Vol.2021-MPS-134 No.8
2021/7/27

両側ヤコビ法を用いた複素数行列に対する特異値分解の実装法

宮前 隆広[1,a)]　髙田 雅美[2,b)]　木村 欣司[1,c)]　中村 佳正[3,d)]

概要：タイムリバーサル法では, 複素数行列の特異値分解（SVD）が必要である．まず, 時系列データから得られたタイムリバーサル演算子の各要素を周波数領域に変換する．次に, 周波数領域に対して SVD を行う．そのため, SVD の対象となる行列の要素はすべて複素数となる．SVD を行うためのヤコビ法は, すべての特異値と特異ベクトルを高精度に計算することができる．しかし, ヤコビ法の計算コストは, QR 法とハウスホルダー変換による 2 重対角化を組み合わせた計算法よりも高い．ただし, 行列サイズが小さい場合には大差ない．ヤコビ法には, 片側ヤコビ法と両側ヤコビ法がある．このうち, 片側ヤコビ法は, LAPACK に実装されている．本研究では, 複素数行列のための両側ヤコビ法を実装する．小さな行列では, 片側ヤコビ法よりも両側ヤコビ法のほうが計算時間が短く精度が高いという実験結果が得られた．

Implementation of Singular Value Decomposition for Complex Matrices Using the two-sided Jacobi Method

1. はじめに

タイムリバーサル (TR) 法では, 複素数行列の特異値分解 (SVD) が必要である [11]．TR 法では, まず, 時系列データから得られるタイムリバーサル演算子の各要素を周波数領域に変換する．その後, 周波数領域内で分解を行う．そのため, SVD の対象となる行列の要素はすべて複素数となる．

TR 法のために, 高速かつ高精度に計算できる複素数行列のための SVD を開発する必要がある．時系列データは, 小さな行列に置き換えて扱うことが可能である．そのため, 小さな行列を高速かつ高精度に計算できる SVD が適している．小規模行列のための SVD として, 片側ヤコビ法と両側ヤコビ法がある [2], [3], [4], [5], [6], [7], [8], [9]．片側ヤコビ法は, LAPACK [10] のルーチンとして存在する．一方, 複素数行列のための両側ヤコビ法には, 実用的な実装方法が確立されていない．本研究では, 複素数行列に対する両面ヤコビ法の実装方法を提案する．提案する実装方法では, 逆正接関数, Rutishauser の実装方法 [12], Givens 回転をそれぞれ提案する [1]．さらに, 複素数の絶対値の高精度計算, 高精度 Givens 回転, 計算の融合による高速化, SIMD 型計算の利用という実装の工夫についても導入する．実験では, LAPACK の片側ヤコビ法と両側ヤコビ法について, 計算時間および精度を比較する．

2. 両側ヤコビ法による特異値分解

特異値分解のための両側ヤコビ法は, 任意の実行列の特異値と特異ベクトルを計算出来る．正方行列だけでなく長方形行列であっても, 前処理として QR 分解を行うことで上三角行列に変換することができる．そのため計算の対象を上三角行列に限定し, 特異値分解のための両側ヤコビ法を説明する．この解法は, 上三角行列に対して左右からヤコビ行列をかけることで行列を対角化していく．複素数行列に対しても, 任意の複素数行列を両側ヤコビ法の計算対象とすることが可能である．複素数行列の場合も, 実行列と同様に前処理を行うことで複素上三角行列の特異値分解の問題に帰着させる．

[1] 福井大学
University of Fukui, Fukui, Fukui 910–8507, JAPAN
[2] 奈良女子大学
Nara Women's University, Nara, Nara 630–8506, JAPAN
[3] 大阪成蹊大学
Osaka Seikei University, Higashiyodogawa-ku, Osaka 533–0007, JAPAN
[a)] ms200319@u-fukui.ac.jp
[b)] takata@ics.nara-wu.ac.jp
[c)] kkimur@u-fukui.ac.jp
[d)] nakamura-yo@osaka-seikei.ac.jp

