数式処理 *Bulletin of JSSAC (200X)*
Vol. XX, No. X, pp. 1 - 4

大会報告

整数行列の最小多項式の候補の計算について

木村 欣司 *

京都大学大学院情報学研究科

整数行列の最小多項式の候補を高速に計算するための技法を紹介する. 入力行列が, 疎行列または密行列の場合の両方を想定し, さらに, 逐次計算と並列計算の両方において, 最適な計算方法を検討する. ただし, 本稿では, 内積, AXPY, 行列ベクトル乗算のみを用いることとし, 行列乗算を用いた工夫は行わないものとする.

1 C 言語レベルでの工夫

1.1 minval 関数について

本稿の主題のために必要とする機能ではないが, C 言語には, 配列の最大値・最小値を計算する場合に, SIMD 演算を明示的に使うようにコンパイラに指示できないという問題点がある. Fortran では,
変数=minval(“配列名”(開始位置：終了位置：ストライド))
とすると, SIMD 演算を用いて, 高速に “配列名” の最小値を計算してくれるコードを自動的に生成する. 一方, C 言語では
tmp=A[0]; for(i=step;i<N;i=i+step) tmp=A[i] < tmp ? A[i]:tmp;
このような長い文を理解して, ユーザーのやりたいことが, Fortran の minval 関数と同等であると理解できるコンパイラは, 未だ存在しない. そのような問題点は, すでに知られており, Intel 社の C++ コンパイラにおいては, 配列表記（Array Notation）というものが採用され,
変数=__sec_reduce_min(配列名 [開始位置 : 要素数 : ストライド]);
という記法が存在する. 配列表記（Array Notation）は、インテル C++ コンパイラのバージョン 12 でサポートされた新しい言語拡張であり, これらの機能を利用した場合, 他のコンパイラではコンパイルできないという問題点もある. そこで,
(1)minval 関数を含む Fortran のソースコードを以下のようにコンパイルする,
gfortran -O3 -mtune=native -march=native -c program_min.f,
そして, program_min.o というオブジェクトコードを作成する.
(2)gcc -O3 -mtune=native -march=native -o test test.c program_min.o

*kkimur@amp.i.kyoto-u.ac.jp

として, Fortran コンパイラが作成したオブジェクトコードを C 言語から呼び出すという解決策が考えられる.

1.2 C 言語における 2 次元配列の扱いについて

C99 のみであるが, 以下のような記述を用いて, 2 次元配列を確保, ならびに, 利用することをお勧めする.

```
double (* restrict A)[N];
A=(double (* restrict)[N])malloc(N*N*sizeof(double));
```

これ以降, A[i][j] としてアクセスすることが可能である. Fortran には, allocate 文があり, 上記のような高度な知識を要求する記述は必要ない. restrict とは, そのポインタが指している領域は, そのポインタにしか指されていないことを確約するキーワードであり, コンパイラにおける最適化の際に, より効率的な実行コードが作成される. Fortran の allocate 文で確保される領域は, 必然的に上記の性質を満たし, 最適化を容易にしている.

1.3 C 言語における有限体の総和演算の実装法について

$p, a, b \in \mathbb{Z}, p < 2^{63}, 0 \le a, b < p, a \oplus b = (a + b) \bmod p$ と定義する. $0 \le a_0, \cdots, a_{99999999} < p$ について, 総和 $t = a_0 \oplus a_1 \oplus \cdots \oplus a_{99999999}$ を考える. $t = a_0 \oplus a_1 \oplus \cdots \oplus a_{99999999} = (a_0 + a_1 + \cdots + a_{99999999}) \bmod p$ を利用して, 以下の実装を行う.

```
typedef unsigned int uint128_t __attribute__((mode(TI)));
unsigned long long a[1000000],t; uint128_t r=0;
for(i=0;i<1000000;i++) r+=a[i];
t=r % p;
```

上記のように,

```
typedef unsigned int uint128_t __attribute__((mode(TI)));
```

と書くと, GNU gcc コンパイラでは, 128bit の符号なし整数が使えるようになる.

