A mixed precision LOBPCG algorithm

Daniel Kressner
École Polytechnique Fédérale de Lausanne
Yuxin Ma (yxma18@fudan.edu.cn)
Fudan University
Meiyue Shao
Fudan University

Research Article

Keywords: Symmetric eigenvalue problem, LOBPCG algorithm, mixed precision algorithm

Posted Date: March 2nd, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2623686/v1

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Numerical Algorithms on May 11th, 2023.
See the published version at https://doi.org/10.1007/s11075-023-01550-9.

A mixed precision LOBPCG algorithm

Daniel Kressner[1†], Yuxin Ma[2*†] and Meiyue Shao[3,4†]

[1]Institute of Mathematics, EPFL, Lausanne, CH-1015, Switzerland.

[2*]School of Mathematical Sciences, Fudan University, Shanghai, 200433, China.

[3]School of Data Science, Fudan University, Shanghai, 200433, China.

[4]MOE Key Laboratory for Computational Physical Sciences, Fudan University, Shanghai, 200433, China.

*Corresponding author(s). E-mail(s): yxma18@fudan.edu.cn;
Contributing authors: daniel.kressner@epfl.ch;
myshao@fudan.edu.cn;
†These authors contributed equally to this work.

Abstract

The locally optimal block preconditioned conjugate gradient (LOBPCG) algorithm is a popular approach for computing a few smallest eigenvalues and the corresponding eigenvectors of a large Hermitian positive definite matrix $\boldsymbol{A}$. In this work, we propose a mixed precision variant of LOBPCG that uses a (sparse) Cholesky factorization of $\boldsymbol{A}$ computed in reduced precision as the preconditioner. To further enhance performance, a mixed precision orthogonalization strategy is proposed. To analyze the impact of reducing precision in the preconditioner on performance, we carry out a rounding error and convergence analysis of PINVIT, a simplified variant of LOBPCG. Our theoretical results predict and our numerical experiments confirm that the impact on convergence remains marginal. In practice, our mixed precision LOBPCG algorithm typically reduces the computation time by a factor of **1.4**–**2.0** on both CPUs and GPUs.

Keywords: Symmetric eigenvalue problem, LOBPCG algorithm, mixed precision algorithm

MSC Classification: c

1 Introduction

Given a large Hermitian positive definite matrix $A \in \mathbb{C}^{n\times n}$, this work considers the computation of the k smallest eigenvalues $0 < \lambda_1 \leq \cdots \leq \lambda_k$ and the corresponding eigenvectors $x_1, \ldots, x_k$ satisfying

$$AX = X\Lambda,$$

where $X = [x_1, \ldots, x_k]$ and Λ is diagonal with diagonal entries $\lambda_1, \ldots, \lambda_k$. This problem is often encountered in many applications, such as PDE and optimization problem, electronic structure calculations and machine learning; see, for example, [1–3].

When a good preconditioner T for A is available, the preconditioned inverse iteration (PINVIT) from [4] is a good candidate for solving such eigenvalue problems. For $k = 1$, PINVIT takes the form

$$x_{i+1} = x_i - T\big(Ax_i - \rho(x_i)x_i\big),$$

for some starting vector x_0. Here, $\rho(x) = (x^*Ax)/(x^*x)$ denotes the Rayleigh quotient, which is also used to approximate the eigenvalue at each iteration. Note that PINVIT with the "ideal" preconditioner $T = A^{-1}$ becomes equivalent to inverse iteration. When computing several ($k > 1$) smallest eigenpairs, one chooses a starting matrix $X_0 \in \mathbb{C}^{n\times m}$ ($m \geq k$) with orthonormal columns and one step of the block version of PINVIT [5] takes the form

$$\tilde{X}_{i+1} = X_i - T(AX_i - X_i\Theta_i),$$

where $\Theta_i = X_i^*AX_i$. The next iterate X_{i+1} is obtained from orthonormalizing the columns of $\tilde{X}_{i+1}$ by, e.g., a QR factorization. Under mild conditions, linear convergence of PINVIT is proven in [6], with a convergence rate depending on the quality of the preconditioner T. The locally optimal block preconditioned conjugate gradient (LOBPCG) method [7] aims at accelerating the convergence of PINVIT by choosing the next iterate optimally from a $3m$-dimensional subspace that contains the current as well as the previous iterate and the preconditioned residual; see Section 2 for more details. LOBPCG converges at least as fast as PINVIT and often significantly faster.

Executing an algorithm in reduced (single) precision on, e.g., a GPU, can be significantly faster than executing it in default working (double) precision. On the other hand, critical applications may require eigenvalues and eigenvectors computed to an accuracy warranted by working precision. In such a scenario the use of mixed precision algorithms can be beneficial; see [8, 9] for an overview. For example, Carson and Higham [10] proposed a general framework for large-scale mixed precision linear system solvers based on iterative refinement. It is highlighted that a mixed precision algorithm can be twice as fast as a traditional linear system solver by computing the most expensive part—LU factorization—in reduced precision. For eigenvalue problems, mixed

precision algorithms have recently been proposed for computing *all* eigenvalues and eigenvectors of a dense matrix. This includes the Newton-like iterative refinement methods for symmetric [11–13] and nonsymmetric [14] eigenvalue problems, as well as a mixed precision one-sided Jacobi SVD algorithm [15]. If only a few eigenvalues and eigenvectors are of interest, one could combine mixed precision with classical iterative refinement [16] for eigenvalue problems, which solves linear systems with the shifted matrix $A - \hat{\lambda}_i I$ in order to correct an approximation $\hat{\lambda}_i$ of the ith eigenvalue. The need for solving several differently shifted linear systems makes such an approach rather expensive.

In this work, we propose mixed precision PINVIT and LOBPCG algorithms that use a (sparse) Cholesky factorization of A computed in reduced precision as preconditioner. This reduces the cost of accurately computing eigenvalues and eigenvectors in significantly compared to inverse iteration, which requires to carry out the Cholesky factorization in working precision. On the theoretical side, we carry out a rounding error analysis of PINVIT, which predicts that reducing precision in the preconditioner usually only has a marginal impact on convergence. On the experimental side, we demonstrate for sparse matrices that our mixed precision LOBPCG algorithm results in up to 1.43× speedup on a CPU and 1.67× speedup on a GPU. For dense matrices, the speedups are 1.67× on a CPU and 2.00× on a GPU.

The rest of this paper is organized as follows. In Section 2, we explain the basic ideas of LOBPCG algorithm. Then in Section 3, we propose our mixed precision algorithms and the details of the implementation. The analysis is shown in Section 4 and numerical experiments are presented in Section 5 to show the efficiency of our mixed precision LOBPCG algorithm.

