ij}, \delta_{ij}$)

適切に,$[A]_{ij}$, 又は $A(i, j)$ という表記法を用いて行列の要素を表す.

2.3 行列演算

転置行列は以下のように表される.($\mathbf{R}^{m\times n} \to \mathbf{R}^{n\times m}$)

$$C = A^T \qquad \Longrightarrow \qquad c_{ij} = a_{ji}. \tag{2.2}$$

加算は以下のように表される.($\mathbf{R}^{m\times n} \times \mathbf{R}^{m\times n} \to \mathbf{R}^{m\times n}$)

$$C = A + B \qquad \Longrightarrow \qquad c_{ij} = a_{ij} + b_{ij}. \tag{2.3}$$

スカラー積は以下のように表される。($\mathbf{R} \times \mathbf{R}^{m\times n} \to \mathbf{R}^{m\times n}$)

$$C = \alpha A \qquad \Longrightarrow \qquad c_{ij} = \alpha a_{ij}. \tag{2.4}$$

行列積は以下のように表される.($\mathbf{R}^{m\times p} \times \mathbf{R}^{p\times n} \to \mathbf{R}^{m\times n}$)

$$C = AB \qquad \Longrightarrow \qquad c_{ij} = \sum_{k=1}^{r} a_{ik} b_{kj}. \tag{2.5}$$

2.4　ベクトルの表記法

$\mathbf{R^n}$ を実数 n ベクトルのベクトル空間とする.

$$x \in \mathbf{R}^n \qquad \Longleftrightarrow \qquad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad x_i \in \mathbf{R}. \tag{2.6}$$

x_i は x の i 番目の要素とみなす. 文脈によって,$[x]_i, x(i)$ それぞれ二つの表記法を適切に使いわける.

$\mathbf{R}^n$ と $\mathbf{R}^{1\times n}$ は同じものとして扱うことができることに気がつけば,$\mathbf{R}^n$ の要素は列ベクトルの集合であることがわかるだろう. 一方で,$\mathbf{R}^{1\times n}$ の要素は, 行ベクトルであることもわかる.

$$x \in \mathbf{R}^{1\times n} \qquad \Longrightarrow \qquad x = (x_1, \cdots, x_n). \tag{2.7}$$

x が列ベクトルならば,$y = x^T$ は行ベクトルである.

2.5　ベクトルの演算

$a \in \mathbf{R}, x \in \mathbf{R}^n, y \in \mathbf{R}^n$ と仮定する.

スカラーベクトル積は以下のように表される.

$$z = ax \qquad \Longleftrightarrow \qquad z_i = ax_i. \tag{2.8}$$

ベクトルの加算は以下のように表される.

$$z = x + y \qquad \Longleftrightarrow \qquad z_i = x_i + y_i. \tag{2.9}$$

内積は以下のように表される.

$$c = x^T y \qquad \Longleftrightarrow \qquad c = \sum_{i=1}^{n} x_i y_i. \tag{2.10}$$

ベクトル積は以下のように表される.

$$z = x. * y \qquad \Longleftrightarrow \qquad z_i = x_i y_i. \tag{2.11}$$

また, 重要な演算として saxpy がある.

$$y = ax + y \qquad \Longleftrightarrow \qquad y_i = ax_i + y_i. \tag{2.12}$$

上式の"="は代入を意味し, 数学的に同等という意味ではない, ベクトル y は更新されることになる. "saxpy"は LAPACK というたくさんのアルゴリズムを含むソフトウェアの中で使われる.

2.6 内積の計算と saxpy

アルゴリズム 2.1 (内積) $x, y \in \mathbf{R}^n$ ならば, このアルゴリズムは $c = x^T y$ を計算する.

$c = 0$

for $i = 1 : n$

$\quad c = c + x(i)y(i)$

end □

二つの n ベクトルの内積は n 回の乗算と n 回の加算を伴う. つまりこのアルゴリズムの計算量は $O(n)$ である.

saxpy の計算量も $O(n)$ であるが, 値はスカラーではなくベクトルで返る.

アルゴリズム 2.2 (saxpy) $x, y \in \mathbf{R}^n, a \in \mathbf{R}$ ならば, このアルゴリズムは $ax + y$ で y を更新する.

for $i = 1 : n$

$\quad y(i) = ax(i) + y(i)$

end □

2.7　行列×ベクトルの乗算とgaxpy

$A \in \mathbf{R}^{m\times n}$ と仮定し, 次の計算をする.

$$y = Ax + y. \tag{2.13}$$

ただし,$x \in \mathbf{R}^n, y \in \mathbf{R}^m$ とする. この一般化された saxpy の演算を”gaxpy”と呼ぶ. gaxpy の基本的な計算手順は, 一度に要素を更新することである.

$$y_i = \sum_{j=1}^{n} a_{ij}x_j + y_i \qquad i = 1 : m. \tag{2.14}$$

これにより次のアルゴリズムが与えられる.

アルゴリズム 2.3 (gaxpy:列バージョン)　$A \in \mathbf{R}^{m\times n}, x \in \mathbf{R}^n, y \in \mathbf{R}^m$ ならば、このアルゴリズムは $Ax + y$ で y を更新する.

for $i = 1 : m$

　　for $j = 1 : n$

　　$y(i) = A(i, j)x(j) + y(i)$

　　end

end　□

アルゴリズム 2.4 (gaxpy:行バージョン)　$A \in \mathbf{R}^{m\times n}, x \in \mathbf{R}^n, y \in \mathbf{R}^m$ ならば, このアルゴリズムは $Ax + y$ で y を更新する.

for $j = 1 : n$

　　for $i = 1 : m$

　　$y(i) = A(i, j)x(j) + y(i)$

　　end

end　□

この２つの gaxpy のアルゴリズムの内部ループは,saxpy 演算を行っていることがわかる. 列バージョンのアルゴリズムは, 行列×ベクトルの乗算がベクトルになることから得られ

る. 一方, 行バージョンのアルゴリズムは, ループの順序を入れ替えることによって得られる. 行列計算において, ループを入れ替えることは重要なことである.

2.8 行と列への行列分割

アルゴリズム 1.3、1.4 は, それぞれ行や列ごとに A のデータをアクセスする. このことをさらに目立たせるために, 分割行列を紹介する.

