Vol.2020-HPC-177 No.15
2020/12/22

実対称行列の固有値分解に対する反復改良法の大規模並列環境における実装と評価

内野 佑基[1,a)]　尾崎 克久[2,b)]　荻田 武史[3,c)]

概要：実対称行列の固有値分解に対する反復改良法を大規模並列環境において実装した結果を報告する．現在までに，標準固有値問題を解く様々な数値計算法が開発されている．それらにより得られる近似固有対の精度を改善する荻田・相島による反復改良法が知られている．本研究では，ScaLAPACK のサブルーチンによって得られる近似固有対を，行列積のエラーフリー変換を組み込んだ荻田・相島の反復改良法により改善する方法を実装した．数値実験として，大規模並列環境に名古屋大学情報基盤センターが運営するスーパーコンピュータ「不老」，及び東京大学情報基盤センターと筑波大学計算科学研究センターが共同運営するスーパーコンピュータ「Oakforest-PACS」を使用した結果を紹介する．

1. はじめに

本研究では，実対称行列 $A \in \mathbb{R}^{n\times n}$ に対する固有値分解

$$A = XDX^T$$

を扱う．ここで，A の固有値 $\lambda_i \in \mathbb{R},\ i = 1, 2, \ldots, n$ 及び，λ_i に対応する正規化された固有ベクトル $x_{(i)} \in \mathbb{R}^n$ に対して

$$D = \mathrm{diag}(\lambda) \in \mathbb{R}^{n\times n}, \tag{1}$$

$$X = (x_{(1)} \ \cdots \ x_{(n)}) \in \mathbb{R}^{n\times n} \tag{2}$$

である．(1)，(2) を求める数値計算法として，現在までに QR 法 [1], [2]，分割統治法 [3], [4]，2 分法 [5]，MRRR 法 [6], [7] 等の様々なものが開発されており，特に上記は ScaLAPACK [8] のサブルーチンである固有値ソルバ `p*syev`，`p*syevd`，`p*syevx`，`p*syevr`（`*` は `d` か `s` である）に実装されている．

ある数値計算法によって，正規化された固有ベクトルの近似 $\widehat{x}_{(i)} \approx x_{(i)}$ が得られたとする．このとき，重複固有値が存在しない場合には

$$|\sin(\arccos(|x_{(i)}^T \widehat{x}_{(i)}|))| \le \alpha_i, \tag{3}$$

$$\alpha_i := \mathcal{O}\left(\frac{\|A\|_2 \mathbf{u}}{\min\limits_{\lambda_i \neq \lambda_j} |\widehat{x}_{(i)}^T A \widehat{x}_{(i)} - \lambda_j|}\right) \tag{4}$$

が成り立つ [10]．$\mathbf{u}$ は単位相対丸めを意味する（e.g., binary32 では $\mathbf{u} = 2^{-24}$，binary64 では $\mathbf{u} = 2^{-53}$）．(3)，(4) より，数値計算により得られる近似固有ベクトル $\widehat{x}_{(i)}$ の精度は，対応する固有値と重複を除いて最も近い固有値の差に依存して低下する．そのような固有値間のギャップは，大規模問題では小さくなる場合があるため，固有値分解を高精度に求めるには反復改良法が有用となる．

荻田・相島は，後述する (5) を満たすある程度正確な X の初期値を与えることで，真値への 2 次収束性をもつ反復改良法を提案した [9], [10]．それらは主に高精度行列積で構成されており，(1)，(2) を求める数値計算法に対して多倍長浮動小数点演算を使用するよりも高速に高精度な結果を得られる場合があることが [9], [10] で示されている．また，尾崎らは，エラーフリー変換を利用した高精度な行列積の数値計算法を提案した [11]．本研究では，尾崎らが提案した高精度行列積を組み込んだ荻田・相島の反復改良法を大規模並列環境において実装し，それを使用して近似固有対を改善する．数値実験として，ScaLAPACK のサブルーチン `pssyevd`，`pdsyevd` により得られる近似固有対に対して，本研究にて実装した反復改良法を適用した結果を紹介する．大規模並列環境には，名古屋大学情報基盤センターが運営するスーパーコンピュータシステム「不老」の Type I サブシステムである FUJITSU Supercomputer PRIMEHPC

[1] 芝浦工業大学大学院 システム理工学専攻
Graduate School of Engineering and Science, Shibaura Institute of Technology
[2] 芝浦工業大学 システム理工学部
Department of Mathematical Sciences, Shibaura Institute of Technology
[3] 東京女子大学 現代教養学部
Division of Mathematical Sciences, Tokyo Woman's Christian University
[a)] mf20013@shibaura-it.ac.jp
[b)] ozaki@sic.shibaura-it.ac.jp
[c)] ogita@lab.twcu.ac.jp