2 LOBPCG algorithm

In this section, we explain the basic idea of the LOBPCG algorithm from [7]. For $k = 1$, LOBPCG can be derived from the preconditioned conjugate gradient (PCG) method. PCG applied to the (singular) linear system $(A - \lambda_1 I)x = 0$ with preconditioner T and initial guess x_0 is a three-term recurrence of the form

$$\begin{aligned} x_{i+1} &= x_i + \alpha_i T(A - \lambda_1 I)x_i + \beta_i(x_i - x_{i-1}) \\ &= (1 + \beta_i)x_i + (-\beta_i)x_{i-1} + \alpha_i T(A - \lambda_1 I)x_i, \end{aligned}$$

where α_i, β_i are chosen to minimize $x_{i+1}^*(A - \lambda_1 I)x_{i+1}$. As the smallest eigenvalue λ_1 is usually unknown, it needs to be replaced by an approximation, the Rayleigh quotient $\rho(x_i)$, leading to the basic form of LOBPCG:

$$x_{i+1} = \alpha_1^{(i)} x_i + \alpha_2^{(i)} x_{i-1} + \alpha_3^{(i)} T(Ax_i - \rho(x_i)x_i),$$

where $\alpha_1^{(i)}$, $\alpha_2^{(i)}$, and $\alpha_3^{(i)}$ are chosen to minimize $\rho(x_{i+1})$. Note that, unlike PCG, LOBPCG is not a Krylov subspace method in the usual sense because $\rho(x_i)$ is different in each iteration.

For $k > 1$, LOBPCG takes an initial guess $X_0 \in \mathbb{C}^{n\times m}$ with $m \geq k$, and produces iterates of the form

$$X_{i+1} = X_i C_1^{(i)} + X_{i-1} C_2^{(i)} + W_i C_3^{(i)} = \begin{bmatrix} X_i & X_{i-1} & W_i \end{bmatrix} \begin{bmatrix} C_1^{(i)} \\ C_2^{(i)} \\ C_3^{(i)} \end{bmatrix} =: S_i C_i,$$

where $W_i = T(AX_i - X_i\Theta_i)$ with $\Theta_i = X_i^* A X_i$. The $3m \times m$ matrix C_i is chosen to minimize

$$\min_{X_{i+1}^* X_{i+1} = I} \operatorname{tr}(X_{i+1}^* A X_{i+1}) = \min_{C_i^* S_i^* S_i C_i = I} \operatorname{tr}(C_i^* S_i^* A S_i C_i), \tag{1}$$

where $\operatorname{tr}(\cdot)$ denotes the trace of a matrix. By the Rayleigh–Ritz method, a solution C_i of (1) is obtained from the eigenvectors belonging to the m smallest eigenvalues of the generalized eigenvalue problem $S_i^* A S_i y = \lambda S_i^* S_i y$; see [17, Section 8.7.2] for numerical algorithms.

Let us stress that the actual implementation of LOBPCG is quite different [18] due to the numerical instability caused by the ill-conditioning of S_i. In practice $[X_i, X_{i-1}]$ can be orthogonalized by an improved Hetmaniuk–Lehoucq trick [18, Section 4.2], and then the remaining block, W_i, also needs to be orthogonalized carefully.

3 Mixed precision algorithms

In this section, we derive a mixed precision LOBPCG algorithm. For this purpose, we consider two precisions: a working precision and a lower/reduced precision, e.g., IEEE double and single precisions. The input and output data of our algorithms are always stored in working precision. The functions `lower`$(\cdot)$ and `working`$(\cdot)$ are used to convert working precision data into lower precision and vice versa.

3.1 Lower precision preconditioning

The application of the preconditioner T usually consumes a considerable fraction of the computational expense of PINVIT and LOBPCG. This suggests to implement the application of T in lower precision. In most cases, we expect that this only has a small impact on convergence. While a more detailed analysis will be provided in Section 4, the existing convergence analysis of PINVIT already provides a good intuition.

By [6, Theorem 2.1], PINVIT with $k = 1$ converges to the smallest eigenvalue and eigenvector when $\gamma := \|I - A^{1/2} T A^{1/2}\|_2 < 1$ and additional mild conditions are satisfied. Asymptotically, the convergence is linear with a rate

Algorithm 1 Mixed precision PINVIT algorithm

Require: A Hermitian positive definite matrix $A \in \mathbb{C}^{n\times n}$; an initial approximate eigenvectors $X_0 \in \mathbb{C}^{n\times m}$; the number of desired eigenpairs $k \leq m$; the maximum number of iterations `maxit`; a function $f_T(\cdot)$ to apply preconditioner T (in lower precision).

Ensure: The diagonal matrix $\Theta \in \mathbb{R}^{k\times k}$ contains the computed smallest eigenvalues, and $X \in \mathbb{C}^{n\times k}$ contains the corresponding computed eigenvectors satisfying $AX = X\Theta$.

$\tilde{X} \leftarrow X_0$.
for $i = 1, 2, \ldots,$ `maxit` **do**
  $X \leftarrow Q$ where Q satisfies $\tilde{X} = QR$.
  $\Theta \leftarrow X^*AX$.
  Compute the residual $R = AX - X\Theta$.
  if k smallest eigenpairs have converged **then**
    Return $X \leftarrow X(:, 1:k)$ and $\Theta \leftarrow \Theta(1:k, 1:k)$.
  end if
  Compute $W_{\texttt{lower}} \leftarrow f_T(\texttt{lower}(R))$ in a lower precision.
  $W \leftarrow \texttt{working}(W_{\texttt{lower}})$.
  $\tilde{X} \leftarrow X - W$.
end for

that is bounded by $\gamma + (1-\gamma)\lambda_1/\lambda_2$. If T is perturbed by rounding error in lower precision one effectively applies a preconditioner T_E, which remains close to T. In turn, the convergence is now determined by $\|I - A^{1/2}T_EA^{1/2}\|_2$, which remains close to γ. Unless γ is very close to 1 we thus expect that replacing T by T_E does not affect convergence significantly. These considerations lead to Algorithm 1, PINVIT with a lower precision preconditioner.

3.2 A mixed precision orthogonalization procedure

In both PINVIT and LOBPCG, we need to produce an orthogonal basis of the searching subspace in each iteration. Moreover, orthogonalization plays an important role to ensure numerical stability for the LOBPCG algorithm [18, 19]. We need to perform the orthogonalization procedure as accurately as possible. However, orthogonalization is often quite expensive in practice. Therefore it is desirable to make use of a lower precision to accelerate this procedure.

There are mainly two existing mixed precision algorithms for computing the QR factorization. The algorithm proposed in [20] uses higher precision to compute the inner product to enhance the numerical stability of Cholesky-QR algorithm. The drawback is that this algorithm can be much slower than the standard Cholesky-QR algorithm if higher precision arithmetic lacks hardware support. To improve the performance, a mixed precision block Gram–Schmidt orthogonalization algorithm was proposed in [21]. For both algorithms the

Algorithm 2 Mixed precision QR factorization algorithm

Require: A matrix $A \in \mathbb{C}^{n\times m}$ with $\operatorname{rank}(A) = m$.

Ensure: A matrix $Q \in \mathbb{C}^{n\times m}$ and an upper triangular matrix $R \in \mathbb{C}^{m\times m}$ satisfying $A = QR$ and $Q^*Q = I_m$.