行の観点から, 行ベクトルのスタックである行列と考える.

$$A \in \mathbf{R}^{m\times n} \qquad \Longleftrightarrow \qquad A = \begin{bmatrix} r_1^T \\ \vdots \\ r_m^T \end{bmatrix} \qquad r_k \in \mathbf{R}^n. \tag{2.15}$$

これを A の行分割と呼ぶ.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \tag{2.16}$$

上式を行分割するならば,A を行の集合として以下のように分割することが出来る.

$$r_1^T = \begin{bmatrix} 1 & 2 \end{bmatrix}, \qquad r_2^T = \begin{bmatrix} 3 & 4 \end{bmatrix}, \qquad r_3^T = \begin{bmatrix} 5 & 6 \end{bmatrix}. \tag{2.17}$$

行分割によって, アルゴリズム (2.3) は以下のように説明できる.

アルゴリズム 2.5

for $i = 1 : m$

$\qquad y_i = r_i^T x + y(i)$

end □

また, 列ベクトルの集合である行列と考える.

$$A \in \mathbf{R}^{m\times n} \qquad \Longleftrightarrow \qquad A = \begin{bmatrix} c_1, & \cdots & c_n \end{bmatrix} \qquad c_k \in \mathbf{R}^m. \tag{2.18}$$

$$c_1 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}, \qquad c_2 = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}. \tag{2.19}$$

これを A の列分割と呼ぶ. 上の 3×2 の例では,c_1, c_2 が A の最初の列と二番目の列にあたることがわかる.

列分割によって, アルゴリズム (2.4) は以下のように説明できる.

アルゴリズム 2.6

for $j = 1 : n$

$y = x_j c_j + y$

end □

2.9 コロンの表記法

コロンの表記法を使うことで, 行列の列や行を明記することが便利になる.$A \in \mathbf{R}^{m \times n}$ ならば,$A(k, :)$ は k 番目の行を表す. すなわち,

$$A(k, :) = [a_{k1}, \cdots, a_{kn}]. \tag{2.20}$$

k 番目の列は以下のように表される.

$$A(:, k) = \begin{bmatrix} a_{1k} \\ \vdots \\ a_{mk} \end{bmatrix}. \tag{2.21}$$

上式よりアルゴリズム 2.3,2.4 をコロンを使って書き換えることが出来る.

アルゴリズム 2.7

for $i = 1 : m$

$y(i) = A(i, :)x + y(i)$

end □

アルゴリズム 2.8

for $j = 1 : n$

$\quad y = x(j)A(:, j) + y$

end □

コロンを使うことによって, 一つ一つ繰り返すのを抑えることが出来る. そのため, ベクトルのレベルで考えることができ, より計算の問題に集中することが出来るようになる.

2.10 外積による更新

コロンの表記法を使って, 外積による更新を理解することが出来る.

$$A = A + xy^T, \qquad A \in \mathbf{R}^{m\times n}, x \in \mathbf{R}^m, y \in \mathbf{R}^n. \tag{2.22}$$

外積の演算 xy^T は一見変に感じるかもしれないが, 全く問題はない. 例えば,

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 4 & 5 \end{bmatrix} = \begin{bmatrix} 4 & 5 \\ 8 & 10 \\ 12 & 15 \end{bmatrix}. \tag{2.23}$$

これは xy^T が二つの行列の積であり, 左の行列 x の列の数と右の行列 y^T の行の数が等しいからである. 外積による更新のアルゴリズムは以下のように記述される.

アルゴリズム 2.9

for $i = 1 : m$

$\quad$ for $j = 1 : n$

$\quad\quad a_{ij} = a_{ij} + x_i y_j$

$\quad$ end

end □

j のループの命令は,A の i 行に y^T の倍数を加えることである. すなわち,

アルゴリズム 2.10

for $i = 1 : m$

$$A(i,:) = A(i,:) + x(i)y^T$$

end □

一方,i のループを内部のループとするなら,i のループの命令は,A の j 列に x の倍数を加えることである. すなわち,

アルゴリズム 2.11

for $j = 1 : n$

$$A(:,j) = A(:,j) + y(j)x$$

end □

つまり, 二つのアルゴリズムは,saxpy による更新の対になっていることがわかる.

2.11 行列と行列の乗算

2×2 の行列と行列の乗算 AB について考える. 内積の公式ではそれぞれの要素は内積として計算される.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1\cdot 5 + 2\cdot 7 & 1\cdot 6 + 2\cdot 8 \\ 3\cdot 5 + 4\cdot 7 & 3\cdot 6 + 4\cdot 8 \end{bmatrix}. \tag{2.24}$$

saxpy では積におけるそれぞれの列は A の一次結合であるとみなせる.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5\begin{bmatrix} 1 \\ 3 \end{bmatrix} + 7\begin{bmatrix} 2 \\ 4 \end{bmatrix}, 6\begin{bmatrix} 1 \\ 3 \end{bmatrix} + 8\begin{bmatrix} 2 \\ 4 \end{bmatrix} \end{bmatrix}. \tag{2.25}$$

最後に外積では, 結果は外積の和であるとみなせる.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \begin{bmatrix} 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix} \begin{bmatrix} 7 & 8 \end{bmatrix} \tag{2.26}$$

これらの行列乗算のバージョンは数学的には同等であるが, 記憶容量の問題で全く違う結果になることがわかる.

2.12 むすび

本章では行列の基礎的な知識, それに付随する表記法について説明した. 次章ではストラッセン法の基礎となるブロック行列について説明する.

第 3 章

ブロック行列

3.1 はじめに

ブロック行列の表記法を使うことは, 行列演算において極めて重要である. なぜならブロック行列の表記法を使うことで, 多くの重要なアルゴリズムを導き出すことが簡単になるからである. さらに, ブロック行列を使ったアルゴリズムは高性能な演算においてますます重要になっている. ブロック行列を使ったアルゴリズムは, 行列と行列の乗算において本質的に有用である. このタイプのアルゴリズムは多くの計算環境においてさらに重要になっていることがわかる.

