Vol.2016-MPS-109 No.2
2016/7/25

ARPACKを用いた大規模行列の部分特異値分解のための実装方法の提案

髙田 雅美[1,a)]　荒木 翔[2,b)]　木村 欣司[2,c)]　藤井 祐貴[2,d)]　中村 佳正[2,e)]

概要：本稿では，大規模行列の部分特異値分解のための新たな実装方法を提案する．提案方法では，実行時間と共有メモリを用いたマルチコアプロセッサのキャッシュの活用に着目している．特異値分解は，固有値問題としてとらえることができる．大きい方，もしくは，小さい方からいくつかの固有値とそれに対応する固有ベクトルを求めたい場合，ARPACK(ARnoldi PACKage) を用いることが効果的である．ARPACKを用いて特異値分解を行うためには，各反復において，行列とベクトルの乗算を 2 回行う必要がある．この際，行列サイズが大きすぎる場合，キャッシュオーバーフローが生じる．この問題を回避するために，提案方法では，対象とする行列の行に属する要素を再利用することによって，キャッシュヒット率を向上させている．提案方法の有効性を確認するために数値実験を行う．数値実験の結果より，提案方法の方が，従来の実装方法よりも約 80% の実行時間で計算を行えることがわかった．

Implementation of Computing Singular Pairs for Large Scale Matrices using ARPACK

Masami Takata[1,a)]　Sho Araki[2,b)]　Kinji Kimura[2,c)]　Yuki Fujii[2,d)]　Yoshimasa Nakamura[2,e)]

1. はじめに

統計解析において，分析対象の特徴を得るために，主成分分析が用いられている．この主成分分析では，大規模な行列の特異値分解が必要となる．これらの行列は，密行列や疎行列の場合がある．統計解析で必要とする特異値および特異ベクトルの対は，いくつかの大きい特異値に対応するものだけである．特異値分解は，対象とする行列にその転置行列を乗算することによって，固有値問題として捉えることができる．

いくつかの大きい固有値とそれに対応する固有ベクトルを求めたい場合，ARPACK (ARnoldi PACKage) [1] に含まれている IRA(Implicitly Restarted Arnoldi) アルゴリズム [2] と IRL(Implicitly Restarted lanczos) アルゴリズム [3] を用いることによって計算することができる．ARPACK は，大規模行列を対象とするパッケージである．Krylov 部分空間を用いたアルゴリズムである IRA アルゴリズムと IRL アルゴリズムは，部分的な固有値問題を解くことができることで有名である．これらのアルゴリズムは，Krylov 部分空間の行列サイズを制限することによって，計算時間を短縮させることができる．

ARPACK では，対象とする行列に対応したベクトルが与えられる．ユーザは，この与えられたベクトルと対象とする行列の内積を計算する．一般的に，求めたい固有値と固有ベクトルの対の数の 10 倍の数の結果を ARPACK に与えることによって，部分的に固有値問題を解くことができる．特異値問題に対して ARPACK を利用する場合，行列とベクトルに対する乗算が，各反復において 2 回必要である．この実装方法は，計算時間や共有メモリ型マルチコアプロセッサにおけるキャッシュ利用率を考慮していな

[1] 奈良女子大学
Nara Women's University, Nara, Nara 630–8506, JAPAN
[2] 京都大学
Kyoto University, Kyoto, Kyoto 606–8501, JAPAN
[a)] takata@ics.nara-wu.ac.jp
[b)] araki@amp.i.kyoto-u.ac.jp
[c)] kimura.kinji.7z@kyoto-u.ac.jp
[d)] fujii@amp.i.kyoto-u.ac.jp
[e)] ynaka@i.kyoto-u.ac.jp

い．そのため，行列サイズが大きくなると，キャッシュのオーバーフローが生じ，計算時間が長くなる．この問題を解決するために，本稿では，新たな実装方法を提案する．この提案方法は，OpenMP を用いて実装することによって，さらに高い性能を得ることが期待される．

2 章では，IRA アルゴリズムと IRL アルゴリズムについて紹介する．3 章では，ARPACK を用いた特異値分解の新たな実装方法を提案する．4 章では，大きいキャッシュを持つマルチコアプロセッサを用いて，提案方法の性能を評価する．