Algorithm 1 `RefSyEv` [9]: 6 行目以外は高精度計算を行う．

Require: $A = A^T \in \mathbb{R}^{n\times n},\ \widehat{X} \in \mathbb{R}^{n\times l},\ 1 \le l \le n$
Ensure: $\tilde{X} \in \mathbb{R}^{n\times l},\ \tilde{D}, \tilde{E} \in \mathbb{R}^{l\times l},\ \omega \in \mathbb{R}$
1: **function** $[\tilde{X}, \tilde{D}, \tilde{E}, \omega] \leftarrow$ `RefSyEv`$(A, \widehat{X})$
2: $R \leftarrow I - \widehat{X}^T\widehat{X}$
3: $S \leftarrow \widehat{X}^T A\widehat{X}$
4: $\tilde{\lambda}_i \leftarrow \dfrac{s_{ii}}{1 - r_{ii}}$ (for $1 \le i \le l$)
5: $\tilde{D} \leftarrow$ `diag`$(\tilde{\lambda})$
6: $\omega \leftarrow 2(\|S - \tilde{D}\|_2 + \|A\|_2\|R\|_2)$
7: $\tilde{e}_{ij} \leftarrow \dfrac{s_{ij} + \tilde{\lambda}_j r_{ij}}{\tilde{\lambda}_j - \tilde{\lambda}_i}$ (if $|\tilde{\lambda}_i - \tilde{\lambda}_j| > \omega$ for $1 \le i, j \le l$)
8: $\tilde{e}_{ij} \leftarrow r_{ij}/2$ (if $|\tilde{\lambda}_i - \tilde{\lambda}_j| \le \omega$ for $1 \le i, j \le l$)
9: $\tilde{X} \leftarrow \widehat{X} + \widehat{X}\tilde{E}$
10: **end function**

FX1000 [12]，及び東京大学情報基盤センターと筑波大学計算科学研究センターが共同運営するスーパーコンピュータシステム「Oakforest-PACS [13]」を使用する．

2. 荻田・相島の反復改良法

以降，

$$\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n$$

とする．ある数値計算法により，

$$E \in \mathbb{R}^{n\times n} \text{ s.t. } \|E\|_2 < 1 \tag{5}$$

に対して，

$$X = \widehat{X}(I + E) \tag{6}$$

が成り立つような $\widehat{X} = (\widehat{x}_{(1)}\ \cdots\ \widehat{x}_{(n)}) \approx X$ が得られたと仮定する．ここで，I は単位行列である．

まず，基本となる荻田・相島の反復改良法（[9], Algorithm 1）を Algorithm 1 に示す．これは，E に対する近似行列 $\tilde{E}$ を求め，(6) のように

$$\tilde{X} \leftarrow \widehat{X}(I + \tilde{E})$$

として固有ベクトル行列を更新するものである．Algorithm 1 を使用して

$$\begin{cases} \tilde{X}^{(0)} := \widehat{X} \\ \tilde{X}^{(\nu)} \leftarrow \texttt{RefSyEv}(A, \tilde{X}^{(\nu-1)}) \quad (\nu = 1, 2, \ldots) \end{cases}$$

のように反復改良を行ったとき，重複固有値が存在しない場合には

$$\|E\|_2 < \min\left(\frac{\min\limits_{1\le i<j\le n} |\lambda_i - \lambda_j|}{10n\|A\|_2}, \frac{1}{100}\right), \tag{7}$$

重複固有値が存在する場合には

$$\|E\|_2 < \frac{1}{3}\min\left(\frac{\min\limits_{\lambda_i\ne\lambda_j,\ 1\le i,j\le n} |\lambda_i - \lambda_j|}{10n\|A\|_2}, \frac{1}{100}\right) \tag{8}$$

Algorithm 2 `RefSyEvCL` [10]: 10，11 行目，ノルムの計算以外は高精度計算を行う．

Require: $A = A^T \in \mathbb{R}^{n\times n},\ \widehat{X} \in \mathbb{R}^{n\times n}$
Ensure: $\tilde{X} \in \mathbb{R}^{n\times n}$
1: **function** $\tilde{X} \leftarrow$ `RefSyEvCL`$(A, \widehat{X})$
2: $[\tilde{X}, \tilde{D}, \tilde{E}, \omega] \leftarrow$ `RefSyEv`$(A, \widehat{X})$
3: **if** $\|\tilde{E}\|_2 \ge 1$, **return**, **end if**
4: Determine $\mathcal{J}_k,\ k = 1, \ldots, J$ for $\tilde{D} =$ `diag`$(\tilde{\lambda}_i)$ and ω.
5: **for** $k \leftarrow 1, \ldots, J$ **do**
6: $V_k \leftarrow \tilde{X}(:, \mathcal{J}_k)$
7: $\mu_k \leftarrow \left(\min_{i\in\mathcal{J}_k} \tilde{\lambda}_i + \max_{i\in\mathcal{J}_k} \tilde{\lambda}_i\right)/2$
8: $A_k \leftarrow A - \mu_k I$
9: $C_k \leftarrow V_k^T A_k V_k$
10: $T_k \leftarrow \mathrm{fl}(C_k)$ ▷ Conversion to ordinary precision
11: $[W_k, \tilde{\ }] \leftarrow$ `eig`(T_k) ▷ Compute eigenvectors of C_k
12: $V_k \leftarrow V_k W_k$
13: **repeat**
14: $[V_k, \tilde{\ }, \tilde{E}_k, \tilde{\ }] \leftarrow$ `RefSyEv`(A_k, V_k)
15: **if** $\|\tilde{E}_k\|_2 \ge 1$, **return**, **end if**
16: **until** $\|\tilde{E}_k\|_2 \le \|\tilde{E}\|_2$
17: $\tilde{X}(:, \mathcal{J}_k) \leftarrow V_k$
18: **end for**
19: **end function**

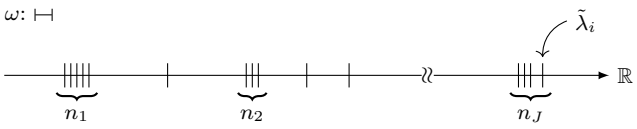

図 1 クラスタ化された $\tilde{\lambda}_i$

を満たせば，X への 2 次収束性が保証される（[9], Theorem 1，2）．

固有値がクラスタ化されている場合，E が式 (7)，(8) を満たすことは困難である．そこで，クラスタ化に対応した荻田・相島の反復改良法（[10], Algorithm 2）が提案されている．これを Algorithm 2 に示す．