3.2 ブロック行列の表記法

列と行の分割はブロック行列の特別な場合である. 一般に $m \times n$ 行列 A の行と列を分割すると次のようになる.

$$A = \begin{array}{c} \\ m_1 \\ \\ m_q \end{array} \begin{array}{c} \begin{array}{ccc} n_1 & & n_r \end{array} \\ \begin{pmatrix} A_{11} & \cdots & A_{1r} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qr} \end{pmatrix} \end{array}. \tag{3.1}$$

$m_1 + \cdots + m_q = m, n_1 + \cdots + n_r = n$ で、$A_{\alpha\beta}$ は (α, β) 要素のブロック（部分行列）であることを示す. この表記法を使うことで, ブロック $A_{\alpha\beta}$ は $m_\alpha \times n_\beta$ の次元を持ち,$A = (A_{\alpha\beta})$ は $q \times r$ のブロック行列であると言える.

3.3 ブロック行列の操作

ブロック行列はちょうどスカラー要素を持つ行列のように結び付けられる.

$$B = \begin{array}{c} \\ m_1 \\ \\ m_q \end{array} \begin{array}{c} \begin{array}{ccc} n_1 & & n_r \end{array} \\ \begin{pmatrix} B_{11} & \cdots & B_{1r} \\ \vdots & & \vdots \\ B_{q1} & \cdots & B_{qr} \end{pmatrix} \end{array}. \tag{3.2}$$

もし上式であるならば,B は A の上に分割されたと言える.$C = A + B$ は $q \times r$ のブロック行列とみなせる.

$$C = \begin{bmatrix} C_{11} & \cdots & C_{1r} \\ \vdots & & \vdots \\ C_{q1} & \cdots & C_{qr} \end{bmatrix} = \begin{bmatrix} A_{11} + B_{11} & \cdots & A_{1r} + B_{1r} \\ \vdots & & \vdots \\ A_{q1} + B_{q1} & \cdots & A_{qr} + B_{qr} \end{bmatrix}. \tag{3.3}$$

ブロック行列の乗算は少しトリッキーである. まず二つの補題から説明する.

補題 3.1

$A \in \mathbf{R}_{m\times p}, B \in \mathbf{R}_{p\times n}$

$$A = \begin{array}{c} \\ m_1 \\ \\ m_q \end{array} \begin{pmatrix} A_1 \\ \vdots \\ A_q \end{pmatrix} \qquad B = \begin{array}{c} \begin{array}{ccc} n_1 & & n_r \end{array} \\ \begin{pmatrix} B_1, & \cdots & , B_r \end{pmatrix} \end{array}, \tag{3.4}$$

ならば,

$$AB = C = \begin{array}{c} \\ m_1 \\ \\ m_q \end{array} \begin{array}{c} \begin{array}{ccc} n_1 & & n_r \end{array} \\ \begin{pmatrix} C_{11} & \cdots & C_{1r} \\ \vdots & & \vdots \\ C_{q1} & \cdots & C_{qr} \end{pmatrix} \end{array}. \tag{3.5}$$

$C_{\alpha\beta} = A_\alpha B_\beta$ であり,$\alpha = 1 : q, \beta = 1 : r$ である. □

補題 3.2

$A \in \mathbf{R}_{m\times p}, B \in \mathbf{R}_{p\times n}$

$$A = \begin{matrix} \quad p_1 & & p_s \end{matrix} \begin{pmatrix} A_1, & \cdots & , A_s \end{pmatrix} \qquad B = \begin{matrix} p_1 \\ \\ p_s \end{matrix} \begin{pmatrix} B_1 \\ \vdots \\ B_s \end{pmatrix}, \tag{3.6}$$

ならば,

$$AB = C = \sum_{\gamma=1}^{s} A_\gamma B_\gamma. \tag{3.7}$$

□

一般的なブロック行列の乗算のために, 次の結果を示す.

補題 3.3

$$A = \begin{matrix} \quad p_1 & & p_s \end{matrix} \begin{matrix} m_1 \\ \\ m_q \end{matrix} \begin{pmatrix} A_{11} & \cdots & A_{1s} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qs} \end{pmatrix} \qquad B = \begin{matrix} \quad n_1 & & n_r \end{matrix} \begin{matrix} p_1 \\ \\ p_s \end{matrix} \begin{pmatrix} B_{11} & \cdots & B_{1r} \\ \vdots & & \vdots \\ B_{s1} & \cdots & B_{sr} \end{pmatrix}. \tag{3.8}$$

$$C = \begin{matrix} \quad n_1 & & n_r \end{matrix} \begin{matrix} m_1 \\ \\ m_q \end{matrix} \begin{pmatrix} C_{11} & \cdots & C_{1r} \\ \vdots & & \vdots \\ C_{q1} & \cdots & C_{qr} \end{pmatrix}. \tag{3.9}$$

以上のような $C = AB$ という積に分割できるとするならば,

$$C_{\alpha\beta} = \sum_{\gamma=1}^{s} A_{\alpha\gamma} B_{\gamma\beta} \qquad \alpha = 1 : q, \qquad \beta = 1 : r. \tag{3.10}$$

□

重要なケースとして,$s = 2, r = 1, n_1 = 1$ の場合が挙げられる.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 \\ A_{21}x_1 + A_{22}x_2 \end{bmatrix}. \tag{3.11}$$

この分割した行列とベクトルの積は次節で再び何度も使われる.

3.4 部分行列の表記法

普通の行列の乗算では, ブロック行列の乗算は様々な方法で構成される. 正確に計算を明示するため, 次のいくつかの表記法を使う.

$A \in \mathbf{R}^{m\times n}$ で,$i = (i_1, \cdots, i_r), (j_1, \cdots, j_c)$ は次の属性を持つ整数ベクトルと仮定する.

$$i_1, \cdots, i_r \in \{1, 2, \cdots, m\} \tag{3.12}$$

$$j_1, \cdots, j_c \in \{1, 2, \cdots, n\}. \tag{3.13}$$

$A(i, j)$ を以下のように $r \times c$ の部分行列として表す.

$$A(i, j) = \begin{bmatrix} A(i_1, j_1) & \cdots & A(i_1, j_c) \\ \vdots & & \vdots \\ A(i_r, j_1) & \cdots & A(i_r, j_c) \end{bmatrix}. \tag{3.14}$$

ベクトル i と j の中の要素が隣接しているならば, コロンの表記法は A の中のスカラー要素の観点から $A(i, j)$ を定義することができる. 特に、$1 \leqq i_1 \leqq i_2 \leqq m, 1 \leqq j_1 \leqq j_2 \leqq n$ ならば、$A(i_1 : i_2, j_1, j_2)$ は,i_1 行から i_2 行と j_1 列から j_2 列を抜き出すことによって得られた部分行列である. 例えば以下のような例が挙げられる.

$$A(3 : 5, 1 : 2) = \begin{bmatrix} a_{31} & a_{32} \\ a_{41} & a_{42} \\ a_{51} & a_{52} \end{bmatrix}. \tag{3.15}$$

i と j がスカラーならば,$A(i, :)$ は A の i 行を表し,$A(:, j)$ は A の j 列を表す.