2. IRA アルゴリズムと IRL アルゴリズム

本章では，文献 [2], [3] を基に，IRA アルゴリズムと IRL アルゴリズムについて説明する．求めたい固有値の数を ℓ とする．IRA アルゴリズムと IRL アルゴリズムでは，各反復において，近似行列が得られるまで， 新しい基底となるベクトルを生成し続けることによって，Krylov 部分空間を拡大する．これらのアルゴリズムは，多くのメモリと計算時間を必要とするため，再直交化のコストは増加する．Krylov 部分空間のサイズを m ($\ell < m \ll n$) に限定することによって，これらのアルゴリズムの再直交化に関するコストを削減することができる．なお，IRA アルゴリズムと IRL アルゴリズムは，どちらも，ARPACK [1] に実装されている．

2.1 Implicitly shifted QR steps

IRA アルゴリズムと IRL アルゴリズムには，implicit QR step が使われている．これは，explicit QR step を改良したものである．

QR step は，次のような式に基づいて $\tilde{H}_m^{(i)} \in \mathbb{R}^{m\times m}$ を更新するアルゴリズムである．

$$\tilde{H}_m^{(i)} = \tilde{Q}_i \tilde{R}_i \tag{1}$$
$$\tilde{H}_m^{(i+1)} = \tilde{R}_i \tilde{Q}_i. \tag{2}$$

初期行列 $H_m^{(1)} \in \mathbb{R}^{m\times m}$, $\tilde{H}_m^{(i)}$ から開始し，i 番目の反復で終了する．この際，次のような式が得られる．

$$\tilde{H}_m^{(i)} = \tilde{Q}_{i-1}^\top \cdots \tilde{Q}_2^\top \tilde{Q}_1^\top H_m^{(1)} \tilde{Q}_1 \tilde{Q}_2 \cdots \tilde{Q}_{i-1} \tag{3}$$
$$= \tilde{Q}^\top \tilde{H}_m^{(1)} \tilde{Q} \quad (\tilde{Q} := \tilde{Q}_1 \tilde{Q}_2 \cdots \tilde{Q}_{i-1}). \tag{4}$$

一方，implicit QR step では，シフト $\mu_i \in \mathbb{R}$ が導入されることにより，

$$\tilde{\tilde{H}}_m^{(i)} - \mu_i I = \tilde{\tilde{Q}}_i \tilde{\tilde{R}}_i \tag{5}$$
$$\tilde{\tilde{H}}_m^{(i+1)} = \tilde{\tilde{R}}_i \tilde{\tilde{Q}}_i + \mu_i I. \tag{6}$$

となる．この場合，最終的に，$\tilde{\tilde{H}}_m^{(i)} \in \mathbb{R}^{m\times m}$ を得ることができる．つまり，以下の式が成り立つ．

$$\tilde{\tilde{H}}_m^{(i)} = \tilde{\tilde{Q}}_{i-1}^\top \cdots \tilde{\tilde{Q}}_2^\top \tilde{\tilde{Q}}_1^\top H_m^{(1)} \tilde{\tilde{Q}}_1 \tilde{\tilde{Q}}_2 \cdots \tilde{\tilde{Q}}_{i-1} \tag{7}$$
$$= \tilde{\tilde{Q}}^\top H_m^{(1)} \tilde{\tilde{Q}} \quad (\tilde{\tilde{Q}} := \tilde{\tilde{Q}}_1 \tilde{\tilde{Q}}_2 \cdots \tilde{\tilde{Q}}_{i-1}) \tag{8}$$

そのため，implicit QR step は次のように考えることができる．まず，μ_i が $\tilde{\tilde{H}}_m^{(i)}$ の固有値である場合を考える．最初の行列 $\tilde{\tilde{H}}_m^{(1)}$ を上 Hessenberg 行列とする．$\tilde{\tilde{R}}_1 \in \mathbb{R}^{m\times m}$ は，$\{\tilde{\tilde{R}}_1\}_{n,n} = 0$ を持つ次のような上三角行列である．

$$\tilde{\tilde{H}}_m^{(1)} = \begin{bmatrix} * & * & \cdots & \cdots & * \\ * & * & \cdots & \cdots & * \\ 0 & * & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * & * \end{bmatrix}, \tag{9}$$

$$\tilde{\tilde{R}}_1 = \begin{bmatrix} * & * & \cdots & \cdots & * \\ 0 & * & \cdots & \cdots & * \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix}. \tag{10}$$