Algorithm 2 は，以下の操作によりクラスタ化された近似固有値を隔離し，対応する近似固有ベクトルを更新する．

(1) Algorithm 1 を使用して

$$[\tilde{X}, \tilde{D}, \tilde{E}, \omega] \leftarrow \texttt{RefSyEv}(A, \widehat{X})$$

を求め，クラスタ化された $\tilde{\lambda}_i$ のインデックスの集合 $\mathcal{J}_k,\ k = 1, \ldots, J$ を

$$\begin{cases} \min\limits_{j\in\mathcal{J}_k\setminus\{i\}} |\tilde{\lambda}_i - \tilde{\lambda}_j| \le \omega & (\forall i \in \mathcal{J}_k) \\ |\tilde{\lambda}_i - \tilde{\lambda}_j| > \omega & (\forall i \in \mathcal{J}_k,\ \forall j \notin \mathcal{J}_k) \\ |\mathcal{J}_k| = n_k \ge 2 \end{cases}$$

のように求める．**図 1** はクラスタ化された $\tilde{\lambda}_i$ のイメージである．

(2) $\tilde{X}$ を使用して A を $S := \tilde{X}^T A\tilde{X}$ のように近似相似変換し，各クラスタに対応するように S を

$$S_k := S(\mathcal{J}_k, \mathcal{J}_k) \in \mathbb{R}^{n_k\times n_k}, \quad k = 1, \ldots, J$$

Algorithm 3 TwoSum [16]
Require: $a, b \in \mathbb{F}$
Ensure: $x, y \in \mathbb{F},\ x + y = a + b,\ x = \mathrm{fl}(a + b)$
1: **function** $[x, y] \leftarrow$ `TwoSum`(a, b)
2: $x \leftarrow \mathrm{fl}(a + b)$
3: $t \leftarrow \mathrm{fl}(x - a)$
4: $y \leftarrow \mathrm{fl}((a - (x - t)) + (b - t))$
5: **end function**

Algorithm 4 FastTwoSum [17]
Require: $a, b \in \mathbb{F},\ a \in 2\mathbf{u} \cdot \mathrm{ufp}(b)\mathbb{Z}$
Ensure: $x, y \in \mathbb{F},\ x + y = a + b,\ x = \mathrm{fl}(a + b)$
1: **function** $[x, y] \leftarrow$ `FastTwoSum`(a, b)
2: $x \leftarrow \mathrm{fl}(a + b)$
3: $y \leftarrow \mathrm{fl}(b - (x - a))$
4: **end function**

に分割する．

(3) $\mu_k := \left(\min_{i \in \mathcal{J}_k} \tilde{\lambda}_i + \max_{i \in \mathcal{J}_k} \tilde{\lambda}_i\right)/2$ に対して $T_k := S_k - \mu_k I_{n_k}$ とし，μ_k の周りのクラスタ化された固有値の差を相対的に拡大する．

(4) T_k の固有ベクトル行列 W_k を使用して，$\tilde{X}(:, \mathcal{J}_k)$ を $\tilde{X}(:, \mathcal{J}_k)W_k$ へ更新する．

3. 倍精度に基づく疑似 4 倍精度四則演算法

$a \in \mathbb{R}$ を近似する Double-Double 数 $x = (x_h, x_l)$ を

- x_h: a を倍精度に丸めた値，
- x_l: $a - x_h$ を倍精度に丸めた値

と定義する．本研究では，Algorithm 1，2 に含まれる行列積以外の高精度計算を Double-Double 数に対する四則演算法 [14], [15] により実装した．Double-Double 数に対する四則演算法には，浮動小数点数同士の和のエラーフリー変換アルゴリズム [16], [17] や浮動小数点数同士の積のエラーフリー変換アルゴリズム [18] が使用される．まず，それらを以下に記す．以降，$\mathrm{fl}(\cdot)$ は括弧内のすべての 2 項演算を浮動小数点演算により評価することを意味し，$\mathbb{F}$ はある固定された精度の 2 進浮動小数点数全体の集合を意味する．また，オーバーフロー，アンダーフローは考慮しない．

$a, b \in \mathbb{F}$ に対して

$$x + y = a + b, \quad x = \mathrm{fl}(a + b), \quad y = (a + b) - x$$

のような $x, y \in \mathbb{F}$ を求めるアルゴリズムを Algorithm 3，4 に記す．ただし，$\mathrm{ufp}(a)$ は $a \in \mathbb{R}$ の Unit in the First Place を意味する．

次に，$a, b \in \mathbb{F}$ に対して

$$x + y = a \cdot b, \quad x = \mathrm{fl}(a \cdot b), \quad y = (a \cdot b) - x$$

のような $x, y \in \mathbb{F}$ を求めるアルゴリズムを Algorithm 5 に記す．ただし，$a, b, c \in \mathbb{F}$ に対して $\mathtt{fma}(a, b, c)$ は，$a \cdot b + c$ を最近点へ丸めた浮動小数点数を求める関数である．

Algorithm 5 TwoProdFMA [18].
Require: $a, b \in \mathbb{F}$
Ensure: $x, y \in \mathbb{F},\ x + y = a \cdot b,\ x = \mathrm{fl}(a \cdot b)$
1: **function** $[x, y] \leftarrow$ `TwoProdFMA`(a)
2: $x \leftarrow \mathrm{fl}(a \cdot b)$
3: $y \leftarrow \mathtt{fma}(a, b, -x)$
4: **end function**