3.5 ブロック行列×ベクトル

定理 3.3 で扱われた重要な場所は, ブロック行列×ベクトルの場合である.gaxpy $y = Ax + y$ の詳細について考える. ただし, $A \in \mathbf{R}^{m\times n}, x \in \mathbf{R}^n, y \in \mathbf{R}^m$ とし,

$$A = \begin{matrix} m_1 \\ \\ m_q \end{matrix}\begin{pmatrix} A_1 \\ \vdots \\ A_q \end{pmatrix} \qquad y = \begin{matrix} m_1 \\ \\ m_q \end{matrix}\begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix}. \tag{3.16}$$

A_i を i 番目のブロックの行とみなす.$m.vec = (m_1, \cdots, m_q)$ がブロックの行の高さのベクトルであるなら,

$$\begin{bmatrix} y_1 \\ \vdots \\ y_q \end{bmatrix} = \begin{bmatrix} A_1 \\ \vdots \\ A_q \end{bmatrix} x + \begin{bmatrix} y_1 \\ \vdots \\ y_q \end{bmatrix}. \tag{3.17}$$

から, 次のアルゴリズムを獲得する.

アルゴリズム 3.1

last= 0

for $i = 1 : q$

first=last+1

last=first+$m.vec(i) - 1$

y(first:last)= A(first:last,:)$x + y$(first:last)

end □

gaxpy の計算をブロック化する別の方法としては, 次のように A と x を分割する方法が

ある.

$$A = \begin{matrix} \quad n_1 \qquad\quad n_r \\ \Big(A_1, \quad \cdots \quad , A_r \Big) \end{matrix} \qquad\qquad x = \begin{matrix} n_1 \\ \\ n_r \end{matrix} \begin{pmatrix} x_1 \\ \vdots \\ x_r \end{pmatrix}. \tag{3.18}$$

この場合 A_j を j 番目のブロックの列とみなす.$n.vec = (n_1, \cdots, n_r)$ がブロックの列幅のベクトルであるなら,

$$y = \begin{bmatrix} A_1, & \cdots & , A_r \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_r \end{bmatrix} + y = \sum_{j=1}^{r} A_j x_j + y. \tag{3.19}$$

から, 次のアルゴリズムを獲得する.

アルゴリズム 3.2

last= 0

for $j = 1 : r$

first=last+1

last=first+$n.vec(j) - 1$

$y = A$(:,first,last)x(first:last)+y

end □

3.6 ブロック行列の乗算

一般的に, スカラーレベルの行列乗算は可能な方法の中で整列される. そのためブロック行列の乗算を行うことが出来る. それぞれのブロック A, B, C によって内積,saxpy, 外積のアルゴリズムのブロックバージョンの準備をすることが出来る. このことを説明するために, これら三つの行列は全て $n \times n$ で $n = Nl$ と仮定する. ただし N, l は正の整数とする.

$A = (A_{\alpha\beta}), B = (B_{\alpha\beta}), C = (C_{\alpha\beta})$ が $l \times l$ ブロックの付随した $N \times N$ ブロック行列ならば, 補題 3.3 から,

$$C_{\alpha\beta} = \sum_{\gamma=1}^{N} A_{\alpha\gamma} B_{\gamma\beta} + C_{\alpha\beta} \qquad \alpha = 1 : N, \quad \beta = 1 : N. \tag{3.20}$$

上式の加算に沿って行列乗算の手順を構成するならば, 次のアルゴリズムを得られる.

アルゴリズム 3.3

for $\alpha = 1 : N$

$i = (\alpha - 1)l + 1 : \alpha l$

for $\beta = 1 : N$

$j = (\beta - 1)l + 1 : \beta l$

for $\gamma = 1 : N$

$k = (\gamma - 1)l + 1 : \gamma l$

$C(i, j) = A(i, k)B(k, j) + C(i, j)$

end

end

end □

$l = 1$ ならば $\alpha \equiv i, \beta \equiv j, \gamma \equiv k$ であることがわかる.

ブロックの saxpy 行列乗算を得るため, 次のように $C = AB + C$ を書く.

$$\left[\begin{array}{ccc} C_1, & \cdots & , C_N \end{array}\right] = \left[\begin{array}{ccc} A_1, & \cdots & , A_N \end{array}\right] \left[\begin{array}{ccc} B_{11}, & \cdots & , B_{1N} \\ \vdots & \ddots & \vdots \\ B_{N1}, & \cdots & , B_{NN} \end{array}\right] + \left[\begin{array}{ccc} C_1, & \cdots & , C_N \end{array}\right]. \tag{3.21}$$

ただし,$A_\alpha, C_\alpha \in \mathbf{R}^{n \times l}, B_{\alpha\beta} \in \mathbf{R}^{l \times l}$ とする. このことから次のアルゴリズムを得る.

アルゴリズム 3.4

for $\beta = 1 : N$

$j = (\beta - 1)l + 1 : \beta l$

for $\alpha = 1 : N$

$i = (\alpha - 1)l + 1 : \alpha l$

$C(:, j) = A(:, i)B(i, j) + C(:, j)$

end

end □

補題 3.2 から,

$$A = \left[\begin{array}{ccc} A_1, & \cdots & , A_N \end{array} \right] \qquad B = \left[\begin{array}{c} B_1^T \\ \vdots \\ B_N^T \end{array} \right], \tag{3.22}$$

ならば,

$$C = \sum_{\gamma=1}^{N} A_\gamma B_\gamma^T + c. \tag{3.23}$$

ただし,$A_\gamma, B_\gamma \in \mathbf{R}^{n \times l}$ とする. これから次のアルゴリズムを得る.