2 最小多項式の候補について

与えられた行列 A に対して, $f(A) = 0$ となる次数が最小の多項式 $f(x) = x^k + c_{k-1}x^{k-1} + \cdots + c_0$ を, 最小多項式という. あるベクトル v_0 について, $g(A)v_0 = 0$ となる次数が最小の多項式 $g(x) = x^k + d_{k-1}x^{k-1} + \cdots + d_0$ を, 最小消去多項式という. さらに, あるベクトル u_0, v_0 について, $u_0^\top h(A)v_0 = 0$ となる次数が最小の多項式 $h(x) = x^k + e_{k-1}x^{k-1} + \cdots + e_0$ を考える場合がある. $h(x)|g(x)|f(x)$ の関係がある. u_0, v_0 が generic(固有ベクトルの全成分を含む) ならば, $f(x) = g(x) = h(x)$ が成り立つ. 本稿において, 最小多項式の候補とは $g(x)$ や $h(x)$ のことをさす. $g(x)$ を求める問題は, $A^i v_0$ がベクトルであるから, ベクトル列の従属性を判定する問題とほぼ等価である. 解法としては, (i) クリロフ部分空間法から生成される連立一次方程式を Hensel 構成で解く方法, (ii)Arnoldi 法の直交行列の生成部分を下三角行列で代用する方法の 2 種類が考えられる. $h(x)$ を求める問題は, $s_i = u_0^\top A^i w_0$ とすると, 以下の Hankel 行列の連立一

次方程式を解き,

$$\begin{pmatrix} s_0 & s_1 & \cdots & s_{k-1} \\ s_1 & s_2 & \cdots & s_k \\ \vdots & \vdots & \vdots & \vdots \\ s_{k-1} & s_k & \cdots & s_{2k-2} \end{pmatrix} \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{k-1} \end{pmatrix} = - \begin{pmatrix} s_k \\ s_{k+1} \\ \vdots \\ s_{2k-1} \end{pmatrix},$$

$s_{2k} + e_{k-1}s_{2k-1} + \cdots + e_0 s_k = 0$ を確認する問題と等価である. 解法としては, (iii)Wiedemann algorithm における最小多項式の構成法がある. また, $g(x)$ を求める方法と $h(x)$ を求める方法の両方において, (iv) ブロッククリロフ部分空間法と Wiedemann algorithm を用いる方法が考えられる. しかし, そのアルゴリズムは行列乗算を必要とするため, 本稿では省略する. (i) では Hensel 構成が, (ii)(iii)(iv) は中国剰余定理が重要な役割を果たす.

(i) クリロフ部分空間法から生成される連立一次方程式を Hensel 構成で解く方法では, 有限体 $\mathbb{Z}/p\mathbb{Z}$ 上で一次従属ならば有理数体 $\mathbb{Q}$ 上でも一次従属であるとみなし (本稿は, 最小多項式の候補の計算であるから), v_0 を乱数ベクトルとして, $\mathbb{Q}$ 上において連立一次方程式,

$$\left(\begin{array}{c|c|c|c|c} A^{k-1}v_0 & A^{n-2}v_0 & \cdots & Av_0 & v_0 \end{array}\right) \begin{pmatrix} c_{k-1} \\ \vdots \\ c_0 \end{pmatrix} = -A^k v_0,$$

を生成する. それを, Hensel 構成によって解く. v_0 が generic で, p が特殊な素数でないならば真の最小多項式を得る.

(ii)Arnoldi 法の直交行列の生成部分を下三角行列で代用する方法は, $AV = VH$ を満たす V と H を生成する. V は下三角行列であり, H は上 Hessenberg 行列である.