$$
\begin{pmatrix} I & 0 & \cdots & \cdots & 0 \\ 0 & c_1 & \cdots & s_1 & \vdots \\ \vdots & \vdots & I & \vdots & \vdots \\ \vdots & -s_1 & \cdots & c_1 & 0 \\ 0 & \cdots & \cdots & 0 & I \end{pmatrix}
\times \begin{pmatrix} \ddots & \cdots & \cdots & \cdots & \cdots \\ \vdots & a_{j,j} & \cdots & a_{j,k} & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & 0 & \cdots & a_{k,k} & \vdots \\ \cdots & \cdots & \cdots & \cdots & \ddots \end{pmatrix}
\times \begin{pmatrix} I & 0 & \cdots & \cdots & 0 \\ 0 & c_2 & \cdots & -s_2 & \vdots \\ \vdots & \vdots & I & \vdots & \vdots \\ \vdots & s_2 & \cdots & c_2 & 0 \\ 0 & \cdots & \cdots & 0 & I \end{pmatrix}
= \begin{pmatrix} \ddots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \hat{a}_{j,j} & \cdots & 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & 0 & \cdots & \hat{a}_{k,k} & \vdots \\ \cdots & \cdots & \cdots & \cdots & \ddots \end{pmatrix} \tag{1}
$$

3. 複素数行列に対する実装法

複素上三角行列に対する両側ヤコビ法は, ヤコビ回転行列を作用させる前に前処理が必要になる．前処理として, 行列の対角成分を実数に変換する方法と, ヤコビ回転行列が作用する非対角成分を実数に変換する方法の 2 種類が必要となる．

3.1 行列の対角成分

以下の複素上三角行列に対して, 対角成分を実数に変換する．

$$
\boldsymbol{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & \cdots & a_{1,n} \\ & a_{2,2} & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & \vdots \\ & & & & a_{n,n} \end{pmatrix}
$$

但し, $a_{j,k} \in \mathbb{C}$ である．
行列 $\boldsymbol{A}$ に対して, 対角成分 (j,j) が $\overline{a_{j,j}}/|a_{j,j}|$, 非対角成分が 0 の $n \times n$ 行列 $\boldsymbol{T}$ を右からかけることにより, 対角成分を絶対値の大きさを変えることなく実数化する．

$$
\boldsymbol{T} = \mathrm{diag.}\left(\frac{\overline{a_{1,1}}}{|a_{1,1}|}, \frac{\overline{a_{2,2}}}{|a_{2,2}|}, \cdots, \frac{\overline{a_{n,n}}}{|a_{n,n}|} \right),
$$

$$
\boldsymbol{AT} = \begin{pmatrix} |a_{1,1}| & \hat{a}_{1,2} & \cdots & \cdots & \hat{a}_{1,n} \\ & |a_{2,2}| & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & \hat{a}_{n-1,n} \\ & & & & |a_{n,n}| \end{pmatrix}
$$

3.2 行列の非対角成分

以下の 2×2 の行列 $\boldsymbol{A}$ に対して, 非対角成分を実数に変換する．

$$
\boldsymbol{A} = \begin{pmatrix} r_{j,j} & r_{j,k} \\ 0 & r_{k,k} \end{pmatrix}, r_{j,j}, r_{k,k} \in \mathbb{R}, r_{j,k} \in \mathbb{C}
$$

この行列に対して, 2 つの行列 $\boldsymbol{U}, \boldsymbol{V}$ を用いる．

$$
\boldsymbol{U} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{r_{j,k}}{|r_{j,k}|} \end{pmatrix}, \boldsymbol{V} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{\overline{r}_{j,k}}{|r_{j,k}|} \end{pmatrix}
$$

行列 $\boldsymbol{A}$ に対して, 左右から $\boldsymbol{U}, \boldsymbol{V}$ をそれぞれかけることで, 絶対値の大きさを変化させずに実数化できる．

$$
\tilde{\boldsymbol{A}} \leftarrow \boldsymbol{UAV} = \begin{pmatrix} r_{j,j} & |r_{j,k}| \\ 0 & r_{k,k} \end{pmatrix}
$$