ゆえに，$\tilde{\tilde{R}}_1 \tilde{\tilde{Q}}_1$ は，$\{\tilde{\tilde{R}}_1 \tilde{\tilde{Q}}_1\}_{n,n-1} = 0$ と $\{\tilde{\tilde{R}}_1 \tilde{\tilde{Q}}_1\}_{n,n} = 0$ を持つ上 Hessenberg 行列となる．

$$\tilde{\tilde{R}}_1 \tilde{\tilde{Q}}_1 = \begin{bmatrix} * & * & \cdots & \cdots & \cdots & * \\ * & * & \cdots & \cdots & \cdots & * \\ 0 & * & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * & * \\ 0 & \cdots & \cdots & 0 & 0 & 0 \end{bmatrix}. \tag{11}$$

$$\tilde{\tilde{H}}_m^{(2)} = \begin{bmatrix} * & * & \cdots & \cdots & \cdots & * \\ * & * & \cdots & \cdots & \cdots & * \\ 0 & * & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & * & * & * \\ 0 & \cdots & \cdots & 0 & 0 & \mu_1 \end{bmatrix}. \tag{12}$$

これらの計算を m 回繰り返すことによって，$\tilde{\tilde{H}}_m^{(m)}$ の対角成分が，$H_m^{(1)}$ の固有値に収束することができる．

2.2 Implicit restarting

Arnoldi 法の反復の m 番目を計算し，Arnoldi 法 のベクトル $\mathbf{v}_1, \ldots, \mathbf{v}_m$ から implicitly shifted QR step を実行することによって得られた新しい初期ベクトル $\mathbf{v}^+ \in \mathbb{R}^n$ を用いて反復を再開する．本節では，このベクトル $\mathbf{v}^+$ の計算方法を説明する．

Arnoldi 法の反復の m 番目から，次の関係が得られる．

$$AV_m = V_m H_m + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top. \tag{13}$$

次に，H_m のすべての固有値 $\lambda_1,\ldots,\lambda_m$ を求め，A の固有値の近似値である $\lambda_1,\ldots,\lambda_\ell$ と $\lambda_{\ell+1},\lambda_{\ell+2},\ldots,\lambda_m$ の 2 つに分割する．それから，H_m を初期行列として, $m-\ell$ 回の implicitly shifted QR step を適用し, 最後に, H_m^+ を得る．各反復のシフト量として, $\lambda_m,\ldots,\lambda_{\ell+2},\lambda_{\ell+1}$ を採用する，

$$\mu_1 = \lambda_m,\ldots,\mu_{m-\ell-1} = \lambda_{\ell+2},\mu_{m-\ell} = \lambda_{\ell+1}. \tag{14}$$

H_m と H_m^+ の関係は，次のように表される．

$$Q^+ := Q_1 Q_2 \cdots Q_{m-\ell}, \tag{15}$$

$$V_m^+ := V_m Q^+, \tag{16}$$

$$H_m^+ := (Q^+)^\top H_m Q^+, \tag{17}$$

$$H_m^+ = \begin{bmatrix} * & \cdots & & & & & \cdots & * \\ * & & & & & & & \vdots \\ 0 & \ddots & & & & & & \\ \vdots & \ddots & \ddots & \ddots & & & & \\ & & & * & * & & & \\ & & & & 0 & \mu_{m-\ell} & & \\ & & & & & \ddots & \ddots & \vdots \\ \vdots & & & & & \ddots & \ddots & \mu_2 \quad * \\ 0 & \cdots & & & & \cdots & 0 & 0 \quad \mu_1 \end{bmatrix}. \tag{18}$$

式 (13) から，

$$AV_m Q^+ = V_m H_m Q^+ + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q^+ \tag{19}$$

$$= V_m Q^+ (Q^+)^\top H_m Q^+ + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q^+ \tag{20}$$

$$= V_m Q^+ H_m^+ + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q^+. \tag{21}$$

以上より, 次の式を得る;

$$AV_m^+ = V_m Q^+ H_m^+ + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top Q^+. \tag{22}$$

式 (16),(17) より，1 番目から ℓ 番目までの基底についての関係式が, (22) より得られる;

$$AV_m^+(:,1:\ell) = V_m^+ H_m^+(:,1:\ell) + \mathbf{v}_m\mathbf{e}_m^\top Q^+(:,1:\ell) \tag{23}$$

$$= V_m^+(:,1:\ell)H_m^+(1:\ell,1:\ell) + h_{\ell+1,\ell}^+\mathbf{v}_{\ell+1}^+\mathbf{e}_\ell^\top + q_{m,\ell}\mathbf{v}_m\mathbf{e}_\ell^\top \tag{24}$$

$$= V_m^+(:,1:\ell)H_m^+(1:\ell,1:\ell) + \mathbf{v}_\ell^+\mathbf{e}_\ell^\top, \tag{25}$$