アルゴリズム 3.5

for $\gamma = 1 : N$

$k = (\gamma - 1)l + 1 : \gamma l$

$C = A(:, k)B(k, :) + C$

end □

3.7 むすび

本章ではブロック行列について説明した. 次章ではブロック行列を基にした行列乗算のストラッセン法について説明する.

第 4 章

ストラッセン法

4.1 はじめに

本章では,行列積を少ない計算回数で求める方法として,ストラッセン法について説明する.

4.2 ストラッセン法の概要

まずストラッセン法の議論の最初の点は 2×2 のブロック行列乗算をするということである.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}. \tag{4.1}$$

ただし, それぞれのブロックは平方とする. 一般的なアルゴリズムでは,$C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$ である. このアルゴリズムは8回の乗算と4回の加算を必要とする. しかし,Strassen(1969)は7回の乗算と18回の加算で C を計算する方法を明らかにした.

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \tag{4.2}$$

$$P_2 = (A_{21} + A_{22})B_{11} \tag{4.3}$$

$$P_3 = A_{11}(B_{12} - B_{22}) \tag{4.4}$$

$$P_4 = A_{22}(B_{21} - B_{11}) \tag{4.5}$$

$$P_5 = (A_{11} + A_{12})B_{22} \tag{4.6}$$

$$P_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \tag{4.7}$$

$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22}) \tag{4.8}$$

$$C_{11} = P_1 + P_4 - P_5 + P_7 \tag{4.9}$$

$$C_{12} = P_3 + P_5 \tag{4.10}$$

$$C_{21} = P_2 + P_4 \tag{4.11}$$

$$C_{22} = P_1 + P_3 - P_2 + P_6. \tag{4.12}$$

これらの等式は代入によって簡単に確かめられる. ブロックが $m \times m$ であるように $n = 2m$

と仮定する. $C = AB$ という計算の中にある加算と乗算の数をカウントすると, 一般的な行列乗算は $(2m)^3$ 回の乗算と $(2m)^3 - (2m)^2$ の加算を伴うことがわかる. 対照的にストラッセンのアルゴリズムはブロックの段階で一般的な乗算にあてはめるならば,$7m^3$ 回の乗算と $7m^3 + 11m^2$ 回の加算を要する. $m \gg 1$ ならば, ストラッセンの方法は一般的なアルゴリズムに比べて 7/8 の計算量ですむことになる.

ストラッセンの考えに立ち返ってみよう. ストラッセンのアルゴリズムを P_i で結び付いたそれぞれの半分の大きさのブロック行列乗算にあてはめることが出来る. このように元の A と B が $n \times n$ で,$n = 2^q$ ならば繰り返しストラッセンのアルゴリズムをあてはめることができる. 一番底の段階としてはブロックが 1×1 の時である. もちろんそこまで下げる必要はない. ブロックのサイズがある程度小さいとき ($n \leqq n_{min}$)、P_i がわかるのには一般的な行列乗算を使ったほうが良いかもしれない. 以下に実行手順を示す.

アルゴリズム 4.1

function: C =strass(A, B, n, n_{min})

if $n \leqq n_{min}$

$C = AB$

else

$m = n/2; u = 1 : m; v = m + 1 : n;$

P_1 =strass$(A(u,u) + A(v,v), B(u,u) + B(v,v), m, n_{min})$

P_2 =strass$(A(v,u) + A(v,v), B(u,u), m, n_{min})$

P_3 =strass$(A(u,u), B(u,v) - B(v,v), m, n_{min})$

P_4 =strass$(A(v,v), B(v,u) - B(u,u), m, n_{min})$

P_5 =strass$(A(u,u), +A(u,v)B(v,v), m, n_{min})$

P_6 =strass$(A(v,u) - A(u,u), B(u,u) + B(u,v), m, n_{min})$

P_7 =strass$(A(u,v) - A(v,v), B(v,u) + B(v,v), m, n_{min})$

$C(u,u) = P_1 + P_4 - P_5 + P_7$

$C(u,v) = P_3 + P_5$

$$C(v,u) = P_2 + P_4$$

$$C(v,v) = P_1 + P_3 - P_2 + P_6$$

end □

上のアルゴリズムより, ストラッセンのアルゴリズムは再帰的であることがわかる.

4.3 計算量

ストラッセンのアルゴリズムの計算量は複雑である.$n_{min} \geqq 1$ ならば, 加算の回数と同じように乗算の回数をカウントできる. ただ乗算の回数だけを数えたいのであれば, 再帰の回数を数え, 一回の再帰の中で起こる全ての乗算の回数を数えればよい. ストラッセンのアルゴリズムでは,$q-d$ 個の再分割された行列があるため、7^{q-d} 回の行列と行列の乗算が存在する. この乗算のサイズは n_{min} であるので, ストラッセンのアルゴリズムは $c = (2^q)^3$(一般的な行列乗算のアルゴリズム) と比較して,$s = (2^d)^3 7^{q-d}$ 回の乗算を伴う. つまり,

$$\frac{s}{c} = \left(\frac{2^d}{2^q}\right)^3 7^{q-d} = \left(\frac{7}{8}\right)^{q-d}. \tag{4.13}$$

$d = 0$, すなわち 1×1 まで再帰のレベルを下げるならば,

$$s = \left(\frac{7}{8}\right)^q c = 7^q = n^{\log_2 7} \doteqdot n^{2.807}. \tag{4.14}$$

このように, ストラッセンのアルゴリズムの計算量は $O(n^{2.807})$ である. しかしながら,(乗算の回数に関する) 加算の回数は n_{min} が小さくなるほど重要になる.

4.4 むすび

本章ではストラッセン法について説明した. 次章ではストラッセン法の実行結果を示す.

第 5 章

結果

5.1 はじめに

本章では, 行列演算における一般的なアルゴリズムとストラッセン法を, それぞれ実際にプログラムし, 実行結果を比較した.

5.2 実行環境

CPU Duron 1.10GHz

メモリ 256MB

OS Windows XP

使用言語 C 言語

5.3 プログラムについて

(1) 一般的なアルゴリズムの場合

1. $n \times n$ 行列 A, B を乱数生成させる. 要素の値は $0 \leqq a, b \leqq 9999$ の範囲の整数とする.
2. $C = AB$ の計算を行う.