4. 計算の高精度化・高速化

4.1 複素数の絶対値の高精度計算

3.1, 3.2 では $u \in \mathbb{C}$ に対して次の計算を行っている．

$$
t2 = \frac{u}{|u|}, \quad t3 = \frac{\overline{u}}{|u|}
$$

この計算をプログラム上でに次のように実装すると, 高精度な計算を達成出来ない場合がある．

```
t1 = ABS(u)
t2 = u/t1
t3 = CONJG(t2)
```

ABS,CONJG は FORTRAN によって実装されている関数でそれぞれ絶対値と複素共役を計算する．この実装は行列サイズが大きくなると大きな計算誤差を生んでしまう．そこで次の式を用いて計算を行う．

$$
\begin{aligned}
& u = a + ib \\
& \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix} \qquad (2) \\
& r = \sqrt{a^2 + b^2}\ ,\ \cos\theta = \frac{a}{r}\ ,\ \sin\theta = \frac{b}{r} \\
& \frac{u}{|u|} = \cos\theta + i\sin\theta, \quad \frac{\overline{u}}{|u|} = \cos\theta - i\sin\theta
\end{aligned}
$$

式 (2) は Givens 回転の式と完全に一致し, $\cos\theta, \sin\theta$ により式 (4.1) の実部, 虚部それぞれを求めることが出来る．こ

の値を高精度に計算可能なルーチンが LAPACK により提供されており, 式 (4.1) の計算は, 次のように行う.

```
t4 = REAL(u)
t5 = AIMAG(u)
CALL SLARTG(ABS(t4) , ABS(t5) , c , s , t1)
c = SIGN(c , t4)
s = SIGN(s , t5)
t2 = CMPLX(c , s)
t3 = CMPLX(c , -s)
```

REAL, AIMAG は, それぞれ実部、虚部を取り出し, CMPLX は 2 つの数から複素数を生成している. SLARTG により $\cos\theta, \sin\theta$ の値を計算できる. この実装により高精度な計算を達成出来る.

4.2 高精度 Givens 回転

Givens 回転では, s_1, s_2, c_1, c_2 の値が必要である. この値は理論的には $s^2 + c^2 = 1$ を満たす. しかし数値計算では丸め誤差等の影響によって, この式を満たさない場合がある. そこで $\hat{s}, \hat{c}$ を補正後の値として, 割線法により値を修正する.

(1) $\frac{\pi}{4} \geq \theta \geq -\frac{\pi}{4}$ すなわち $c \geq 0$ かつ $c \geq |s|$ の場合
$x_0 = 1, x_1 = c$ として, $f(x) = x^2 + s^2 - 1$ に対する割線法

$$x_{n+1} = \frac{x_{n-1} \times f(x_n) - x_n \times f(x_{n-1})}{f(x_n) - f(x_{n-1})} \qquad (3)$$

によって $x_2 = \hat{c} = 1 - s \times (s/(1+c))$ を得る.
$\hat{c}$ を用いると, ベクトル x, y に対するヤコビ回転行列の 2 本のベクトルへの作用 $x \leftarrow \hat{c}x + sy, y \leftarrow -sx + \hat{c}y$ は, 次のように修正される.

$$w = \frac{s}{1+c}$$
$$x \leftarrow s \times (-w \times x + y) + x$$
$$y \leftarrow -s \times (w \times y + x) + y$$

(2) $\frac{3\pi}{4} \geq \theta > \frac{\pi}{4}$ すなわち $s > 0$ かつ $s \geq |c|$ の場合
$x_0 = 1, x_1 = s$ として, $f(x) = x^2 + c^2 - 1$ に対する割線法によって式 (3) より $x_2 = \hat{s} = 1 - c \times (c/(1+s))$ を得る.
$\hat{s}$ を用いると, ベクトル x, y に対するヤコビ回転行列の 2 本のベクトルへの作用 $x \leftarrow cx + \hat{s}y, y \leftarrow -\hat{s}x + cy$ は. 次のように修正される.

$$w = \frac{c}{1+s}$$
$$x \leftarrow c \times (-w \times y + x) + y$$
$$y \leftarrow c \times (w \times x + y) + x$$