ここで， $\mathbf{v}_\ell^+ := h_{\ell+1,\ell}^+\mathbf{v}_{\ell+1}^+ + q_{m,\ell}\mathbf{v}_m$ とする．ゆえに，初期ベクトル $\mathbf{v}^+$ と式 (25) で Arnoldi 法を再開することができる．

Algorithm 1 IRA アルゴリズム

1: Set m: an upper limit and set ℓ: the number of the desired eigenpairs
2: Input: Arnoldi decomposition $AV_m = V_m H_m + h_{m+1,m}\mathbf{v}_{m+1}\mathbf{e}_m^\top$;
3: **for** $i := 1,2,\cdots$ **do**
4: Compute all the eigenvalues of H_m: $\lambda_1,\ldots,\lambda_m$;
5: Divide eigenvalues: $\lambda_1,\ldots,\lambda_\ell$ and $\lambda_{\ell+1},\ldots,\lambda_m$;
6: Implicitly shifted QR steps for H_m $m-\ell$ times ($\lambda_{\ell+1},\lambda_{\ell+2},\cdots,\lambda_m$ are shift values);
7: $Q^+ = Q_1 Q_2 \cdots Q_{m-\ell}$;
8: $V_m^+ = V_m Q^+$, $H_m^+ = (Q^+)^\top H_m Q^+$;
9: $\mathbf{v}_{m+1}^+ := \mathbf{v}_{m+1}^+ / h_{m+1,m}$;
10: $V_\ell^+ := V_m^+(:,1:\ell), H_\ell^+ := H_m^+(1:\ell,1:\ell)$;
11: $m-\ell$ step Arnoldi algorithm starting with $AV_\ell^+ = V_\ell^+ H_\ell^+ + h_{\ell+1,\ell}\mathbf{v}_{\ell+1}^+\mathbf{e}_\ell^\top$;
12: **end for**

Algorithm 2 既存方法

1: $\tilde{\mathbf{x}} = A\mathbf{x}$;
2: $\mathbf{r} = A^\top\tilde{\mathbf{x}}$;

Algorithm 1 は，IRA アルゴリズムの疑似コードである．

IRL アルゴリズムは，対称行列を対象とする IRA アルゴリズムの改良版である．

3. ARPACK を用いた特異値分解

$w \times n$ $(w \geq n)$ 次の長方行列 $A^{(r)}$ が与えられたとする．この時，IRL アルゴリズムでは，一般的に，各反復に対して 2 回，行列とベクトルを用いた乗算を行うことによって，${A^{(r)}}^\top A^{(r)}\mathbf{x}$ を得ることができる．具体的には，まず $\tilde{\mathbf{x}} = A^{(r)}\mathbf{x}$ $(\tilde{\mathbf{x}} \in \mathbb{R}^w)$ を計算し，次に $\mathbf{r} = {A^{(r)}}^\top\tilde{\mathbf{x}}$ $(\mathbf{r} \in \mathbb{R}^n)$ を求める．本稿では，この実装方法を，既存の方法と呼ぶ．Algorithm 2 は，既存の実装方法の疑似コードである．

計算時間と共有メモリ型マルチコアプロセッサのキャッシュヒット率を改善するために，我々は，${A^{(r)}}^\top A^{(r)}\mathbf{x}$ を計算する次の実装方法を提案する．

(1) $t = A^{(r)}(i,:)\mathbf{x}$
(2) $\mathbf{r} = \mathbf{r} + t{A^{(r)}}^\top(:,i)$

ここで，$A^{(r)}(i,:)$ $(i: 1 \leq i \leq n) \in \mathbb{R}^n$ は，行列 $A^{(r)}$ の i 番目の行ベクトルである．$A^{(r)}(i,:)$ と ${A^{(r)}}^\top(:,i)$ は，同じデータである．そのため，各行の計算において，キャッシュにデータが保持されることで, データの再利用性が生じる．本稿では，この実装方法のことを，提案方法と呼ぶ．特に，提案方法は，大きなキャッシュを持つ共有メモリ型マルチコアプロセッサにおいて，有効である．アルゴリズム 3 は，提案方法の疑似コードである．

4. 数値実験

本章では，提案方法の性能を評価するために，数値実験

Algorithm 3 提案方法

```
1: r = 0;
2: #omp parallel for private(t) reduction(+:r)
3: for i = 1 to n do
4:     t = ⟨a_i, x⟩ (a_i = A^(r)(i,:));
5:     r = r + t a_i;
6: end for
7: #omp end parallel for
```