(2) ストラッセン法の場合

1. $n \times n$ 行列 A, B を乱数生成させる. 要素の値は $0 \leqq a, b \leqq 9999$ の範囲の整数とする.
2. 生成された行列 A, B を,(4.1) 式のように 2×2 のブロック行列に分割する.
3. ストラッセン法によって計算を行う. その際再帰を用い,1×1 の段階まで再帰のレベルを下げる.

5.4 実行結果

以上作成したプログラムを行列 A, B の次元を 2×2 から 1024×1024 まで実行した計算時間を下の表に示す. ただし, ストラッセン法に関して, 再帰を用い,1×1 の段階まで再帰のレベルを下げることから, 行列 A, B の次元は,$n \times n$: $(n = 2^q)$, 実行時間の単位は秒 (second) とする.

n ＼ アルゴリズム	一般的なアルゴリズム	ストラッセン法
2	0.00	0.00
4	0.00	0.00
8	0.00	0.00
16	0.00	0.00
32	0.00	0.02
64	0.00	0.14
128	0.06	1.24
256	0.47	7.38
512	3.75	46.10
1024	29.78	—

表 5.1: 行列演算における一般的なアルゴリズムとストラッセン法の実行結果

5.5 むすび

本章では, 行列演算における一般的なアルゴリズムとストラッセン法を, それぞれ実際にプログラムし, 実行結果を比較した. 次章では, 本章の実行結果の考察を行う.

第 6 章

まとめ

6.1 はじめに

本章では, 本論文の結びとして, 本論文のまとめを述べる.

6.2 まとめ

本論文では, 行列の乗算において理論的に速いことがわかっているストラッセン法に関して, C言語によるプログラムを作成し, ストラッセン法と一般的な方法との実行結果を比較し, 考察した.

第 2 章で, 行列の基礎的な知識, それに付随する表記法について説明した.

第 3 章では, ストラッセン法の基礎となるブロック行列について説明した.

第 4 章では, 行列乗算におけるストラッセン法について説明した.

第 5 章では, 行列演算における一般的なアルゴリズムとストラッセン法を, 実際にプログラムして実行結果を比較した.

本論文の構成は以上のようであった.

6.3 考察

実行結果から, 理論値では速いストラッセン法が一般的なアルゴリズムよりも非常に遅いという結果が出た. そこで, 何点かストラッセン法の実行値が遅いという結果になった要因を考える.

再帰

一般的なアルゴリズムと違ってストラッセン法では再帰が使われる. 再帰の際に, 分割した行列の要素の値を格納しなければならないので, 新たな配列を作っておかなければならない. そこで, 一般的なアルゴリズムに比べてストラッセン法はメモリを余分に取ってしまうことになる. n が 1024 ではエラーが出たことはメモリが足りなくなってしまうことが要因であると考えられる.

コピー

付録にストラッセン法のプログラムを載せているが, 実際にはストラッセンのアルゴリズムだけでなく, もう一つ便宜上の計算が入っている. それがコピーである. (プログラム内では copymat と記述されている関数である.) 例えば,

P_2 =strass$(A(v,u)+A(v,v)), B(u,u), m, n_{min})$

では,$(A(v,u)+A(v,v)$ を A に格納するのは加算によるものであるから, 計算回数に影響は出ない. しかし,$B(u,u)$ を B に格納する場合, 加算も減算も乗算もないので便宜上コピーをすることになる. よって, 計算回数の増加は明白である.

その他の要因

また, 今回のストラッセン法のプログラムでは再帰のレベルを 1×1 まで下げている. 4 章で述べたように再帰のレベルを 1×1 まで下げると加算の回数が大きく影響を及ぼすので, 18 回の加算を要するストラッセン法が遅くなったことの要因として挙げられる. 他には, ストラッセン法は一般的なアルゴリズムに比べて複雑なので, それを制御するための計算や, 関数のオーバーヘッドによって遅くなってしまったことも充分に考えられる.

6.4 むすび

本論文の目的は行列の乗算において理論的に速いことがわかっているストラッセン法に関して, C 言語によるプログラムを作成し, ストラッセン法と一般的な方法との実行結果を比較し, 考察することであった. 実行結果から, ストラッセン法の方が遅く, 一般に使われていない理由が明らかになったと言える. しかし, 先程述べた考察からもわかる通り, 再帰に特化された言語を使うことや, プログラムを更に工夫することで, ストラッセン法の実行速度を上げることは充分に可能な範囲内である. 以上のことを今後の課題として本論文のむすびとしたい.

謝辞

本研究を進めるに当たり, 終始丁寧な御指導及び御激励を賜り, その他多くの面でも色々と御面倒を見て下さり御助言を与えて下さいました山本哲朗教授, に深く感謝いたします.

また、終始丁寧な御指導と御教示をして下さいました山本研究室院生２年, 阿口 誠司氏, 小笹 耕平氏、に大いに感謝いたします.

また, 日常生活において色々とお世話になりました, 山本研究室４年井上 陽介氏, 佐藤 裕介氏, 藤田 祐作氏に深く感謝いたします.

最後に, 研究だけでなく日常の生活の中でお世話になりました山本研究室の皆様に深く感謝いたします.