(3) $\frac{5\pi}{4} > \theta > \frac{3\pi}{4}$ すなわち $c < 0$ かつ $|c| > |s|$ の場合
$x_0 = -1, x_1 = c$ として, $f(x) = x^2 + s^2 - 1$ に対する割線法によって式 (3) より $x_2 = \hat{c} = -1 + s \times (s/(1-c))$ を得る.
$\hat{c}$ を用いると, ベクトル x, y に対するヤコビ回転行列の 2 本のベクトルへの作用 $x \leftarrow \hat{c}x + sy, y \leftarrow -sx + \hat{c}y$ は, 次のように修正される.

$$w = \frac{s}{1-c}$$
$$x \leftarrow s \times (w \times x + y) - x$$
$$y \leftarrow s \times (w \times y - x) - y$$

但し, 両側ヤコビ法による固有値分解・特異値分解においてこの計算が現れることはない.

(4) $-\frac{\pi}{4} > \theta \geq -\frac{3\pi}{4}$ すなわち $s < 0$ かつ $|s| \geq |c|$ の場合
$x_0 = -1, x_1 = s$ として, $f(x) = x^2 + c^2 - 1$ に対する割線法によって式 (3) より $x_2 = \hat{s} = -1 + c \times (c/(1-s))$ を得る.
$\hat{s}$ を用いると, ベクトル x, y に対するヤコビ回転行列の 2 本のベクトルへの作用 $x \leftarrow cx + \hat{s}y, y \leftarrow -\hat{s}x + cy$ は, 次のように修正される.

$$w = \frac{c}{1-s}$$
$$x \leftarrow c \times (w \times y + x) - y$$
$$y \leftarrow c \times (-w \times x + y) + x$$

4.3 計算の融合による高速化

3.1, 3.2 の実数化を行う計算は, 4.2 のヤコビ回転行列の計算と融合することが出来る. 計算を融合することは, メモリから CPU へのデータの移動量を減らすことになり, 計算速度を高速化することができる.

4.4 SIMD 型計算の利用

SIMD(Single Instruction Multiple Date) 型計算は, 複数の同一の演算を一つの命令で行うものである. 例えば, 下の 8 次元ベクトルの演算を実行する場合, 8 つの成分を同時に処理する.

$$\text{一つの命令} \rightarrow \begin{cases} d_1 \leftarrow a_1 \times b_1 + c_1 \\ d_2 \leftarrow a_2 \times b_2 + c_2 \\ d_3 \leftarrow a_3 \times b_3 + c_3 \\ d_4 \leftarrow a_4 \times b_4 + c_4 \\ d_5 \leftarrow a_5 \times b_5 + c_5 \\ d_6 \leftarrow a_6 \times b_6 + c_6 \\ d_7 \leftarrow a_7 \times b_7 + c_7 \\ d_8 \leftarrow a_8 \times b_8 + c_8 \end{cases}$$

この演算を使うことで計算を高速化することが可能になる. しかし, 値が複素数のとき, 以下のような従来の実装法では

GNU のコンパイラが SIMD 型命令を, 機械語列内に生成することは, 現状ではできていない.

```
DO I=1,N
  U=X(I)
  V=Y(I)*T
  X(I)=S*(-F*U+V)+U
  Y(I)=-S*(F*V+U)+V
ENDDO
```

そこで, 次のようにソースコードを改良した.

```
DO I=1,N
  UR=REAL(X(I))
  UC=AIMAG(X(I))
  V=Y(I)*T
  VR=REAL(V)
  VC=AIMAG(V)
  X(I)=CMPLX(S*(-F*UR+VR)+UR,S*(-F*UC+VC)+UC)
  Y(I)=CMPLX(-S*(F*VR+UR)+VR,-S*(F*VC+UC)+VC)
ENDDO
```

この実装法では, 複素数の値であってもコンパイラが SIMD 型命令機械語列内に生成し, 高速な計算が可能になる.

5. 実験結果

我々が実装した Givens 回転を用いた実装を採用した両側ヤコビ法と Lapack-3.9.1 の片側ヤコビ法について比較実験を行う. 数値実験に利用する入力行列は, 全要素が乱数の上三角行列と全要素が $1+i$ の上三角行列である.