1: Compute the QR factorization of A in lower precision, i.e., $\texttt{lower}(A) = Q_{\texttt{lower}} R_{\texttt{lower}}$.
2: Compute $V \leftarrow A \cdot \texttt{working}(R_{\texttt{lower}}^{-1})$ by solving an upper triangular linear system.
3: Compute Cholesky factorization of V^*V such that $V^*V = LL^*$.
4: Compute $Q \leftarrow VL^{-*}$ by solving an upper triangular linear system.

orthogonality of the output depends linearly on the condition number of the input.

We propose another mixed precision approach for orthogonalization. We first use Householder-QR to factorize $\texttt{lower}(W_i) = Q_{\texttt{lower}} R_{\texttt{lower}}$ in lower precision. Then $\texttt{working}(R_{\texttt{lower}})$ is used as a preconditioner—we apply Cholesky-QR to the preconditioned matrix $W_i \cdot \texttt{working}(R_{\texttt{lower}}^{-1})$ to refine the orthogonality. Under mild assumptions $W_i \cdot \texttt{working}(R_{\texttt{lower}}^{-1})$ is reasonably well-conditioned, so that the Cholesky-QR algorithm is sufficiently accurate. This mixed precision QR factorization algorithm is summarized in Algorithm 2.

3.3 A mixed precision LOBPCG algorithm

In addition to preconditioning and orthogonalization, the application of A and other parts of PINVIT and LOBPCG may also constitute nonnegligible expenses, depending on the specific setting. Carrying out these parts in lower precision bears the risk of limiting the attainable accuracy to lower precision. However, very often it is still possible to further exploit lower precision arithmetic.

As PINVIT and LOBPCG converge linearly in general, we can break the computation in two stages as follows. In the first stage we can first perform all computations in lower precision to produce an approximate solution in lower precision. Then in the second stage we switch back to the working precision while using the approximate solution as an initial guess and applying lower precision preconditioning. In this manner we are able to obtain a satisfactory solution in working precision by making use of lower precision arithmetic as much as possible.

In summary, we compute a good initial guess in lower precision, and then refine the solution using the LOBPCG algorithm in working precision. Lower precision are exploited in both preconditioning and orthogonalization in the LOBPCG algorithm. The resulting mixed precision LOBPCG algorithm is summarized in Algorithm 3.

Algorithm 3 Mixed precision LOBPCG algorithm

Require: A Hermitian positive definite matrix $A \in \mathbb{C}^{n\times n}$; an initial approximate eigenvectors $X_0 \in \mathbb{C}^{n\times m}$; the number of desired eigenpairs $k \le m$; the maximum number of iterations `maxit`; a function $f_T(\cdot)$ to apply preconditioner T (in lower precision).

Ensure: The diagonal matrix $\Theta \in \mathbb{C}^{k\times k}$ contains the computed smallest eigenvalues and $X \in \mathbb{C}^{n\times k}$ contains the corresponding computed eigenvectors satisfying $AX = X\Theta$.

Compute X by a lower precision LOBPCG algorithm with the initial guess $\texttt{lower}(X_0)$.
$P \leftarrow [\,]$, $\Theta \leftarrow X^*AX$.
for $i = 1, 2, \ldots,$ `maxit` **do**
  Compute residual $R \leftarrow AX - X\Theta$.
  Determine the number of convergence eigenpairs n_c.
  if $n_c \ge k$ **then**
    Return $X \leftarrow X(:, 1:k)$ and $\Theta \leftarrow \Theta(1:k, 1:k)$.
  end if
  Compute $W_{\texttt{lower}} \leftarrow f_T(\texttt{lower}(R))$ in a lower precision.
  $W \leftarrow \texttt{working}(W_{\texttt{lower}})$.
  Orthogonalize W against $[X, P]$ twice.
  Factorize $W = QR$ and then replace W by Q.
  $A_p \leftarrow S^*AS$ with $S = [X, P, W]$.
  Solve eigenvalue problem of A_p to obtain $A_pC = CD$.
  Orthogonalize $C(1:m, m+1:n)^*$ to obtain an orthogonal matrix V.
  $X \leftarrow SC(:, 1:m)$, $P \leftarrow SC(:, m+1:2m)V$ and $\Theta \leftarrow D(1:m, 1:m)$.
end for

4 Convergence in finite-precision arithmetic

In our experiments, we observe that rounding error does not significantly affect the convergence of Algorithms 1 and 3 until an accuracy on the level of *working* precision is reached. To gain theoretical insights on this observation, we study the effect of rounding error on PINVIT for $k = 1$:

$$x_{i+1} = x_i - T\big(Ax_i - \rho(x_i)x_i\big). \tag{2}$$

For simplicity, we consider real matrices, that is, $A \in \mathbb{R}^{n\times n}$ is positive definite with eigenvalues $0 < \lambda_1 < \lambda_2 \le \cdots \le \lambda_n$. Moreover, we assume that n^{-1} is far larger than the unit roundoff, even in reduced precision.

In analyzing the effect of rounding error on (2), we assume that the computed matrix–vector product $\mathrm{fl}(Ax_i)$ satisfies the backward error

$$\mathrm{fl}(Ax_i) = (A + \Delta A)x_i \qquad \text{with} \qquad \|\Delta A\|_2 \le \epsilon_A \|A\|_2, \tag{3}$$

for some symmetric ΔA (depending on x_i). When carrying out standard matrix–vector multiplication with a dense or sparse matrix A then Lemma 6.6 in [22] states that (3) holds with

$$\epsilon_A = \sqrt{n}\gamma_n^h \qquad \text{with} \qquad \gamma_n^h = \frac{n\boldsymbol{u}_h}{1 - n\boldsymbol{u}_h},$$

where $\boldsymbol{u}_h$ denotes the unit roundoff in working precision.

Lemma 1. *Let $\hat{r}_i$ denote the result of evaluating $r_i := Ax_i - \rho(x_i)x_i$ in working precision. Assuming that (3) holds, there exist a symmetric matrix $F \in \mathbb{R}^{n\times n}$ and a diagonal matrix $E \in \mathbb{R}^{n\times n}$ such that*

$$\hat{r}_i = (I + E)(r_i + Fx_i),$$

where $\|E\|_2 \le \boldsymbol{u}_h$ and $\|F\|_2 \le \epsilon_r\|A\|_2$ with

$$\epsilon_r = \big(\gamma_n^h + \epsilon_A + \gamma_n^h\epsilon_A + (n+1)\boldsymbol{u}_h\big)\frac{1 + \boldsymbol{u}_h}{1 - 2n\boldsymbol{u}_h} + \epsilon_A + \boldsymbol{u}_h.$$

Proof We first analyze the rounding error when forming $\rho(x_i)$. From [22, Equation (3.5)] and (3), we obtain

$$\big|\mathrm{fl}(x_i^\top Ax_i) - x_i^\top\,\mathrm{fl}(Ax_i)\big| \le \gamma_n^h\|x_i\|_2\|\mathrm{fl}(Ax_i)\|_2 \le \gamma_n^h(1+\epsilon_A)\|A\|_2\|x_i\|_2^2.$$