Appendix A

付録

A.1　行列乗算における一般的なアルゴリズムのプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* 行列の生成と初期化 */

void gen_mat(int n,int **a,int **b,int **c){

    int i,j;

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            a[i][j]=rand()/10;
            b[i][j]=rand()/10;
            c[i][j]=0;
        }
    }
}

/* 行列の表示 */

void pri_mat(int n, int **a){

    int i,j;

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            printf(" %4d", a[i][j]);
        }
        printf("¥n");
    }
}

/* 行列の乗算 */

void mul_mat(int n,int **a,int **b,int **c){

    int i,j,k;

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            for(k=0;k<n;k++){
                c[i][j]+=a[i][j]*b[i][j];
            }
        }
    }
}

/* 答えの表示 */

void pri_ans(int n,int **c){

    int i,j;
```

```
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            printf(" %8d",c[i][j]);
        }
        printf("¥n");
    }
}

int main(void);

int main(void){

    int **a,**b,**c;
    int i,m,n;
    clock_t start, end;

    printf("次元の数は？¥n");
    scanf("%d", &n);

    a=(int**)malloc(n*sizeof(int*));
    b=(int**)malloc(n*sizeof(int*));
    c=(int**)malloc(n*sizeof(int*));

    for(i=0;i<n;i++){
        a[i]=(int*)malloc(n*sizeof(int));
        b[i]=(int*)malloc(n*sizeof(int));
        c[i]=(int*)malloc(n*sizeof(int));
    }

    srand((unsigned int)time(NULL));

    gen_mat(n,a,b,c);

    printf("行列を表示しますか？(1:YES ELSE:NO)¥n");
    scanf("%d", &m);

    if(m==1){
        printf("A=¥n");
        pri_mat(n,a);
        printf("B=¥n");
        pri_mat(n,b);
    }

    printf("答えを表示しますか？(1:YES ELSE:NO)¥n");
    scanf("%d", &m);

    start=clock();

    mul_mat(n,a,b,c);

    end=clock();

    if(m==1){
        printf("C=¥n");
        pri_ans(n,c);
    }
```

```
    free(a);free(b);free(c);

    printf("%.2f 秒 ¥n",(double)(end-start)/CLOCKS_PER_SEC);

