第 5 章

MATLAB を用いた基本線型計算のベンチマークテスト

5.1 基本線型計算のアルゴリズムと計算量

5.1.1 浮動小数点数の四則演算と Landau の O 記号

小学校で習う小数の加減乗除は整数のそれと本質的には同じものである。人間の感覚で言う「面倒くさい」計算は，そのままコンピュータについても当てはまる。面倒な計算は時間を要する。従って四則演算は前章のベンチマークテストからも分かるように，

$$T(\text{加算 (FADD)}) = T(\text{減算 (FSUB)}) \leq T(\text{乗算 (FMUL)}) < T(\text{除算 (FDIV)}) \tag{5.1}$$

という順に計算時間を要すると考えて良い。後で述べる初等関数はこれらの演算を組み合わせて実行されるため，更に時間を要するのが普通である。但し，現在の CPU は内部に積み込んだ高速転送可能なキャッシュ (cache) メモリを持っており，一度メインメモリから読み込んだ値をそこに記憶しておき，2 度目以降のアクセスはそれを取り出すだけで済む。従って，このキャッシュメモリをうまく利用できるようにした線型計算プログラムは，素朴に組んだものより高速になる。よって単純に計算量だけでは計算時間を推定できないこともある。

従って，数値計算のアルゴリズムの計算時間は加減算の実行回数以上に，乗除算や初等関数の実行回数に左右される。「アルゴリズムの演算回数」という言葉がしばしば後者の意味で使われるのはそのためである。

計算回数に限らず，様々な場面で使用される言葉としてオーダ (order) がある。これは以下に示す Landau の O(ラージオー) と同義である。

定義 5.1 (Landau の O 記号)
ある一変数実関数 $f(x), g(x) \in \mathbb{R}$ に対して，$f(x) = O(g(x))$ とは

$$\lim_{x \to \alpha} \frac{f(x)}{g(x)} = \text{定数 } (\neq 0)$$

となることを意味し，このような $f(x)$ は $g(x)$ のオーダ (order) であると呼ぶ。この $O(g(x))$ を Landau の O(ラージオー) 記号という。α としては 0 もしくは $\pm\infty$ がよく使用される。

また

$$\lim_{x \to \alpha} \frac{f(x)}{g(x)} = 0$$

であるときは特に $f(x) = o(g(x))$ と書き，これを o(スモールオー) 記号と呼ぶ。

以降，オーダという言葉は O(ラージオー) の意味で使用する。$g(x)$ としてよく使用されるのは x の多項式であり，特に $x^2, x^3, ...x^n$ である。$O(x^n)$ はほぼ x^n に比例していることを表しており，直感的に理解しやすいため，様々な場面で使用される。

5.1.2　複素数の四則演算

本書は実数の演算が主体であるが，複素数の演算が必要となる場面に遭遇することもある。ここで復習も兼ねて，複素数の演算とその演算量について若干の考察を行う。

任意の複素数 $c = \mathrm{Re}(c) + \mathrm{Im}(c)\sqrt{-1} \in \mathbb{C}$ は 2 つの実数の組 $(\mathrm{Re}(c), \mathrm{Im}(c))$ として表現できる。従って，複素数の四則演算は全て実数のそれを組み合わせることによって実現できる。

$$|a| = \sqrt{(\mathrm{Re}(a))^2 + (\mathrm{Im}(a))^2} \tag{5.2}$$

$$a \pm b = (\mathrm{Re}(a) \pm \mathrm{Re}(b)) + \sqrt{-1}(\mathrm{Im}(a) \pm \mathrm{Im}(b)) \tag{5.3}$$

$$ab = (\mathrm{Re}(a)\mathrm{Re}(b) - \mathrm{Im}(a)\mathrm{Im}(b)) + \sqrt{-1}(\mathrm{Im}(a)\mathrm{Re}(b) + \mathrm{Re}(a)\mathrm{Im}(b)) \tag{5.4}$$

$$a/b = \frac{a\bar{b}}{|b|^2} \tag{5.5}$$

ここで $\bar{b} = \mathrm{Re}(b) - \mathrm{Im}(b)\sqrt{-1}$ である。

但し，オーバーフローを防止するため，$|a|$ と a/b は次のように計算するのが良いとされている [2]。

$$|a| = \begin{cases} \mathrm{Re}(a) & (\text{if } \mathrm{Im}(a) = 0) \\ \mathrm{Im}(a) & (\text{if } \mathrm{Re}(a) = 0) \\ |\mathrm{Re}(a)|\sqrt{1 + \left(\frac{\mathrm{Im}(a)}{\mathrm{Re}(a)}\right)^2} & (\text{if } |\mathrm{Re}(a)| \geq |\mathrm{Im}(a)| > 0) \\ |\mathrm{Im}(a)|\sqrt{1 + \left(\frac{\mathrm{Re}(a)}{\mathrm{Im}(a)}\right)^2} & (\text{if } |\mathrm{Im}(a)| > |\mathrm{Re}(a)| > 0) \end{cases} \tag{5.6}$$