Algorithm 6 Double-Double 数の加算 [14], [15]
Require: $a = (a_h, a_l),\ b = (b_h, b_l),\ a_h, a_l, b_h, b_l \in \mathbb{F}$
Ensure: $c = (c_h, c_l),\ c_h, c_l \in \mathbb{F}$
1: **function** $c \leftarrow$ `dd_add`(a, b)
2: $[s_h, e_h] \leftarrow$ `TwoSum`(a_h, b_h)
3: $[s_l, e_l] \leftarrow$ `TwoSum`(a_l, b_l)
4: $s_e \leftarrow \mathrm{fl}(e_h + s_l)$
5: $[s'_h, s'_e] \leftarrow$ `FastTwoSum`(s_h, s_e)
6: $t_e \leftarrow s'_e + e_l$
7: $[c_h, c_l] \leftarrow$ `FastTwoSum`(s'_h, t_e)
8: **end function**

Algorithm 7 Double-Double 数の減算 [14], [15]
Require: $a = (a_h, a_l),\ b = (b_h, b_l),\ a_h, a_l, b_h, b_l \in \mathbb{F}$
Ensure: $c = (c_h, c_l),\ c_h, c_l \in \mathbb{F}$
1: **function** $c \leftarrow$ `dd_sub`(a, b)
2: $b'_h \leftarrow -b_h$
3: $b'_l \leftarrow -b_l$
4: $c \leftarrow$ `add`(a, b') ▷ $b' = (b'_h, b'_l)$
5: **end function**

Algorithm 8 Double-Double 数の乗算 [14], [15]
Require: $a = (a_h, a_l),\ b = (b_h, b_l),\ a_h, a_l, b_h, b_l \in \mathbb{F}$
Ensure: $c = (c_h, c_l),\ c_h, c_l \in \mathbb{F}$
1: **function** $c \leftarrow$ `dd_mul`(a, b)
2: $[p_1, p_2] \leftarrow$ `TwoProdFMA`(a_h, b_h)
3: $p_2 \leftarrow \mathrm{fl}(p_2 + a_h \cdot b_l)$
4: $p_2 \leftarrow \mathrm{fl}(p_2 + a_l \cdot b_h)$
5: $[c_h, c_l] \leftarrow$ `FastTwoSum`(p_1, p_2)
6: **end function**

Algorithm 9 Double-Double 数の除算 [14], [15]
Require: $a = (a_h, a_l),\ b = (b_h, b_l),\ a_h, a_l, b_h, b_l \in \mathbb{F},\ b_h \neq 0$
Ensure: $c = (c_h, c_l),\ c_h, c_l \in \mathbb{F}$
1: **function** $c \leftarrow$ `dd_div`(a, b)
2: $s \leftarrow a_h / b_h$
3: $[p, e] \leftarrow$ `TwoProdFMA`(s, b_h)
4: $t \leftarrow \mathrm{fl}((a_h - p - e + a_l - s \cdot b_l)/b_h)$
5: $[c_h, c_l] \leftarrow$ `FastTwoSum`(s, t)
6: **end function**

Algorithm 3〜5 を使用した Double-Double 数に対する四則演算法を Algorithm 6〜9 に記す．

4. 高精度行列積

尾崎らは，行列 $A \in \mathbb{F}^{m \times k}$，$B \in \mathbb{F}^{k \times n}$ に対して

$$A = \sum_{1 \leq i \leq p} A^{(i)}, \quad A^{(i)} \in \mathbb{F}^{m \times k}, \tag{9}$$

Algorithm 10 `Split_Mat` [11]

Require: $A \in \mathbb{F}^{m\times n},\ \ell \in \mathbb{N},\ LR \in \{\mathtt{'L'}, \mathtt{'R'}\}$
Ensure: $D = \sum_{i=1}^{\ell} D^{(i)} \in \mathbb{F}^{m\times n}$

function $D \leftarrow$ `Split_Mat`(A, ℓ, LR)
  $k \leftarrow 1$
  $D^{(1)} \leftarrow O$ ▷ zero matrix
  if $LR == \mathtt{'L'}$ **then** ▷ A is a multiplicand
    $\beta \leftarrow \mathrm{fl}(\lceil(-\log_2 \mathbf{u} + \log_2 n)/2\rceil)$
  else ▷ A is a multiplier
    $\beta \leftarrow \mathrm{fl}(\lceil(-\log_2 \mathbf{u} + \log_2 m)/2\rceil)$
  end if
  while $k < \ell$ **do**
    if $LR == \mathtt{'L'}$ **then** ▷ A is a multiplicand
      $\mu_i \leftarrow \max_{1\le j\le n} |a_{ij}|$ (for $1 \le i \le m$)
      if $\max_{1\le i\le m} \mu_i == 0$, **return**, **end if**
      $\omega_i \leftarrow \mathrm{fl}(2^{\lceil \log_2 \mu_i \rceil + \beta})$ (for $1 \le i \le m$)
      $S \leftarrow \mathrm{fl}((\omega_1, \ldots, \omega_m)^T \cdot (1, \ldots, 1))$
    else ▷ A is a multiplier
      $\mu_i \leftarrow \max_{1\le i\le m} |a_{ij}|$ (for $1 \le j \le n$)
      if $\max_{1\le i\le n} \mu_i == 0$, **return**, **end if**
      $\omega_i \leftarrow \mathrm{fl}(2^{\lceil \log_2 \mu_i \rceil + \beta})$ (for $1 \le i \le n$)
      $S \leftarrow \mathrm{fl}((1, \ldots, 1)^T \cdot (\omega_1, \ldots, \omega_m))$
    end if
    $D^{(k)} \leftarrow \mathrm{fl}((A + S) - S)$
    $A \leftarrow \mathrm{fl}(A - D^{(k)})$
    $k \leftarrow \mathrm{fl}(k + 1)$
  end while
  if $k == \ell,\ D^{(k)} \leftarrow A$ **end if**
end function

$$B = \sum_{1\le i\le q} B^{(i)}, \quad B^{(i)} \in \mathbb{F}^{k\times n} \tag{10}$$