表 1 実験環境
Environment (Appro 2548X), Kyoto University

CPU	Intel Xeon E5-4650L @2.6GHz, 32cores (8 cores ×4)
	L3 cache: 20MB × 4
RAM	DDR3-1066 1.5TB, 136.4GB/sec
Compiler	Intel C++/Fortran Compiler 14.0.2
Options	-O3 -xHOST -ipo -no-prec-div -mcmodel=medium
	-shared -intel
Software	Intel Math Kernel Library 11.1.2

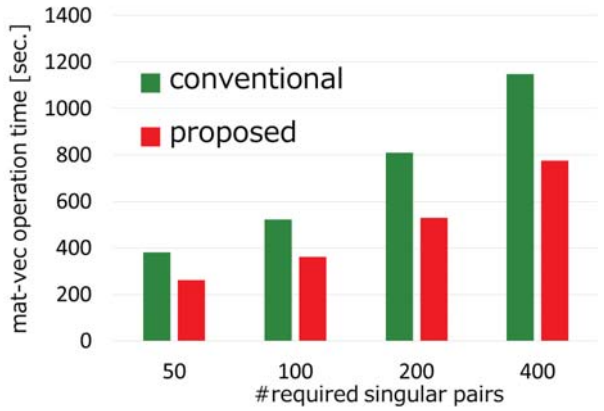

図 1 $A^{(r)\top}A^{(r)}\mathbf{x}$ ($A^{(r)}$: $200,000 \times 100,000$ の実行列) に対する既存方法と提案方法の計算時間

を行う．テスト行列として，200000×100000 の疎行列を用いる．この行列の各行には，1000 個の 0 以外の数値を持つ要素が含まれている．数値実験において，求める特異値と特異ベクトルの対の数を $\ell = 50, 100, 200, 400$ とする．また，特異値は大きい方から順に求めることとする．ARPACK において，Krylov 部分空間 m をユーザが決定する必要がある．そこで，本実験では，$m = 2\ell$ とする．

表 1 は，本実験で用いた計算機のスペックである．

倍精度を用いているため，テスト行列の 1 要素の容量は，64bit である．そのため，テスト行列の全ての各行の要素は，L3 キャッシュよりも小さくなる．ゆえに，$A^{(r)}(i,:)$ のすべてのデータを，キャッシュで保持することができる．また，$A^{(r)}(i,:)$ と $A^{(r)\top}(:,i)$ は同じデータ情報であり，キャッシュを利用することで, データの再利用性が現れる．そのため, 提案方法は, 高い性能を達成できる．

図 1 は，数値実験結果を示す．横軸は求めたい特異値の数を表し，縦軸は $A^{(r)\top}A^{(r)}\mathbf{x}$ の検査時間を表す．図 1 より，提案方法を適用することによって，$A^{(r)\top}A^{(r)}\mathbf{x}$ の計算時間が既存の方法よりも 80% におさえられていることがわかる．

5. まとめ

特異値分解は，固有値問題に変換することが可能である．ARPACK とは，大規模行列の部分固有値問題を解くためのソフトウェアとして有名である．ゆえに，ARPACK を用いることで, 効率的に部分異特値対を求めることができる．

一般的に，ARPACK において，特異対問題は，各反復において行列とベクトルの乗算を 2 回行うことによって計算することができる．大規模な行列を対象とする場合，行列の 0 でない全ての要素のデータをキャッシュが保持することはできない．ゆえに，本稿において，我々は，新たな実装方法を提案する．提案方法では，対象とする行列の各行の全要素がキャッシュに入る場合，データの再利用が可能となる．

提案方法の性能を確認するために，数値実験を行った．この数値実験では，20MB の L3 キャッシュを用いている．また，数値実験に用いたテスト行列は, 各行には，1000 個の 0 以外の数値を持つ要素が含まれているため，L3 キャッシュのサイズよりも小さい．数値実験の結果，提案方法の計算時間は，既存方法の計算時間と比べて，80% に抑えることができることを確認した．

参考文献

[1] Lehoucq, R. B., Sorensen, D. C., and Yang, C.: ARPACK User's Guide: Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods. 入手先 ⟨http://www.caam.rice.edu/software/ARPACK⟩ (1998)

[2] Sorensen, D. C.: *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl., vol.13, pp.357-385, 1992.

[3] Sorensen, D. C., Calvetti, D., and Reichel, L.: *An Implicitly Restarted Lanczos Method for Large Symmetric Eigenvalue Problems*, Elect. Trans. Numer. Anal., vol.2, pp.1-21, 1994.