(iii)Wiedemann algorithm では, $k+1$ において, 以下の行列が,

$$\begin{pmatrix} s_0 & s_1 & \cdots & s_k \\ s_1 & s_2 & \cdots & s_{k+1} \\ \vdots & \vdots & \vdots & \vdots \\ s_k & s_{k+1} & \cdots & s_{2k} \end{pmatrix},$$

はじめて特異行列になるならば, k における解が $h(x)$ になる. $k+1$ での計算に k の計算結果を使うことができ, s_i の計算量を除外すると, $h(x)$ を $O(k^2)$ で計算できる.

3 数学的考察からの工夫

3.1 $s_i = u_0^\top A^i w_0$ の計算における工夫

$s_0 = (u_0^\top)(w_0) = (u_0)^\top(w_0)$, $s_1 = (u_0^\top)(Aw_0) = (u_0)^\top(Aw_0)$, $s_2 = (u_0^\top A)(Aw_0) = (A^\top u_0)^\top(Aw_0)$, $s_3 = (u_0^\top A)(A^2 w_0) = (A^\top u_0)^\top(A^2 w_0)$, $s_4 = (u_0^\top A^2)(A^2 w_0) = ((A^\top)^2 u_0)^\top(A^2 w_0)$, $\cdots$. このとき, 次の 2 系列を考える. $w_0, Aw_0, A^2 w_0, \cdots$, $u_0, A^\top u_0, (A^\top)^2 u_0, \cdots$. この 2 系列は, 互いに無関係であり, 平行計算できる, さらに, Ax, $A^\top y$ は, 次の『Wilkinson の技巧』を用いて合理的に計算できる.

```
for(j=0;j<MAXmat;j++) ATY_TMP[j]=0;
for (j = 0; j < MAXmat; j++){
    TMP1 = 0;
    for (k = 0; k < MAXmat; k++){
        TMP1 = (uint128_t) A[j][k] * (uint128_t) X[k] + TMP1;
        ATY_TMP[k]=(uint128_t) A[j][k] * (uint128_t) Y[j] + ATY_TMP[k];
    }
    AX[j] = TMP1 % CM;
}
for(j=0;j<MAXmat;j++) ATY[j]=ATY_TMP[j] % CM;
```

4 タイミングデータ

計算機環境として, CPU:Core i7-4770K 3.50GHz, Memory:32GB, OS:Fedota 19, Compiler:gcc 4.8.1, Compiler Option:-O3 -mtune=native -march=native -fopenmp を用いて, 実験を行った. 比較対象として, (i),(ii),(ii) の 3 つの解法の並列実装を, 密行列と疎行列に適用した結果を述べる. 32bit の演算を用いる方法と 64bit の演算を用いる方法の 2 通りを作成した.

4.1 密行列

各要素が 1 以上 10 以下の整数を乱数で生成する. 単位は sec. で表示する.

n	(i)	(ii),32bit	(ii),64bit	(iii),32bit	(iii),64bit
250	0.230	0.075	0.147	0.087	0.268
500	2.515	0.853	1.766	1.100	3.466
750	11.453	5.132	10.685	5.790	16.638
1000	34.248	28.709	41.531	26.330	52.596

この計算機環境では, (iii) のアルゴリズムを 32bit の演算を利用して実装する方法が最良である. Core i7-4770K には, AVX2 が搭載されていることが, その要因と考えられる.

4.2 疎行列

各要素が 1 以上 10 以下の整数を乱数で生成する. しかし, 充填率は 1% であり, 0 が大半を占める行列である. 単位は sec. で表示する.

n	(i)	(ii),32bit	(ii),64bit	(iii),32bit	(iii),64bit
2000	35.419	> 15	> 15	10.119	19.041
3000	155.621	> 50	> 50	48.011	85.213
4000	479.982	> 190	> 190	182.197	270.672
5000	n/a	> 550	> 550	546.642	640.267

n/a は, ここではメモリオーバーを意味する. 疎行列においても, (iii) のアルゴリズムを 32bit の演算を利用して実装する方法が最良となる.