Thus, we have

$$\begin{aligned}\big|\mathrm{fl}(x_i^\top Ax_i) - x_i^\top Ax_i\big| &\le \big|\mathrm{fl}(x_i^\top Ax_i) - x_i^\top\,\mathrm{fl}(Ax_i)\big| + \big|x_i^\top\,\mathrm{fl}(Ax_i) - x_i^\top Ax_i\big| \\ &\le \gamma_n^h(1+\epsilon_A)\|A\|_2\|x_i\|_2^2 + \|x_i^\top\Delta Ax_i\|_2 \\ &\le \big(\gamma_n^h(1+\epsilon_A) + \epsilon_A\big)\|A\|_2\|x_i\|_2^2.\end{aligned}$$

Combined with $\mathrm{fl}(x_i^\top x_i) = x_i^\top x_i(1+\delta_1)$ for $|\delta_1| \le \gamma_n^h$, this implies for $\rho(x_i) = x_i^\top Ax_i/(x_i^\top x_i)$ that there is $|\delta_2| \le \boldsymbol{u}_h$ such that

$$\begin{aligned}\big|\mathrm{fl}(\rho(x_i)) - \rho(x_i)\big| &= \left|\frac{\mathrm{fl}(x_i^\top Ax_i)}{x_i^\top x_i(1+\delta_1)}(1+\delta_2) - \frac{x_i^\top Ax_i}{x_i^\top x_i(1+\delta_1)}(1+\delta_2+\delta_1-\delta_2)\right| \\ &\le \left|\frac{(\mathrm{fl}(x_i^\top Ax_i) - x_i^\top Ax_i)(1+\delta_2)}{x_i^\top x_i(1+\delta_1)}\right| + \left|\frac{x_i^\top Ax_i(\delta_2-\delta_1)}{x_i^\top x_i(1+\delta_1)}\right| \\ &\le \frac{\big|\mathrm{fl}(x_i^\top Ax_i) - x_i^\top Ax_i\big|}{x_i^\top x_i}\left|\frac{1+\delta_2}{1+\delta_1}\right| + \rho(x_i)\left|\frac{\delta_2-\delta_1}{1+\delta_1}\right| \\ &\le \Big(\gamma_n^h(1+\epsilon_A) + \epsilon_A\Big)\frac{1}{1-2n\boldsymbol{u}_h}\|A\|_2 + \frac{(1+n)\boldsymbol{u}_h}{1-2n\boldsymbol{u}_h}\rho(x_i) \\ &\le \big(\gamma_n^h(1+\epsilon_A) + \epsilon_A + (n+1)\boldsymbol{u}_h\big)\frac{\|A\|_2}{1-2n\boldsymbol{u}_h}. \qquad (4)\end{aligned}$$

The vector subtraction and scaling when forming $r_i = Ax_i - \rho(x_i)x_i$ yield two diagonal matrices E and E_1 such that

$$\hat{r}_i = (I+E)\big((A+\Delta A)x_i - (I+E_1)\,\mathrm{fl}(\rho(x_i))x_i\big)$$

$$= (I+E)(r_i + Fx_i),$$
$$F := \Delta A - \mathrm{fl}(\rho(x_i))E_1 - \big(\mathrm{fl}(\rho(x_i)) - \rho(x_i)\big)I.$$

where $\|E\|_2 \leq \boldsymbol{u}_h$ and $\|E_1\|_2 \leq \boldsymbol{u}_h$. Combined with (4), this concludes the proof because

$$\|F\|_2 \leq \boldsymbol{u}_h\|A\|_2 + (1+\boldsymbol{u}_h)\big(\gamma_n^h(1+\epsilon_A) + \epsilon_A + (n+1)\boldsymbol{u}_h\big)\frac{1}{1-2n\boldsymbol{u}_h}\|A\|_2 + \epsilon_A\|A\|_2$$
$$\leq \Big(\big(\gamma_n^h + \epsilon_A + \gamma_n^h\epsilon_A + +(n+1)\boldsymbol{u}_h\big)\frac{1+\boldsymbol{u}_h}{1-2n\boldsymbol{u}_h} + \epsilon_A + \boldsymbol{u}_h\Big)\|A\|_2.$$

□

We model the inexact application of the preconditioner T to $\hat{r}_i$ in the iteration (2) with the equation

$$\hat{w}_i = T_E\hat{r}_i, \tag{5}$$

where T_E depends on the choice of preconditioner T and the way to compute $T\hat{r}_i$. Note that T_E also depends on i.

Theorem 2. *Consider the setting of Lemma 1 and (5). If* $\lambda_1 < \rho(x_i) < \lambda_2$ *and*

$$\gamma := \|I - A^{1/2}T_EA^{1/2}\|_2 + \gamma_2^h\|T_E\|_2\|A\|_2 + \beta(x_i)\big(\boldsymbol{u}_h + (1+\gamma_2^h)\epsilon_r\|T_E\|_2\|A\|_2\big) < 1,$$

with

$$\beta(x_i) = \max\left\{\frac{\sqrt{\lambda_1\lambda_n}}{\rho(x_i)-\lambda_1}, \frac{\sqrt{\lambda_2\lambda_n}}{\lambda_2-\rho(x_i)}\right\},$$

then the computed result $\hat{x}_{i+1}$ *of the PINVIT iteration (2) satisfies*

$$\frac{\rho(\hat{x}_{i+1})-\lambda_1}{\lambda_2-\rho(\hat{x}_{i+1})} \leq \Big(\gamma + (1-\gamma)\frac{\lambda_1}{\lambda_2}\Big)^2\frac{\rho(x_i)-\lambda_1}{\lambda_2-\rho(x_i)}.$$

Proof By (2), (5), and Lemma 1, there exists a diagonal matrix E_0 (coming from the vector addition) such that $\|E_0\| \leq \boldsymbol{u}_h$ and

$$\hat{x}_{i+1} = (I+E_0)\big(x_i - T_E(I+E)(Ax_i - \rho(x_i)x_i + Fx_i)\big)$$
$$= x_i - \big(\tilde{T}_E(A-\rho(x_i)I) - E_0 + \tilde{T}_EF\big)x_i,$$

where $\tilde{T}_E = (I+E_0)T_E(I+E)$. Setting $A_\rho = A - \rho(x_i)I$ and using that $x_i = A_\rho^{-1}r_i$, it follows that

$$\hat{x}_{i+1} = x_i - \big(\tilde{T}_E - E_0A_\rho^{-1} + \tilde{T}_EFA_\rho^{-1}\big)r_i,$$

which takes the form of PINVIT with a perturbed preconditioner. This allows us to apply [6, Theorem 2.1], which requires the preconditioner to satisfy

$$\big\|I - A^{1/2}(\tilde{T}_E - E_0A_\rho^{-1} + \tilde{T}_EFA_\rho^{-1})A^{1/2}\big\|_2 < 1. \tag{6}$$