のようにエラーフリー変換を行い，行列積 AB を

$$AB = \sum_{1\le i\le p,\ 1\le j\le q} \mathrm{fl}(A^{(i)}B^{(j)}) \tag{11}$$

のように変換する手法を提案した [11]．ただし，(9) の $A^{(r)} = (a_{ij}^{(r)})$ 及び (10) の $B^{(r)} = (b_{ij}^{(r)})$ は，それぞれ $s > t$ に対して

$$|a_{ij}^{(s)}| \ge |a_{ij}^{(t)}|, \quad a_{ij}^{(s)} \ne 0,$$
$$|b_{ij}^{(s)}| \ge |b_{ij}^{(t)}|, \quad b_{ij}^{(s)} \ne 0$$

を満たす．(11) は

$$\mathrm{fl}\left(\sum_{1\le i\le p,\ 1\le j\le q} \mathrm{fl}(A^{(i)}B^{(j)})\right)$$

のように計算できる．(9)，(10) は Algorithm 10（[11], Algorithm 3）により，

$$\widehat{A} \leftarrow \mathtt{Split\_Mat}(A, p, \mathtt{'L'}),$$
$$\widehat{B} \leftarrow \mathtt{Split\_Mat}(B, q, \mathtt{'R'})$$

のようにして求める．

本研究では，$p = q = 3$ とし，

Algorithm 11 `AccMul_AB`

Require: $A \in \mathbb{F}^{m\times k},\ B \in \mathbb{F}^{k\times n}$
Ensure: $C = C^{(1)} + C^{(2)} \in \mathbb{F}^{m\times n}$

function $C \leftarrow$ `AccMul_AB`(A, B)
  $\widehat{A} \leftarrow$ `Split_Mat`$(A, 3, \mathtt{'L'})$
  $\widehat{B} \leftarrow$ `Split_Mat`$(B, 3, \mathtt{'R'})$
  $D \leftarrow \mathrm{fl}(\widehat{A}^{(1)}\widehat{B}^{(1)})$
  $E \leftarrow \mathrm{fl}(\widehat{A}^{(1)}\widehat{B}^{(2)} + \widehat{A}^{(2)}\widehat{B}^{(1)})$
  $[C_{ij}^{(1)}, C_{ij}^{(2)}] \leftarrow$ `FastTwoSum`(D_{ij}, E_{ij})
  (for $1 \le i \le m,\ 1 \le j \le n$)
  $F \leftarrow \mathrm{fl}(\widehat{B}^{(2)} + \widehat{B}^{(3)})$
  $G \leftarrow \mathrm{fl}(\widehat{A}^{(1)}\widehat{B}^{(3)} + \widehat{A}^{(2)}F + \widehat{A}^{(3)}B)$
  $C^{(2)} \leftarrow \mathrm{fl}(C^{(2)} + G)$
end function

$$A = A^{(1)} + A^{(2)} + A^{(3)}, \quad B = B^{(1)} + B^{(2)} + B^{(3)}$$

の行列積を 2 つの行列 $C^{(1)}$, $C^{(2)}$ の和として返すように実装した．これを Algorithm 11 に示す．

5. 数値実験

本実験では，

(A) `pssyevd` により得られる固有ベクトル行列を X の初期値として Algorithm 2 を適用した場合の精度の確認，

(B) `pdsyevd` により得られる固有ベクトル行列を X の初期値として Algorithm 2 を適用した場合の性能の確認，

(C) `pssyevd` により得られる固有ベクトル行列を X の初期値として Algorithm 2 を適用した場合に，`pdsyevd` による結果よりも高速に高精度な結果を得られるかの確認

を行う．ただし，`pssyevd`，`pdsyevd` は ScaLAPACK のサブルーチンであり，それぞれ単精度，倍精度で計算を行う固有値ソルバである．`pssyevd` による初期値を使用する実験では，近似値の精度が低い場合には Algorithm 1，2 に倍精度演算，精度が良い場合には高精度計算を使用し，`pdsyevd` による初期値を使用する実験では，すべての反復に高精度計算を使用した．

高精度計算として，行列積には 4 章で述べたエラーフリー変換による高精度行列積（Algorithm 11）を使用し，その他の計算には 3 章で述べた倍精度に基づく Double-Double 数の四則演算法（Algorithm 6〜9）を使用した．ただし，Algorithm 2 の 7, 8 行目は倍精度で計算した．また，Algorithm 1 の 9 行目の $\tilde{E}$，$\tilde{X}$ は Double-Double 数の上位の倍精度のみ保持する．エラーフリー変換における最適化は，コンパイルオプション`-fp-model precise`，`-fprotect-parens` を適用することで抑制した．Algorithm 1 の 6 行目及び 2 の 3，15，16 行目ではスペクトルノルムが使用されているが，ScaLAPACK にはスペクトルノルムを求めるサブルーチンが存在しない．そこで，$M \in \mathbb{R}^{m\times n}$ に対する