実験の結果, 我々の両側ヤコビ法は, 計算速度はどちらの入力行列とも 1000×1000 行列までは Lapack の片側ヤコビ法よりも高速な計算を達成出来る. すなわち小さい行列に特化した特異値分解であることが分かる. また 1000×1000 行列よりも大きい行列では片側ヤコビ法よりも計算速度は遅くなっている. これはキャッシュメモリを使いきってしまっているからと考えられる. 今回の実験環境での計算速度のターニングポイントとなる行列サイズは 1000 である. すなわちキャッシュメモリの容量により高速な計算が達成出来る行列サイズが変化するため, よりキャッシュメモリの大きい環境を用いることにより大きい行列に対しても高速化を行える可能性がある.

計算精度については, すべての行列サイズにおいて片側ヤコビ法よりも計算精度を向上できている. 左右特異ベクトルの直交性もすべての行列サイズにおいて片側ヤコビ法よりも精度を向上できている. このことから Givens 回転による両側ヤコビ法の実装は, 片側ヤコビ法よりも高精度な計算法といえる. 特に左特異ベクトルの直交性は片側ヤコビ法に比べて格段に向上している.

6. まとめ

本稿では, 複素数行列のための両側ヤコビ法の実装方法を提案している. 提案した実装では, 高精度で計算融合が可能な Givens 回転を採用してる. 実験では, LAPACK にある複素数行列のための片側ヤコビ法との比較を行っている. 実験の結果, 小規模の複素数行列に対して高速かつ高精度な SVD の開発に成功している. TR 法の行列サイズが小規模であることを考慮すると, TR 法の実装においてその応用が期待される. さらに, 両側ヤコビ法は, 左得意ベクトルの直交性が良好であるため, その点においても, TR 法の改良への応用が期待される.

謝辞 本研究は JSPS 科研費 JP17H02858 の助成を受けたものです.

参考文献

[1] Araki, S., Aoki, M., Takata, M., Kimura, K., Nakamura, Y., ; *Implementation of Two-sided Jacobi Method*, IPSJ Trans. Modeling Appl., Vol. 14, No. 1, pp. 12–20 (2021).
[2] Brent, R. P., Luk, F. T., and van Loan, C.: *Computation of the singular value decomposition using mesh-connected processors*, J. VLSI Comput. Syst., Vol. 1, pp. 242–270 (1985)
[3] Drmac, Z. and Veselic, K.: *New fast and accurate Jacobi SVD algorithm: I.*, SIAM J. Matrix Anal. Appl., Vol. 29, pp. 1322–1342 (2008)
[4] Drmac, Z. and Veselic, K.: *New fast and accurate Jacobi SVD algorithm: II.*, SIAM J. Matrix Anal. Appl., Vol. 29, pp. 1343–1362 (2008)
[5] Drmac, Z.,; *Computing singular and generalized singular values* , Ph.D. University of Hagen, Germany (1994).
[6] Drmac, Z.,; *A posteriori computation of singular vectors in a preconditioned Jacobi SVD algorithm*, IMA J. Numer. Anal. 19 pp. 191–213 (1999).
[7] Forsythe, G. E. and Henrici, P.: *The cyclic Jacobi method for computing the principal values of a complex matrix*, Trans. Amer. Math. Soc., Vol. 94, pp. 1–23 (1960)
[8] Hari, V., and Zadelj-Martić V.,; *Parallelizing the Kogbetliantz method: A first attempt*, Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM), Vol. 2, No. 1–2, pp. 49–66 (2007).
[9] Kogbetliantz, E.: *Solution of linear equations by diagonalization of coefficients matrix*, Quarterly of Applied Mathematics, Vol. 13, pp. 123–132 (1955)
[10] Linear Algebra PACKage, 入手先 ⟨http://www.netlib.org/lapack/⟩, (2021.05.31).
[11] Odedo, V., Yavuz, M.E., Costen, F., Himeno, R., Yokota, H.,; *Time reversal technique based on spatiotemporal windows for through wall imaging*, IEEE Transactions on Antennas and Propagation, Vol. 65, No. 6, pp. 3065-3072 (2017).
[12] Rutishauser, H.: *The Jacobi method for real symmetric matrices*, Numerische Mathematik, Vol. 9, No. 1, pp. 1–10 (1966)