We now treat the different terms involved in (6) separately. First, we have

$$\|I - A^{1/2}\tilde{T}_EA^{1/2}\|_2 \leq \|I - A^{1/2}T_EA^{1/2}\|_2 + \|A^{1/2}(\tilde{T}_E - T_E)A^{1/2}\|_2$$

$$\leq \|I - A^{1/2}T_E A^{1/2}\|_2 + \gamma_2^h \|T_E\|_2 \|A\|_2. \tag{7}$$

By the assumptions, the spectral radius of $A_\rho^{-1}A^{1/2}$ is given by

$$\max\left\{\frac{\sqrt{\lambda_1}}{\rho(x_i)-\lambda_1}, \frac{\sqrt{\lambda_2}}{\lambda_2-\rho(x_i)}\right\}.$$

This allows us to bound the other terms in (6) as follows:

$$\begin{aligned}
&\|A^{1/2}(-E_0A_\rho^{-1} + \tilde{T}_E F A_\rho^{-1})A^{1/2}\|_2 \\
&\leq \|A^{1/2}\|_2 \|A_\rho^{-1}A^{1/2}\|_2 \big(\|E_0\|_2 + \|\tilde{T}_E\|_2 \|F\|_2\big) \\
&\leq \|A^{1/2}\|_2 \|A_\rho^{-1}A^{1/2}\|_2 \big(\boldsymbol{u}_h + (1+\gamma_2^h)\epsilon_r \|T_E\|_2 \|A\|_2\big) \\
&= \beta(x_i)\big(\boldsymbol{u}_h + (1+\gamma_2^h)\epsilon_r \|T_E\|_2 \|A\|_2\big).
\end{aligned}$$

Together with (7), this implies that the left-hand side of (6) is bounded by $\gamma < 1$ and the statement of the theorem follows from [6, Theorem 2.1]. □

Remark. We remark that the conclusion of Theorem 2 does *not* imply that $\rho(x_i) - \lambda_1$ can eventually drop below machine precision. For the relative error $\big(\rho(x_i) - \lambda_1\big)/\big(\lambda_2 - \rho(x_i)\big)$ to be reduced by the factor

$$\left(\gamma + (1-\gamma)\frac{\lambda_1}{\lambda_2}\right)^2 = \left(\frac{\lambda_1}{\lambda_2} + \left(1 - \frac{\lambda_1}{\lambda_2}\right)\gamma\right)^2$$

during the ith iteration, Theorem 2 requires that

$$\lambda_1 < \rho(x_i) < \lambda_2 - \frac{\boldsymbol{u}_h + (1+\gamma_2^h)\epsilon_r \|T_E\|_2 \|A\|_2}{1 - \big\|I - A^{1/2}T_E A^{1/2}\big\|_2 - \gamma_2^h \|T_E\|_2 \|A\|_2} \sqrt{\lambda_2 \lambda_n}$$

holds. This reduction takes place until a Rayleigh quotient $\rho(\hat{x})$ for an iterate $\hat{x}$ is produced for which

$$\frac{\rho(\hat{x}) - \lambda_1}{\sqrt{\lambda_1 \lambda_n}} \leq \frac{\boldsymbol{u}_h + (1+\gamma_2^h)\epsilon_r \|T_E\|_2 \|A\|_2}{1 - \|I - A^{1/2}T_E A^{1/2}\|_2 - \gamma_2^h \|T_E\|_2 \|A\|_2}.$$

For reasonable choices of T_E, this means that the error is reduced until it reaches the level of working precision.

The quantity $\|I - A^{1/2}T_E A^{1/2}\|_2$ critically determines the convergence rate of PINVIT. The following lemma provides an estimate if T_E corresponds to applying A^{-1} in low precision via the Cholesky factorization.

Lemma 3. *Suppose that the application of the preconditioner T in one step of PINVIT (2) is implemented by applying A^{-1} in low precision, via performing the Cholesky factorization of A followed by forward and backward substitution. If $\epsilon_T := 4n(3n+1)\kappa(A)\boldsymbol{u}_l < 1$, where $\kappa(A) = \|A\|_2\|A^{-1}\|_2$ and $\boldsymbol{u}_l$ denotes unit roundoff in low precision, then*

$$\big\|I - A^{1/2}T_E A^{1/2}\big\|_2 \leq \frac{\epsilon_T}{1-\epsilon_T}.$$

Proof Using [22, Theorem 10.4], there exists a symmetric matrix E_0 such that

$$\hat{w}_i = \mathrm{fl}(T\hat{r}_i) = (A + E_0)^{-1}\hat{r}_i, \qquad \|E_0\|_2 \leq 4n(3n+1)\boldsymbol{u}_l\|A\|_2,$$

which means $T_E = (A + E_0)^{-1}$ and, moreover,

$$A^{1/2}T_E A^{1/2} = \left(I + A^{-1/2}E_0A^{-1/2}\right)^{-1}.$$

Then by $\left\|A^{-1/2}E_0A^{-1/2}\right\|_2 \leq 4n(3n+1)\kappa(A)\boldsymbol{u}_l < 1$, we have

$$\left(I + A^{-1/2}E_0A^{-1/2}\right)^{-1} = \sum_{i=0}^{\infty}\left(-A^{-1/2}E_0A^{-1/2}\right)^i.$$

Thus, it holds that

$$\begin{aligned}\left\|I - A^{1/2}T_E A^{1/2}\right\|_2 &= \left\|I - \left(I + A^{-1/2}E_0A^{-1/2}\right)^{-1}\right\|_2 \\ &\leq \sum_{i=1}^{\infty}\left\|A^{-1/2}E_0A^{-1/2}\right\|_2^i \\ &\leq \frac{\epsilon_T}{1-\epsilon_T}.\end{aligned}$$

□

5 Numerical experiments

In this section, we present numerical results for our mixed precision LOBPCG algorithm. In our tests, the working precision is IEEE double precision and the lower precision is IEEE single precision. Most tests are performed on a Linux server equipped with two twelve-core Intel Xeon E5-2670 v3 2.30 GHz CPUs and two Nvidia GeForce GTX 1080 GPUs. The tests in Section 5.5 also use an Nvidia A30 GPU. There are 128 GB of main memory on the CPUs and 11,178.6 MB of main memory on each GPU. Our program uses only one GPU and one thread on the CPU.

5.1 Experiment settings

In our experiments we compute a few smallest eigenvalues and the corresponding eigenvectors of Hermitian matrices using the LOBPCG algorithm. The following variants of the LOBPCG algorithm are tested:

1. **DLOBPCG-dchol**: LOBPCG algorithm performed entirely in double precision.
2. **DLOBPCG-schol**: LOBPCG algorithm performed in double precision, except for single precision preconditioning.
3. **MPLOBPCG-schol**: mixed precision LOBPCG algorithm (Algorithm 3) with single precision preconditioning and initial guess computed by the single precision LOBPCG algorithm; mixed precision orthogonalization (Algorithm 2) is also used.