$$a/b = \begin{cases} \text{計算不能} & (\text{if } b = 0) \\ & (\text{即ち } \mathrm{Re}(b) = \mathrm{Im}(b) = 0) \\ \dfrac{\mathrm{Re}(a) + \mathrm{Im}(a)\cdot\left(\frac{\mathrm{Im}(b)}{\mathrm{Re}(b)}\right)}{s} + \dfrac{-\mathrm{Re}(a)\cdot\left(\frac{\mathrm{Im}(b)}{\mathrm{Re}(b)}\right) + \mathrm{Im}(a)}{s}\sqrt{-1} & \\ & (\text{if } |\mathrm{Re}(b)| \geq |\mathrm{Im}(b)| \geq 0) \\ \text{ここで } s = \mathrm{Re}(b) + Im(b)\cdot\left(\frac{Im(b)}{\mathrm{Re}(b)}\right) & \\ \dfrac{\mathrm{Re}(a)\cdot\left(\frac{\mathrm{Re}(b)}{\mathrm{Im}(b)}\right) + \mathrm{Im}(a)}{s} + \dfrac{-\mathrm{Re}(a) + \mathrm{Im}(a)\cdot\left(\frac{\mathrm{Re}(b)}{\mathrm{Im}(b)}\right)}{s}\sqrt{-1} & \\ & (\text{if } |\mathrm{Im}(b)| \geq |\mathrm{Re}(b)| \geq 0) \\ \text{ここで } s = \mathrm{Re}(b)\cdot\left(\frac{\mathrm{Re}(b)}{\mathrm{Im}(b)}\right) + \mathrm{Im}(b) & \end{cases} \tag{5.7}$$

以上の複素数演算の計算回数を表 5.1 にまとめておく。対応する実数の演算と比べて，2〜3 倍の演算量を必要とすることが分かる。従って，複素数の演算は実数のそれに比べてかなり「高くつく」ことを認識しておく必要がある。当然のことながら，必要となるメモリ量も実数の 2 倍になる。

例題 5.1

1. 式 (5.6)，(5.7) がそれぞれ $|a|$ と a/b を計算していることを確認せよ。
2. 前章の多倍長浮動小数点数の四則演算のベンチマークテストの結果を用いて，複素数演算の性能評価を行え。また実際にベンチマークテストを行った結果と比較せよ。

表 5.1　複素数演算の計算回数

	加減算	乗算	除算	平方根
$\|a\|$ ((5.2) 式)	1	2	0	1
$\|a\|$ ((5.6) 式)	1	2	1	1
$a \pm b$	2	0	0	0
ab	2	4	0	0
a/b ((5.5) 式)	3	6	2	0
a/b ((5.7) 式)	3	3	3	0

表 5.2　ベクトル演算の計算回数

	加減算	乗算	演算量
$\alpha\mathbf{a}$	0	n	$n = O(n)$
$\mathbf{a} \pm \mathbf{b}$	n	0	$n = O(n)$
$(\mathbf{a}, \mathbf{b})$	$n - 1$[a]	n	$2n - 1 = O(n)$

[a] 実際の計算ではあらかじめゼロクリアした変数に成分の積を足し込むため，n となることもある。

表 5.3　行列演算の計算回数

	加減算	乗算	演算量
$A\mathbf{b}$	$n(n-1)$[a]	n^2	$2n^2 - n = O(n^2)$
αA	0	n^2	$n^2 = O(n^2)$
AB	$n^2(n-1)$[b]	n^3	$2n^3 - n^2 = O(n^3)$

[a] 実際の計算ではあらかじめゼロクリアした変数に成分の積を足し込むため，n^2 となることもある。
[b] 実際の計算ではあらかじめゼロクリアした変数に成分の積を足し込むため，n^3 となることもある。

5.1.3　基本線型計算

現在の数値計算は大規模化が進んでおり，そこではベクトル及び行列の基本線型計算 (linear compuation) が多用される。基本線型計算は次元数が上がるにつれて莫大な計算量を必要とすることを認識しておく必要がある。ここではその一端に触れることにする。

実ベクトル $\mathbf{a} = [a_1\ a_2\ \cdots a_n]^T \in \mathbb{R}^n$ の基本線型計算，および，実正方行列 $A = [a_{ij}] \in \mathbb{R}^{n\times n}$ $(i, j = 1, 2, ..., n)$ の基本線型計算の計算回数を表 5.2，5.3 にまとめておく。