    return(0);
}
```

A.2 ストラッセン法のプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* 行列の生成と初期化 */

void gen_mat(int n,int *****a,int *****b,int *****c,
int *****p,int *****d,int *****e){

    int i,j,k,l,m;

    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            for(k=0;k<n;k++){
                for(l=0;l<n;l++){
                    a[0][i][j][k][l]=rand()/10;
                    b[0][i][j][k][l]=rand()/10;
                    c[0][i][j][k][l]=0;
                }
            }
        }
    }

    for(i=0;i<2;i++){
        for(j=0;j<7;j++){
            for(k=0;k<n;k++){
                for(l=0;l<n;l++){
                    p[0][i][j][k][l]=0;
                }
            }
        }
    }

    for(i=0;i<20;i++){
        for(j=0;j<2;j++){
            for(k=0;k<2;k++){
                for(l=0;l<n;l++){
                    for(m=0;m<n;m++){
                        d[i][j][k][l][m]=0;
                        e[i][j][k][l][m]=0;
                    }
                }
            }
        }
```

```
    }
}

/* 行列の表示 */

void pri_mat(int n, int *****a){

    int i,j,k,l;

    for(i=0;i<2;i++){
        for(j=0;j<n;j++){
            for(k=0;k<2;k++){
                for(l=0;l<n;l++){
                    printf(" %4d", a[0][i][k][j][l]);
                }
            }
            printf("¥n");
        }
    }
}

/* 行列の乗算 */

void mul_mat(int m,int i,int j,int k,int l,int p,
int *****a,int *****b,int *****c){

    c[0][0][p][0][0]=a[m][i][j][0][0]*b[m][k][l][0][0];
    printf("c=%d¥n",c[0][0][p][0][0]);
}

/* 行列の加算 */

void add_mat(int m,int n,int i1,int i2,int j1,int j2,
int *****a,int *****b,int *****c){

    int i,j,k,l;

    if(n==1){
        c[m][0][0][0][0]=a[m-1][i1][i2][0][0]+b[m-1][j1][j2][0][0];
    }

    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            for(k=0;k<n/2;k++){
                for(l=0;l<n/2;l++){
                    c[m][i][j][k][l]
                    =a[m-1][i1][i2][i*n/2+k][j*n/2+l]
                    +b[m-1][j1][j2][i*n/2+k][j*n/2+l];
                }
            }
        }
    }
    printf("%d¥n",c[m][0][0][0][0]);
}

void add2_mat(int m,int n,int i1,int i2,int j1,int j2,int *****a,int *****c){
```

```
    int i,j;

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            c[0][j1][j2][i][j]+=a[0][i1][i2][i][j];
        }
    }
}

/* 行列の減算 */

void sub_mat(int m,int n,int i1,int i2,int j1,int j2,
int *****a,int *****b,int *****c){

    int i,j,k,l;

    if(n==1){
        c[m][0][0][0][0]=a[m-1][i1][i2][0][0]-b[m-1][j1][j2][0][0];
    }

    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            for(k=0;k<n/2;k++){
                for(l=0;l<n/2;l++){
                    c[m][i][j][k][l]
                    =a[m-1][i1][i2][i*n/2+k][j*n/2+l]
                    -b[m-1][j1][j2][i*n/2+k][j*n/2+l];
                }
            }
        }
    }
    printf("%d¥n",c[m][0][0][0][0]);

}

void sub2_mat(int m,int n,int i1,int i2,int j1,int j2,int *****a,int *****c){

    int i,j;

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            c[0][j1][j2][i][j]-=a[0][i1][i2][i][j];
        }
    }
}

/* コピー */

void copy_mat(int m,int n,int i1,int i2,int *****a,int *****c){

    int i,j,k,l;

    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            for(k=0;k<n;k++){
                for(l=0;l<n;l++){
                    c[m][i][j][k][l]=a[m-1][i1][i2][i*n/2+k][j*n/2+l];
```

```
                }
            }
        }
    }
}

/* ストラッセン法 */

void str(int m,int n,int *****a,int *****b,int *****c,
int *****p,int *****d,int *****e){

    add_mat(m,n,0,0,1,1,a,a,d);
    add_mat(m,n,0,0,1,1,b,b,e);

    if(n==1){
        mul_mat(m,0,0,0,0,0,d,e,p);
    }
    else{
        n=n/2;
        m++;
        str(m,n,d,e,c,p,d,e);
    }

    add_mat(m,n,1,0,1,1,a,a,d);
    copy_mat(m,n,0,0,b,e);
    if(n==1){
        mul_mat(m,0,0,1,1,1,d,e,p);
    }
    else{
        n=n/2;
        m++;
        str(m,n,a,b,c,p,d,e);
    }

    copy_mat(m,n,0,0,a,d);
    sub_mat(m,n,0,1,1,1,b,b,e);
    if(n==1){
        mul_mat(m,0,0,0,0,2,d,e,p);
    }
    else{
        n=n/2;
        m++;
        str(2,n,a,b,c,p,d,e);
    }

    copy_mat(m,n,1,1,a,d);
    sub_mat(m,n,1,0,0,0,b,b,e);
    if(n==1){
        mul_mat(m,1,1,0,0,3,d,e,p);
    }
    else{
        n=n/2;
        m++;
        str(3,n,a,b,c,p,d,e);
    }

    add_mat(m,n,0,0,0,1,a,a,d);
```

```
    copy_mat(m,n,1,1,b,e);
    if(n==1){
        mul_mat(m,0,0,1,1,4,d,e,p);
    }
    else{
        n=n/2;
        m++;
        str(4,n,a,b,c,p,d,e);
    }

    sub_mat(m,n,1,0,0,0,a,a,d);
    add_mat(m,n,0,0,0,1,b,b,e);
    if(n==1){
        mul_mat(m,0,0,0,0,5,d,e,p);
    }
    else{
        n=n/2;
        m++;
        str(5,n,a,b,c,p,d,e);
    }

    sub_mat(m,n,0,1,1,1,a,a,d);
    add_mat(m,n,1,0,1,1,b,b,e);
    if(n==1){
        mul_mat(m,0,0,0,1,6,d,e,p);
    }
    else{
        n=n/2;
        m++;
        str(6,n,a,b,c,p,d,e);
    }

    add2_mat(m,n,0,0,0,0,p,c);
    add2_mat(m,n,0,3,0,0,p,c);
    sub2_mat(m,n,0,4,0,0,p,c);
    add2_mat(m,n,0,6,0,0,p,c);

    add2_mat(m,n,0,2,0,1,p,c);
    add2_mat(m,n,0,4,0,1,p,c);

    add2_mat(m,n,0,1,1,0,p,c);
    add2_mat(m,n,0,3,1,0,p,c);

    add2_mat(m,n,0,0,1,1,p,c);
    add2_mat(m,n,0,2,1,1,p,c);
    sub2_mat(m,n,0,1,1,1,p,c);
    add2_mat(m,n,0,5,1,1,p,c);
}

/* 答えの表示 */

void pri_ans(int n, int *****c){

    int i,j,k,l;

    for(i=0;i<2;i++){
        for(j=0;j<n;j++){
```

```
            for(k=0;k<2;k++){
                for(l=0;l<n;l++){
                    printf(" %8d", c[0][i][k][j][l]);
                }
            }
            printf("¥n");
        }
    }
}

int main(void);

int main(void){

    int *****a,*****b,*****c,*****p,*****d,*****e;
    int i,j,k,l,m,n;
    clock_t start,end;

    printf("次元の数は？¥n");
    scanf("%d", &n);

    n=n/2;

    a=(int*****)malloc(1*sizeof(int****));
    b=(int*****)malloc(1*sizeof(int****));
    c=(int*****)malloc(1*sizeof(int****));


    for(i=0;i<1;i++){
        a[i]=(int****)malloc(2*sizeof(int***));
        b[i]=(int****)malloc(2*sizeof(int***));
        c[i]=(int****)malloc(2*sizeof(int***));
        for(j=0;j<2;j++){
            a[i][j]=(int***)malloc(2*sizeof(int**));
            b[i][j]=(int***)malloc(2*sizeof(int**));
            c[i][j]=(int***)malloc(2*sizeof(int**));
            for(k=0;k<2;k++){
                a[i][j][k]=(int**)malloc(n*sizeof(int*));
                b[i][j][k]=(int**)malloc(n*sizeof(int*));
                c[i][j][k]=(int**)malloc(n*sizeof(int*));
                for(l=0;l<n;l++){
                    a[i][j][k][l]=(int*)malloc(n*sizeof(int));
                    b[i][j][k][l]=(int*)malloc(n*sizeof(int));
                    c[i][j][k][l]=(int*)malloc(n*sizeof(int));
                }
            }
        }
    }

    p=(int*****)malloc(1*sizeof(int****));

    for(i=0;i<1;i++){
        p[i]=(int****)malloc(2*sizeof(int***));
        for(j=0;j<2;j++){
            p[i][j]=(int***)malloc(7*sizeof(int**));
            for(k=0;k<7;k++){
                p[i][j][k]=(int**)malloc(n*sizeof(int*));
```

```
            for(l=0;l<n;l++){
                p[i][j][k][l]=(int*)malloc(n*sizeof(int));
            }
        }
    }
}

d=(int*****)malloc(20*sizeof(int****));
e=(int*****)malloc(20*sizeof(int****));

for(i=0;i<20;i++){
    d[i]=(int****)malloc(2*sizeof(int***));
    e[i]=(int****)malloc(2*sizeof(int***));
    for(j=0;j<2;j++){
        d[i][j]=(int***)malloc(2*sizeof(int**));
        e[i][j]=(int***)malloc(2*sizeof(int**));
        for(k=0;k<2;k++){
            d[i][j][k]=(int**)malloc(n*sizeof(int*));
            e[i][j][k]=(int**)malloc(n*sizeof(int*));
            for(l=0;l<n;l++){
                d[i][j][k][l]=(int*)malloc(n*sizeof(int));
                e[i][j][k][l]=(int*)malloc(n*sizeof(int));
            }
        }
    }
}

srand((unsigned int)time(NULL));

gen_mat(n,a,b,c,p,d,e);

printf("行列を表示しますか？(1:YES ELSE:NO)¥n");
scanf("%d", &m);

if(m==1){
    printf("A=¥n");
    pri_mat(n,a);
    printf("B=¥n");
    pri_mat(n,b);
}

printf("答えを表示しますか？(1:YES ELSE:NO)¥n");
scanf("%d", &m);

start=clock();

str(1,n,a,b,c,p,d,e);

end=clock();

if(m==1){
    printf("C=¥n");
    pri_ans(n,c);
}

free(a);free(b);free(c);free(p);free(d);
```

```
    printf("%.2f秒¥n",(double)(end-start)/CLOCKS_PER_SEC);

    return(0);
}
```

参考文献

[1] Gene H.Golub,Charles F.Van Loan:"Matrix Computations",The Johns Hopkins University Press,Third Edition,1996 年

[2] Volker Strassen:"Gaussian Elimination is not Optimal",Numer.Math.13,354-356,1969 年