$$\|M\|_F \le \sqrt{\min(m, n)} \cdot \|M\|_2$$

の関係（[19], (2.3.7)）から，スペクトルノルムの代わりに

表 1　MATLAB [20] を使用する環境

CPU	Intel Core i9-7900X (3.30 GHz)
メモリ容量	128 GB
コンパイラ	MATLAB R2020a

表 2　Oakforest-PACS システム

計算ノード	FUJITSU Server PRIMERGY CX1640 M1
CPU	Intel Xeon Phi プロセッサ 7250
倍精度理論演算性能	3.04 TFLOPS/1 ノード
単精度理論演算性能	6.09 TFLOPS/1 ノード
メモリ容量	高バンド幅: 16 GB/1 ノード
	低バンド幅: 96 GB/1 ノード
メモリバンド幅	高バンド幅: 490 GB/s
	低バンド幅: 115.2 GB/s
コア数	68 コア/1 ノード
スレッド数	4 スレッド/1 コア
コンパイラ	Intel C++ Compiler
コンパイルオプション	`-xHost -qopenmp`
	`-fp-model precise`
	`-fprotect-parens`

表 3　FUJITSU Supercomputer PRIMEHPC FX1000 システム

CPU	A64FX(Arm v8.2-A + SVE)
倍精度理論演算性能	3.3792 TFLOPS/1 ノード
単精度理論演算性能	6.7584 TFLOPS/1 ノード
メモリ容量	28 GiB/1 ノード
メモリバンド幅	1024 GB/s
コア数	48 コア + 2 アシスタントコア/1 ノード
CMG	4 グループ/1 ノード
コンパイラ	富士通 Technical Computing Suite
コンパイルオプション	`-Kfast -Kopenmp`

フロベニウスノルムを $2/\sqrt{\min(m,n)}$ 倍したものを使用した．実験環境には**表 1**〜**表 3** を使用した．

本実験では，

$$d_i := \mathrm{fl}(2^{(i-n)/(n-1)}), \quad i = 1, 2, \ldots, n$$

のような $d \in \mathbb{F}^n$ に対して，Algorithm 12 を使用して

$$[A, D] \leftarrow \texttt{Generate\_Problem}(d), \quad D = (\lambda_i)$$

のようにテスト行列 $A \in \mathbb{F}^{n\times n}$ とその真の固有値 $\lambda_i \in \mathbb{F}$, $i = 1, 2, \ldots, n$ を生成した [21]．λ_i に対応する固有ベクトルは Hadamard 行列の第 i 列目を $\pm 1/\sqrt{n}$ 倍したものに一致する．Algorithm 12 の 2 行目の $\texttt{hadamard}(n)$ は n 次 Hadamard 行列を生成する関数であり，4 行目の $\mathrm{fl}_\triangle(\cdot)$ は括弧内のすべての 2 項演算を上向きの丸めで評価する．本実験では $n = 2^k$, $k \in \mathbb{N}$ とした．

5.1　MATLAB における実験結果

MATLAB による実行結果を**表 4** に示す．ただし，単精度による結果を使用する場合は，第 1，2 反復目を倍精度演算，第 3 反復目を高精度計算で実行した．また，得られた近似固有値 $\tilde{D}$，近似固有ベクトル行列 $\tilde{X}$，及び固有値ソルバを実行する際の精度 $p \in \{s, d\}$（s: 単精度，d: 倍精度）に対して

$$pD_{err} := \max_{1\le i\le n} \frac{|\lambda_i - \tilde{\lambda}_i|}{\lambda_i}, \quad pX_{err} := \frac{\||X| - |\tilde{X}|\|_2}{\|X\|_2} \tag{12}$$

であり，*stime*, *dtime* は各関数の計算時間を表す．(12) の X は $\mathrm{fl}(\texttt{hadamard}(n)/\sqrt{n})$ のように生成しており，$n = 2^{2k} \in \mathbb{F} \cap \mathbb{N}$, $k \in \mathbb{N} \cup \{0\}$ であれば丸め誤差は生じない．表 4 より，単精度による結果を初期値とした場合でも，$\|E\|_2 < 1$ となり，精度が改善され，最終的には真の固有対が得られた．倍精度による結果を初期値とした場合には，1 回の反復で真の固有対を得た．計算時間については，倍精度による結果を初期値とした方が約 1.4 倍高速に真の固有対を得た．

Algorithm 12 d_i を目標値として固有値 λ_i を生成し，それを真の固有値とする行列 A を生成する [21].

Require: $d \in \mathbb{F}^n$, $n = 2^k \in \mathbb{N}$, $k \in \mathbb{N}$
Ensure: $A \in \mathbb{F}^{n\times n}$, $D = (\lambda_i) \in \mathbb{F}^n$
1: **function** $[A, D] \leftarrow \texttt{Generate\_Problem}(d)$
2: $H \leftarrow \texttt{hadamard}(n)$ ▷ $H \in \mathbb{F}^{n\times n}$: Hadamard 行列
3: $t \leftarrow \mathrm{ufp}\left(\max_{1\le i\le n} |d_i|\right)$
4: $t \leftarrow \mathrm{fl}_\triangle(12 \cdot n \cdot t)$
5: $\lambda_i \leftarrow \mathrm{fl}((t + d_i) - t)$ (for $1 \le i \le n$)
6: $A \leftarrow \mathrm{fl}(H^T \cdot \texttt{diag}(\lambda) \cdot H)$ ▷ No rounding error occurs
7: $\lambda_i \leftarrow \mathrm{fl}(n \cdot \lambda_i)$ (for $1 \le i \le n$)
8: $D \leftarrow (\lambda_1, \ldots, \lambda_n)^T$
9: **end function**