5.2　ループを用いたベクトルと行列の定義

以上，MATLAB におけるベクトル，行列の扱い方を見てきたが，要素をすべて決め打ちで入力してきたものばかりであった。もっと大きなサイズのベクトルや行列を扱う際には，要素の規則性を利用したり，ファイルから要素を読み込んだりする必要が出てくる。そのためには繰り返しの処理を記述するためのループを記述する。C/C++ 言語同様，do 文, while 文, for 文の 3 つのループ記述方法があるが，ここでは for 文のみを用いることにする。

for ループの使い方は

```
for 変数=初期値:(間隔値:) 終了値
   命令
end;
```

とする。間隔値を省略すると 1 ずつ増加させていくことになる。

例えば，$\mathbf{v} = [-1\ -2\ -3\ -4\ -5]^T \in \mathbb{R}^5$ というベクトルを変数 vec_v にセットするためには次のようにすればよい。

```
vec_v = [] % 空のリスト (ベクトル)
for i = 1:5
```

```
    vec_v(i,1) = -i;
end;
disp('vec_v = '); disp(vec_v); % vec_v を表示
```

同様に，行列も for ループを二重にすることで値の設定が可能となる。例えば $A \in \mathbb{R}^{n\times n}$ が

$$A = [-(i+j)-1]_{i,j=1}^{n} = \begin{bmatrix} -1 & -2 & \cdots & -n \\ -2 & -3 & \cdots & -(n+1) \\ \vdots & \vdots & & \vdots \\ -n & -(n+1) & \cdots & -(2n-1) \end{bmatrix}$$

という行列である時，$n = 5$ の時は次のように指定すればよい。

```
mat_a = []; % 空のリスト (行列)
n = 5; % 次元数
for i = 1:n
    for j = 1:n
        mat_a(i, j) = -(i + j - 1);
    end;
end;
disp('mat_a = '); disp(mat_a); % mat_a を表示
```

より複雑な行列 $B \in \mathbb{R}^{n\times n}$ として

$$B = \begin{bmatrix} n & n-1 & \cdots & 1 \\ n-1 & n-1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} = [n - \max(i,j) + 1]_{i,j=1}^{n}$$

を考えてみよう。定義は

```
mat_b = [];
n = 5; % 5 次元とする
for i = 1:n
    for j = 1:n
        mat_b(i, j) = n - max(i, j) + 1;
    end;
end;
disp('b = '); disp(mat_b);
```

とすればよい。
更に逆行列を inv 関数を用いて計算してみる。

```
disp('b^(-1) = ');
mat_b_inv = inv(mat_b);
disp(mat_b_inv);
```

さすれば，$BB^{-1} = I_n$ となるはずである。それを確認してみよう。

```
disp("b * b^(-1) = ");
disp(mat_b * mat_b_inv);
```

実際にそうなっているかどうかの確認をノルム絶対誤差 $\|I - B\widetilde{B^{-1}}\|_2$ を使って計算してみよう。ちなみに単位行列は eye 関数を用いて定義できる。

```
disp('|| I - B * B^(-1) ||_2 = ');
disp(norm(eye(n,n) - mat_b * mat_b_inv));
```

実行結果については各自きちんと解釈できるようになること。

問題 5.1

ヒルベルト行列 $H \in \mathbb{R}^{n\times n}$ は

$$H = [1/(i+j-1)]_{i,j=1}^{n} = \begin{bmatrix} 1 & 1/2 & \cdots & 1/n \\ 1/2 & 1/3 & \cdots & 1/(n+1) \\ \vdots & \vdots & \vdots & \\ 1/n & 1/(n+1) & \cdots & 1/(2n-1) \end{bmatrix}$$

である。任意の $n \in \mathbb{N}$ に対して n 次のヒルベルト行列を生成するスクリプト “matrix_hilbert.m”を作り，$n = 3, 4, 5$ として逆行列 H^{-1} とノルム絶対誤差 $\|I - \widetilde{H}\widetilde{H^{-1}}\|_2$ も求めよ。

5.3　基本線型計算のベンチマークテスト

MATLAB スクリプトで，基本線型計算の演算量を確認してみよう。まず $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^{n}$

$$A = \begin{bmatrix} \sqrt{2}(n-2) & \sqrt{2}(n-3) & \cdots & \sqrt{2}(-1) \\ \sqrt{2}(n-3) & \sqrt{2}(n-4) & \cdots & \sqrt{2}(-2) \\ \vdots & \vdots & & \vdots \\ \sqrt{2}(-1) & \sqrt{2}(-2) & \cdots & \sqrt{2}(-n) \end{bmatrix} = [\ \sqrt{2}(n-(i+j)]_{i,j=1}^{n}$$

$$\mathbf{b} = \begin{bmatrix} \sqrt{2} \\ 2\sqrt{2} \\ \vdots \\ n\sqrt{2} \end{bmatrix}$$