When to computing k eigenpairs, we run LOBPCG algorithm with a block size that is about 50% larger in order to enhance robustness. The algorithm

Table 1 Libraries used in our implementation.

	Cholesky	`TRSM`	mat–vec	others
CPU/sparse	CHOLMOD	CHOLMOD	MKL	LAPACK
GPU/sparse	CHOLMOD	cuSPARSE	cuSPARSE	MAGMA
CPU/dense	LAPACK	LAPACK	LAPACK	LAPACK
GPU/dense	MAGMA	MAGMA	MAGMA	MAGMA

terminates once the k smallest eigenvalues and the corresponding eigenvectors converge. The convergence criterion is

$$\|AX(:,j) - \Theta(j,j)X(:,j)\|_2 \leq \texttt{tol} \cdot (\|A\|_2 + |\Theta(j,j)|)\|X(:,j)\|_2, \qquad (8)$$

where $\|A\|_2$ is estimated through $\|A\|_2 \approx \|\Omega A\|_{\mathsf{F}}/\|\Omega\|_{\mathsf{F}}$ using a Gaussian random matrix $\Omega \in \mathbb{C}^{m\times n}$ with $m \ll n$. The threshold `tol` in (8) is set to 10^{-12} for all these three algorithms, and is 5×10^{-6} when computing a good initial guess for MPLOBPCG-schol.

In our tests, we use $\Pi^{-*}L^{-*}L^{-1}\Pi^{-1}$ as the preconditioner for Algorithm 3, where Π is a permutation matrix, and L is the (pivoted) Cholesky factor of A satisfying $\Pi^* A\Pi = LL^*$ computed in single precision. The preconditioning stage in DLOBPCG-schol/MPLOBPCG-schol is to compute $W_{\texttt{lower}} = \Pi L^{-*}L^{-1}\Pi^* \texttt{lower}(R)$ by solving two triangular systems in single precision. In practice, we apply Π to the given matrix A instead of applying Π to $\texttt{lower}(R)$ in each iteration. We can benefit from it if A is sufficiently sparse or the convergence of LOBPCG is not too rapid (i.e., it takes many iterations to converge).

We test the LOBPCG algorithm for both sparse matrices and dense matrices. Table 1 summarizes the software libraries used under different settings. The CHOLMOD package [23] can compute sparse Cholesky factorization on both CPU and GPU, while triangular linear solvers are only supported only in CPU. Note that CHOLMOD was developed only for double precision arithmetic; we have derived a single precision version for the purpose of our tests.

5.2 Advantage of mixed precision orthogonalization

Before discussing the LOBPCG algorithm, we first report the run time and savings of the mixed precision orthgonalization algorithm (i.e., Algorithm 2) in Table 2. We can see that for tall-skinny matrices Algorithm 2 can reduce the run time by a factor of 1/4–1/3 compared to `DGEQRF` in cuSOLVER. Thus, it is worth using this mixed approach for orthogonalization.

5.3 Tests for sparse matrices

We choose six sparse positive definite matrices from from the SuiteSparse Matrix Collection.[1] Table 3 shows the information of these sparse matrices.

[1] https://sparse.tamu.edu

Table 2 Run time of Algorithm 2 in seconds.

(n, k)	Run time of Algorithm 2	Run time of `DGEQRF`	Savings
$(40000, 20)$	1.544×10^{-3}	2.078×10^{-3}	25.7%
$(40000, 25)$	1.907×10^{-3}	2.823×10^{-3}	32.5%
$(40000, 30)$	2.371×10^{-3}	3.579×10^{-3}	33.7%
$(40000, 35)$	3.586×10^{-3}	4.886×10^{-3}	26.6%
$(40000, 40)$	4.000×10^{-3}	5.726×10^{-3}	30.1%
$(40000, 45)$	4.495×10^{-3}	6.862×10^{-3}	34.5%

Table 3 Information of sparse testing matrices.

Name	Size	NNZ	Sparsity	NNZ of L
obstclae	40,000	197,608	1.235×10^{-4}	1,561,880
shallow_water2	81,920	327,680	4.883×10^{-5}	3,483,014
Dubcova2	65,025	1,030,225	2.437×10^{-4}	3,804,558
Dubcova3	146,689	3,636,643	1.690×10^{-4}	7,409,077
finan512	74,752	596,992	1.068×10^{-4}	3,376,835
2D-Laplace	25,000	114,990	1.840×10^{-4}	466,491

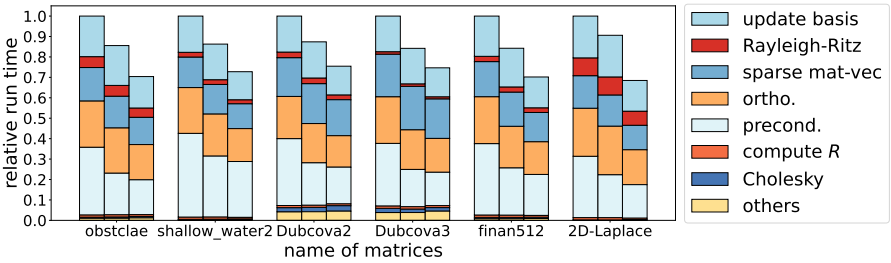

Fig. 1 Tests for real sparse matrices on CPU. For each matrix, the three columns from left to right represent the result of DLOBPCG-dchol, DLOBPCG-schol, and MPLOBPCG-schol, respectively.

We compute 30 eigenpairs using a randomly generated initial guess with 45 columns for each matrix, and report the relative run time, which is the ratio of the wall clock time of a solver over the wall clock time of DLOBPCG-dchol.

Figures 1 and 2, respectively, show the relative run time on CPU and GPU. For all test cases, preconditioning in single precision reduces the execution time of the LOBPCG algorithm. Using an initial guess computed by the single precision LOBPCG algorithm and adopting mixed precision orthogonalization makes the algorithm more efficient. Compared to DLOBPCG-dchol, MPLOBPCG-schol is about 1.43× faster on CPU, and is about 1.67× faster on GPU.

We should also mention that the number of iterations for different variants of the LOBPCG algorithms are similar, though they are not shown in the figures. Sometimes MPLOBPCG-schol can require fewer iterations to converge because there is a restart when we use the lower precision result as the initial guess. For instance, the total iterations of DLOBPCG-dchol, DLOBPCG-schol and MPLOBPCG-schol are 533, 534, and 461, respectively, for the 2D-Laplace matrix.

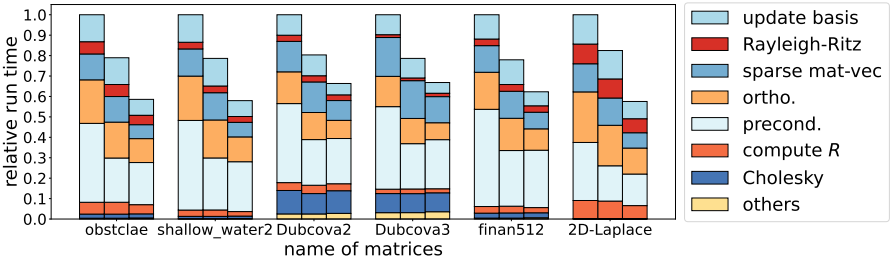

Fig. 2 Tests for real sparse matrices on GPU. For each matrix, the three columns from left to right represent the result of DLOBPCG-dchol, DLOBPCG-schol, and MPLOBPCG-schol, respectively.