表 4　MATLAB において `eig` 関数及び Algorithm 2 を 3 回実行したときの累積計算時間（sec）と相対誤差

n	結果	`eig`	1 回目	2 回目	3 回目
2^8	*stime*	2.29e-03	3.76e-01	7.50e-01	2.54e+00
	sD_{err}	2.42e-07	1.84e-11	1.84e-15	0.00e+00
	sX_{err}	3.77e-04	1.55e-08	2.63e-13	0.00e+00
	dtime	4.49e-03	1.76e+00	3.55e+00	5.33e+00
	dD_{err}	4.14e-16	0.00e+00	0.00e+00	0.00e+00
	dX_{err}	6.78e-13	0.00e+00	0.00e+00	0.00e+00
2^{10}	*stime*	3.16e-02	5.82e+00	1.16e+01	4.01e+01
	sD_{err}	3.34e-07	9.68e-11	1.61e-15	0.00e+00
	sX_{err}	2.47e-03	3.93e-07	1.50e-12	0.00e+00
	dtime	5.16e-02	2.84e+01	5.68e+01	8.54e+01
	dD_{err}	3.63e-16	0.00e+00	0.00e+00	0.00e+00
	dX_{err}	4.64e-12	0.00e+00	0.00e+00	0.00e+00

5.2　Oakforest-PACS における実験結果

本実験にはワークスペース含め倍精度の行列を 18 個使用するため，Oakforest-PACS の 1 ノードに記憶できる行列サイズは最大で約 $26,755$ である（$2^{14} < 26,755 < 2^{15}$）．

表 5　Oakforest-PACS において pssyevd，pdsyevd 及び Algorithm 2 を 3 回実行したときの累積計算時間（sec）と相対誤差

n	結果	ソルバ	1 回目	2 回目	3 回目
2^{14}	$stime$	6.41e+01	1.49e+02	1.56e+02	1.61e+02
	sD_{err}	5.56e-07	9.05e-09	2.70e-12	0.00e+00
	sX_{err}	2.00e-03	1.13e-05	1.46e-09	9.38e-17
	$dtime$	7.25e+01	1.17e+02	1.61e+02	2.06e+02
	dD_{err}	4.44e-16	0.00e+00	0.00e+00	0.00e+00
	dX_{err}	3.94e-12	0.00e+00	0.00e+00	0.00e+00
2^{16}	$stime$	6.43e+02	2.54e+03	4.54e+03	4.91e+03
	sD_{err}	3.22e-06	2.77e-07	1.20e-06	1.13e-08
	sX_{err}	1.51e-02	1.16e-02	4.41e-03	2.99e-05
	$dtime$	1.05e+03	2.86e+03	4.70e+03	6.54e+03
	dD_{err}	3.55e-16	0.00e+00	0.00e+00	0.00e+00
	dX_{err}	1.21e-11	0.00e+00	0.00e+00	0.00e+00

ここでは，

- 行列サイズ: $2^{14}(=16,384)$, $2^{16}(=65,536)$
- ノード数: 64
- MPI プロセス数: $512(=64\times 8)$
- OMP スレッド数: 34

として実験を行った．1 ノード当たりに保持する行列サイズはそれぞれ 2^{11}, 2^{13} となる．実験結果を**表 5** に記す．ただし，sX_{err}, dX_{err} の計算ではスペクトルノルムの代わりにフロベニウスノルムを使用した．また，単精度による結果を使用する場合は，$n=2^{14}$ については第 1，2 反復目を倍精度演算，第 3 反復目を高精度計算で実行し，$n=2^{16}$ についてはすべての反復を倍精度演算で実行した．

表 5 より，テスト問題に対する精度の改善が確認できる．$n=2^{16}$ の単精度実行については，初期値の精度が十分ではないため，1 回目の sX_{err} は少し拡大したが，以降の収束性を確認した．同様の結果が $n=21$ の Wilkinson 行列に対する実験にて確認されている [10]．倍精度による結果を初期値とした場合には，1 回の反復で真の固有対を得た．計算時間については，単精度による結果を改善するよりも，倍精度による結果を改善したほうが高速に高精度な結果を得た．

5.3　不老におけるベンチマーク

pssyevd，pdsyevd，及び倍精度で行列積を計算する ScaLAPACK のサブルーチン pdgemm の FX1000 におけるベンチマークを紹介する．固有値ソルバの計算時間とノード数，MPI プロセス数の関係を**表 6** に記す．pdgemm の計算時間とノード数，MPI プロセス数の関係を**表 7** に記す．表 7 から推定される Algorithm 1 の計算時間とその pdsyevd に対する比率を**表 8** に示す．表 8 より，FX1000 において Algorithm 1 を実行したとき，ノード数の 4 倍の MPI プロセス数を指定した場合には pdsyevd の約 0.9 倍，

表 6　FX1000 における固有値ソルバの計算時間（sec）

n	ノード	MPI	pssyevd	pdsyevd
$1\cdot 10^4$	4	4	24.6	38.2
$1\cdot 10^4$	4	16	10.2	15.2
$2\cdot 10^4$	4	4	105	176
$2\cdot 10^4$	4	16	35.1	60.4
$4\cdot 10^4$	4	4	513	930
$4\cdot 10^4$	4	16	146	327
$4\cdot 10^4$	16	16	212	422
$4\cdot 10^4$	16	64	72	126