を用いて行列・ベクトル積 $A\mathbf{b}$ の計算と，加法・乗法 1 回あたりの演算時間を計測するスクリプト (benchmark.m) を以下に示す。なお，行頭の「数字:」は便宜上行番号を付加したものなので，スクリプト作成時にはコロン「:」の次の文字から打ち込んでいくこと。

```
 1: % benchmark.m: 行列ベクトル積のベンチマークテスト
 2:
 3: %  見出し
 4: fprintf("行列ベクトル積ベンチマークテスト\n");
 5: fprintf("次元数,        秒数,     GFlops\n")
 6:
 7: %  次元数セット
 8: index = 1;
 9: graph_x = [];
10: graph_gflops = [];
11: graph_sec = [];
12:
13: % 演算繰り返し回数
14: max_iter_times = 20000;
15:
16: % メインループ
17: for dim = 500:100:1500
18:
19:     %  ベクトル (vec) のセット
20:     vec = [];
21:     for i = 1:dim
22:         vec(i, 1) = sqrt(2) * i;
23:     end
24:
25:     %  行列 (mat) のセット
26:     mat = [];
27:     for i = 1:dim
28:         for j = 1:dim
29:             mat(i, j) = sqrt(2) * (dim - (i + j));
30:         end
```

```
31:     end
32:
33:     %  行列・ベクトル積の計算
34:     max_vec_mul_time = 0;
35:     tic;
36:     for iter_times = 1:max_iter_times
37:         vec_ret = mat * vec;
38:     end
39:     mat_vec_mul_time = toc / max_iter_times;
40:
41:     %  グラフ描画用
42:     graph_x(index) = dim;
43:     graph_sec(index) = mat_vec_mul_time;
44:     graph_gflops(index) = (dim * dim * 2) / graph_sec(index) / 1024^3;
45:     fprintf("%6d, %10.3g, %10.3g\n", dim, graph_sec(index), graph_gfl
ops(index));
46:
47:     index = index + 1;
48: end %  ベンチマーク終了
49:
50: %  グラフ描画
51: plot(graph_x, graph_gflops);
```

これを実行すると，まず 29 行目の行列・ベクトル積の計算に要した計算時間が変数 `mat_vec_mul_time` に格納される。これを使って加法と乗法の演算量を合計した $2n^2$ を割ると，1 秒間に何回の浮動小数点演算が実行できるか，即ち，Flops(FLoating-point Operations Per Second) 値が算出できる。桁が大きすぎるため，ここでは 1024^3 で割って，GFlops(Giga Flops) を算出してある。

算出された値を順次出力し，

```
行列ベクトル積ベンチマークテスト
次元数,       秒数,     GFlops
   500,   2.99e-05,       15.6
   600,   3.92e-05,       17.1
   700,   5.82e-05,       15.7
   800,   9.56e-05,       12.5
   900,   0.000231,       6.54
  1000,   0.000385,       4.84
・・・・・・・・・・・・・・・・
```

のような出力結果を得たら，GFlops 値をプロットしたグラフを最後に出力している。

問題 5.2

1. benchmark.m を参考にして，正方行列 A, B の積 AB の演算時間を計測し，GFlops 値を求める MATLAB スクリプト，benchmark_mat.m を作れ。行列 $A = \sqrt{2}[i + j - 1]$, $B = \sqrt{3}[2n - (i + j - 1)]$ は下記のように与えよ。

```
% 行列 a(mat_a) のセット
mat_a = [];
for i = 1:dim
    for j = 1:dim
        mat_a(i, j) = sqrt(2) * (i + j - 1);
    end
end;

% 行列 b(mat_b) のセット
mat_b = [];
for i = 1:dim
    for j = 1:dim
        mat_b(i, j) = sqrt(3) * (dim * 2 - (i + j - 1));
    end
end
```

2. benchmark.m を参考にして，複素正方行列 A, B の積 AB の演算時間を計測し，GFlops 値を求める MATLAB スクリプト，benchmark_cmat.m を作れ。行列はそれぞれ $A = \sqrt{2}[(i + j - 1) + (i + j - 1)\mathrm{i}]$, $B = \sqrt{3}[(2n - (i +$

$j-1)) - (2n-(i+j-1))$i] と設定せよ。