5.4 Tests for dense matrices

We also test the LOBPCG algorithm for a few dense matrices which are popular in machine learning. These dense matrices are kernel matrices generated by certain kernel functions as follows. Let $x_1, x_2, \ldots, x_n \in \mathbb{R}^n$ be uniform random vectors generated by `XLARNV` from LAPACK. We construct a matrix K by applying the Gaussian kernel function

$$K_{ij} = k(x_i, x_j) = \mathrm{e}^{-\|x_i - x_j\|_2/2}.$$

Similarly, we can apply the polynomial kernel function

$$k(x_i, x_j) = (x_i^\top x_j + 1)^3$$

to construct another kernel matrix. Using two sets of random vectors $\{x_1, x_2, \ldots, x_n\}$ and $\{y_1, y_2, \ldots, y_n\}$ in $\mathbb{R}^n$, we also construct complex kernel matrices through

$$K_{ij} = k(x_i, x_j) + k(y_i, y_j) + \mathrm{i}\big(k(x_i, y_j) - k(y_i, x_j)\big),$$

where $k(\cdot, \cdot)$ is either the Gaussian kernel function or the polynomial kernel function.

We choose $n \in \{1024, 2048, 4096, 8192\}$ in our experiments, and compute $5n/1024$ smallest eigenvalues and the corresponding eigenvectors. The rank of initial guess is chosen as $8n/1024$ accordingly. Figures 3, 4, and 5 show the relative run time of different variants of the LOBPCG algorithm. For real matrices, MPLOBPCG-schol achieves 1.67× and 2× speedup compared to DLOBPCG-dchol on CPU and GPU, respectively. The speedup is higher than that for sparse matrices, because dense matrices are more compute-intensive. The benefit for mixed precision approaches is more significant for complex matrices—the speedup becomes over 2.5× and up to 5× on GPU.

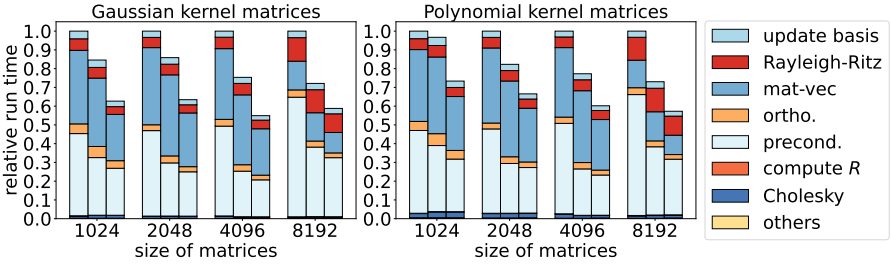

Fig. 3 Tests for real dense kernel matrices on CPU. For each matrix, the three columns from left to right represent the result of DLOBPCG-dchol, DLOBPCG-schol, and MPLOBPCG-schol, respectively.

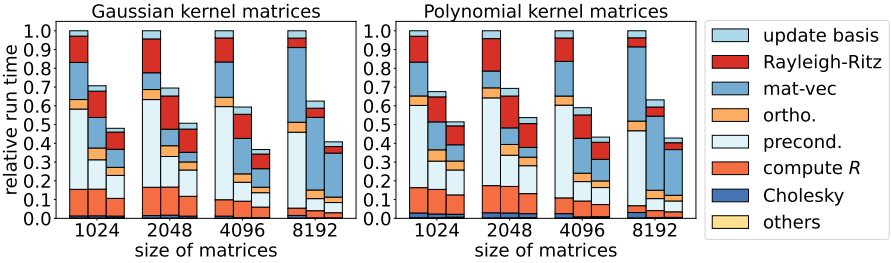

Fig. 4 Tests for real dense kernel matrices on GPU. For each matrix, the three columns from left to right represent the result of DLOBPCG-dchol, DLOBPCG-schol, and MPLOBPCG-schol, respectively.

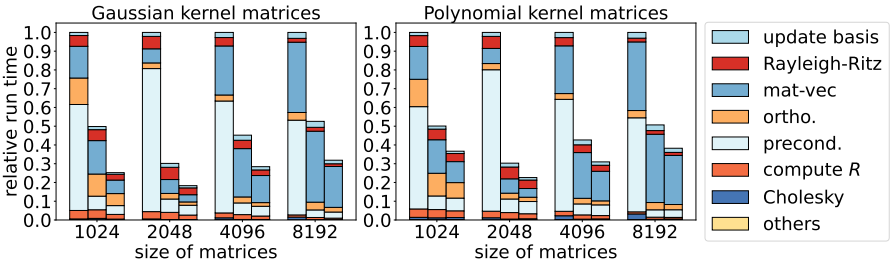

Fig. 5 Tests for complex dense kernel matrices on GPU. For each matrix, the three columns from left to right represent the result of DLOBPCG-dchol, DLOBPCG-schol, and MPLOBPCG-schol, respectively.

5.5 Tests on different GPUs

By far our tests are performed with an Nvidia GeForce GTX 1080 GPU, which is a consumer-grade GPU. In fact, there are two different types of GPU—consumer-grade and server-grade. Compared to consumer-grader GPUs server-grade GPUs usually have better hardware support for double precision arithmetic. Hence the performance difference between single and double precision arithmetic is larger on consumer-grade GPUs.

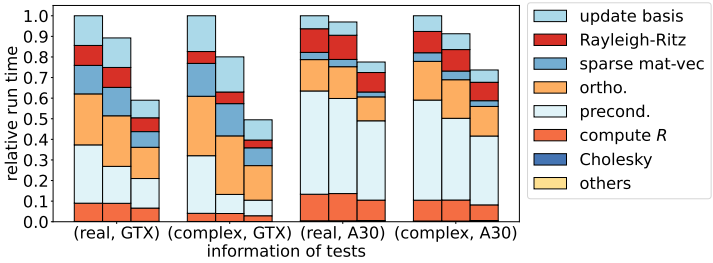

Fig. 6 Tests for real and complex 2D-Laplace matrices in different GPUs. For each case, the three columns from left to right represent the result of DLOBPCG-dchol, DLOBPCG-schol, and MPLOBPCG-schol, respectively.

In the following we report some results collected from runs on an Nvidia A30 GPU, which is a server-grade one. We use the matrix 2D-Laplace in this test. By perturbing off-diagonal entries of this matrix by $\pm 10^{-16} \cdot \mathrm{i}$, we also obtain a Hermtian positive definite matrix for testing complex arithmetic. From Figure 6, it can be seen that single precision has limited advantage over double precision on this server-grade GPU. Though MPLOBPCG-schol still achieves about 1.3× speedup compared to DLOBPCG-dchol, the benefit for adopting single precision arithmetic is much lower than that on NVIDIA GeForce GTX-1080 which is a consumer-grade GPU.