表 7　FX1000 における行列積の計算時間（sec）

n	ノード	MPI	pdgemm
$1\cdot 10^4$	4	4	0.42
$1\cdot 10^4$	4	16	0.41
$2\cdot 10^4$	4	4	2.05
$2\cdot 10^4$	4	16	2.11
$4\cdot 10^4$	4	4	12.4
$4\cdot 10^4$	4	16	13.1
$4\cdot 10^4$	16	16	4.92
$4\cdot 10^4$	16	64	4.28

表 8　FX1000 における Algorithm 1 の推定計算時間（sec）と pdsyevd に対する比率

n	ノード	MPI	Alg. 1	比率
$1\cdot 10^4$	4	4	10.1	0.26
$1\cdot 10^4$	4	16	9.84	0.65
$2\cdot 10^4$	4	4	49.2	0.28
$2\cdot 10^4$	4	16	50.6	0.84
$4\cdot 10^4$	4	4	298	0.32
$4\cdot 10^4$	4	16	314	0.96
$4\cdot 10^4$	16	16	118	0.28
$4\cdot 10^4$	16	64	103	0.82

ノード数と同じ数の MPI プロセス数を指定した場合には pdsyevd の約 0.3 倍の計算時間を要すると推定される．

6.　結論

本研究では，荻田・相島が提案した実対称行列の固有値分解に対する反復改良法を大規模並列環境において実装した．荻田・相島の反復改良法に要する高精度計算は，Bailey らによる Double-Double 数の四則演算法，及び尾崎らによるエラーフリー変換を用いた高精度行列積によって再現した．数値実験では，サイズが 2^{14}，2^{16} の問題に対して，単精度，倍精度ともに精度の改善を確認し，大規模並列環境における実験結果を示した．また，単精度による結果を初期値とする場合には，第 1，2 反復目を倍精度演算で実装しても高精度行列積を用いた結果と同程度の精度を得た．本実験ではスーパーコンピュータ「Oakforest-PACS」を利用したが，スーパーコンピュータ「不老」では単精度と倍精度の固有値ソルバのベンチマーク比がより拡大するため，

単精度による結果を改善したほうが高速に高精度な結果を得られる可能性がある．

今後の展望として，高精度計算の行列積の回数の調整を行い，高速化を検討する．Oakforest-PACS については，実行時オプション等の調整パラメータが多数存在するため，チューニングを行い高速化する．また，GPU における実装，実験を予定している．

謝辞 本研究は，JST/CREST「モデリングのための精度保証付き数値計算論の展開」，HPCI「Developing Accuracy Assured High Performance Numerical Libraries for Eigenproblems」の支援による．

参考文献

[1] J.H. Wilkinson: *The algebraic eigenvalue problem*, Clarendon Press, Oxford (1965).
[2] H. Bowdler, R.S. Martin, C. Reinsch, J. H. Wilkinson: The QR and QL algorithms for symmetric matrices, *Numer. Math.*, **11** (1968), 293–306.
[3] J.J.M. Cuppen: A divide and conquer method for the symmetric tridiagonal eigenproblem, *Numer. Math.*, **36**:2 (1980), 177–195.
[4] M. Gu, S.C. Eisenstat: A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem, *SIAM J. Matrix Anal. Appl.*, **16**:1 (1995), 172–191.
[5] J.W. Demmel: *Applied numerical linear algebra*, SIAM, Philadelphia (1997).
[6] I.S. Dhillon: A new O(n^2) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem, (Ph. D. thesis, University of California, 2004).
[7] I.S. Dhillon, B.N. Parlett, C. Vömel: The design and implementation of the MRRR algorithm, *ACM Trans. Math. Softw.*, **32**:4 (2006), 533–560.
[8] Scalable Linear Algebra PACKage (ScaLAPACK), `http://www.netlib.org/scalapack/`.
[9] T. Ogita, K. Aishima: Iterative refinement for symmetric eigenvalue decomposition, *Jpn. J. Ind. Appl. Math.*, **35**:3 (2018), 1007–1035.
[10] T. Ogita, K. Aishima: Iterative refinement for symmetric eigenvalue decomposition II: clustered eigenvalues, *Jpn. J. Ind. Appl. Math.*, **36**:2 (2019), 435–459.
[11] K. Ozaki, T. Ogita, S. Oishi, S.M. Rump: Error-free transformation of matrix multiplication by using fast routines of matrix multiplication and its applications, *Numer. Algorithms*, **59**:1 (2012), 95–118.
[12] FUJITSU Supercomputer PRIMEHPC, `https://www.fujitsu.com/jp/products/computing/servers/supercomputer/`.
[13] Oakforest-PACS スーパーコンピュータシステム, `https://www.cc.u-tokyo.ac.jp/guide/hpc/ofp/`
[14] Y. Hida, X.S. Li, D.H. Bailey: Quad-double arithmetic: algorithms, implementation, and application, Technical Report LBNL-46996, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (2000).
[15] Y. Hida, X.S. Li, D.H. Bailey: Algorithms for quad-double precision floating point arithmetic, *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, 155–162 (2001).
[16] D.E. Knuth: *Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley Professional (1997).
[17] T.J. Dekker: A Floating-Point Technique for Extending the Available Precision, *Numer. Math.*, **18**:3 (1971), 224–242.
[18] A.H. Karp, P. Markstein: High-Precision Division ans Square Root, *ACM Trans. Math. Softw.*, **23** (1997), 561–589.
[19] G.H. Golub, C.F. Van Loan: *Matrix Computations, 4th ed.*, The Johns Hopkins University Press, Baltimore (2013).
[20] MathWorks, `https://jp.mathworks.com/`.
[21] 尾崎 克久，荻田 武史：厳密な固有値・特異値がわかる行列の生成法，日本応用数理学会年会，名古屋大学 (2018/09/3–5)．