6 Conclusion

In this paper, we have proposed a mixed precision LOBPCG algorithm with a preconditioner based on a (sparse) Cholesky factorization. Both the initial guess and the preconditioner are computed in reduced precision. This largely improves the performance while it only has marginal impact on convergence. In our mixed precision LOBPCG algorithm, orthogonalization is also performed in a mixed precision manner to further improve performance. We analyze the rounding error of the PINVIT algorithm, which can be viewed as a simplified version of the LOBPCG algorithm, to confirm that our mixed precision algorithm is as accurate as the fixed precision one. Numerical experiments illustrate that adopting mixed precision arithmetic can significantly accelerate the execution of the LOBPCG algorithm on both CPUs and GPUs.

Declarations

- Ethics approval: Not Applicable.
- Availability of data and materials: the results/data/figures in this manuscript have not been published elsewhere, nor are they under consideration (from you or one of your Contributing Authors) by another publisher. All of the material is owned by the authors and/or no permissions are required.

- Competing interests: we declare that the authors have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.
- Funding: Yuxin Ma is partially supported by the State Scholarship Fund of China Scholarship Council (CSC) under Grant No. 202106100093, National Key R&D Program of China under Grant No. 2021YFA1003305 and National Natural Science Foundation of China under Grant No. 71991471. Meiyue Shao is partially supported by by the National Natural Science Foundation of China under grant No. 11971118.
- Authors' contributions: these authors contributed equally to this work.
- Acknowledgments: The authors thank Erin Carson for helpful discussions. Part of this work was performed when the second author was visiting EPF Lausanne in 2022.

References

[1] Balcan, D., Gonçalves, B., Hu, H., Ramasco, J.J., Colizza, V., Vespignani, A.: Modeling the spatial spread of infectious diseases: the GLobal Epidemic and Mobility computational model. J. Comput. Sci. **1**(3), 132–145 (2010). https://doi.org/10.1016/j.jocs.2010.07.002

[2] Knyazev, A.: Recent implementations, applications, and extensions of the locally optimal block preconditioned conjugate gradient method (LOBPCG). arXiv:1708.08354 (2017)

[3] Saad, Y.: Numerical Methods for Large Eigenvalue Problems: Revised Edition. SIAM, Philadelphia, PA, USA (2011)

[4] Neymeyr, K.: A geometric theory for preconditioned inverse iteration I: Extrema of the Rayleigh quotient. Linear Algebra Appl. **322**(1-3), 61–85 (2001). https://doi.org/10.1016/S0024-3795(00)00239-1

[5] Neymeyr, K.: A geometric theory for preconditioned inverse iteration applied to a subspace. Math. Comp. **71**(237), 197–216 (2002). https://doi.org/10.1090/S0025-5718-01-01357-6

[6] Argentati, M., Knyazev, A., Neymeyr, K., Ovtchinnikov, E., Zhou, M.: Convergence theory for preconditioned eigenvalue solvers in a nutshell. Found. Comput. Math. **17**, 713–727 (2017). https://doi.org/10.1007/s10208-015-9297-1

[7] Knyazev, A.V.: Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM J. Sci. Comput. **23**(2), 517–541 (2001). https://doi.org/10.1137/S1064827500366124

[8] Abdelfattah, A., Anzt, H., Boman, E.G., Carson, E., Cojean, T., Dongarra, J., Fox, A., Gates, M., Higham, N.J., Li, X.S., Loe, J., Luszczek, P., Pranesh, S., Rajamanickam, S., Ribizel, T., Smith, B.F., Swirydowicz, K., Thomas, S., Tomov, S., Tsai, Y.M., Yang, U.M.: A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. Int. J. High Perform. Comput. Appl. **35**(4), 344–369 (2021). https://doi.org/10.1177/10943420211003313

[9] Higham, N.J., Mary, T.: Mixed precision algorithms in numerical linear algebra. Acta Numer. **31**, 347–414 (2022). https://doi.org/10.1017/S0962492922000022

[10] Carson, E., Higham, N.J.: Accelerating the solution of linear systems by iterative refinement in three precisions. SIAM J. Sci. Comput. **40**(2), 817–847 (2018). https://doi.org/10.1137/17M1140819

[11] Ogita, T., Aishima, K.: Iterative refinement for symmetric eigenvalue decomposition. Japan J. Indust. Appl. Math. **35**(3), 1007–1035 (2018). https://doi.org/10.1007/s13160-018-0310-3

[12] Ogita, T., Aishima, K.: Iterative refinement for symmetric eigenvalue decomposition II: clustered eigenvalues. Japan J. Indust. Appl. Math. **36**(2), 435–459 (2019). https://doi.org/10.1007/s13160-019-00348-4

[13] Ogita, T., Aishima, K.: Iterative refinement for singular value decomposition based on matrix multiplication. J. Comput. Appl. Math. **369**, 112512 (2020). https://doi.org/10.1016/j.cam.2019.112512

[14] Bujanović, Z., Kressner, D., Schröder, C.: Iterative refinement of Schur decompositions. Numer. Algorithms **92**(1), 247–267 (2023). https://doi.org/10.1007/s11075-022-01327-6

[15] Gao, W., Ma, Y., Shao, M.: A mixed precision Jacobi SVD algorithm. arXiv:2209.04626 (2022)

[16] Dongarra, J.J.: Algorithm 589: SICEDR: A FORTRAN subroutine for improving the accuracy of computed matrix eigenvalues. ACM Trans. Math. Software **8**(4), 371–375 (1982). https://doi.org/10.1145/356012.356016

[17] Golub, G.H., Van Loan, C.F.: Matrix Computations, 4th edn. Johns Hopkins University Press, Baltimore, MD, USA (2013)

[18] Duersch, J.A., Shao, M., Yang, C., Gu, M.: A robust and efficient implementation of LOBPCG. SIAM J. Sci. Comput. **40**(5), 655–676 (2018). https://doi.org/10.1137/17M1129830

[19] Hetmaniuk, U., Lehoucq, R.: Basis selection in LOBPCG. J. Comput. Phys. **218**(1), 324–332 (2006). https://doi.org/10.1016/j.jcp.2006.02.007

[20] Yamazaki, I., Tomov, S., Dongarra, J.: Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs. SIAM J. Sci. Comput. **37**(3), 307–330 (2015). https://doi.org/10.1137/14M0973773

[21] Yamazaki, I., Tomov, S., Kurzak, J., Dongarra, J., Barlow, J.: Mixed-precision block Gram Schmidt orthogonalization. In: ScalA '15: Proceedings of the 6th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, pp. 2–128 (2015). https://doi.org/10.1145/2832080.2832082

[22] Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. SIAM, Philadelphia, PA, USA (2002)

[23] Chen, Y., Davis, T.A., Hager, W.W., Rajamanickam, S.: Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. ACM Trans. Math. Software **35**(3), 22–12214 (2008). https://doi.org/10.1145/1391989.1391995