High Efficiency Spectral Analysis and BLAS-3 Randomized QRCP with Low-Rank Approximations

by

Jed Alma Duersch

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Applied Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Fall 2015

High Efficiency Spectral Analysis and BLAS-3 Randomized QRCP with Low-Rank Approximations

Contents

Acknowledgments

I gratefully thank my coauthors, Dr. Meiyue Shao and Dr. Chao Yang, for their time and expertise. I would also like to thank Assistant Professor Lin Lin, Dr. Osni Marques, and Dr. Eugene Vecharynski for useful conversations regarding symmetric eigenvalues problems, Professor Laura Grigori, Professor James Demmel, and Chris Melgaard for conversations regarding QRCP.

I would also like to thank my family and friends for their support and encouragement. In particular, Dr. Valerie Morash, Giovanni Pinuellas, Roger Hanna, and Sonia Rogers for listening to me endlessly explain this work.

Finally, I would like to especially thank my coauthor and advisor, Professor Ming Gu, for his many patient hours of guidance and tireless support.

The purpose of this work is to improve stability and performance of selected matrix decompositions in numerical linear algebra. Chapter 1 examines the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) algorithm used to compute eigenvector-eigenvalue pairs for large sparse symmetric matrices or symmetric matrix pencils. Modifications are developed and analyzed to improve both performance and reliability. Numerical experiments demonstrate the final algorithm often operates several times faster than competing versions and succeeds in processing difficult matrices for which other versions fail.

Chapters 2 extends the work on symmetric eigenvalue problems by developing an algorithm specialized to resolve eigenpairs in the interior of the spectrum of a symmetric matrix. An new Spectral Target Residual Descent (STRD) algorithm is proposed based on LOBPCG. This algorithm is demonstrated to reduce the number of iterations required to achieve convergence by roughly half. The component subroutines of STRD also have generalized versions to process interior eigenvalues of symmetric matrix pencils.

Chapter 3 explores a variation on the QR decomposition with Column Pivoting using randomized-sampling to process pivoting decisions. Randomized QRCP eliminates the leading order of communication by processing column pivots on a much smaller sample matrix. This produces blocks of column pivots that are processed with BLAS-3 operations on the original matrix. Sample update formulations are explored that allow full matrices to be factorized without re-compressing the matrix after each block iteration. Related low-rank factorizations are also developed and compared in numerical experiments. These experiments show approximation quality for structured matrices to be comparable, and often better than, approximations based on QRCP. Furthermore, performance is shown to be nearly as good as QR without pivoting, which is often an order of magnitude faster than QRCP.

Chapter 1

A Robust and Efficient Implementation of LOBPCG

1.1 Contributions and results

This research develops stability and performance improvements for the LOBPCG symmetric iterative eigenvalue solver. Prior to this work, the most stable framework for implementing LOBPCG had been explored in the work of Hetmaniuk and Lehoucq [32]. Despite the increased stability of their approach, failure cases were still frequently observed due to remaining difficulties in implementation details of `ortho` [65].

1.1.1 Robustness and reliability

The first priority of this research is to resolve remaining robustness and reliability issues observed in other implementations of LOBPCG. The algorithm developed resolves instabilities in `ortho` and `svqb` which eliminates the primary point of failure—that is ill-conditioning of the search subspace basis—in other implementations of LOBPCG. Furthermore, this work identifies inadequacy of the standard convergence detection method: the relative residual norm threshold. An alternative convergence detection computation is proposed and analyzed that resolves test cases that otherwise fail to show convergence.

1.1.2 Performance and efficiency

The second priority of this work is to maximize performance so that the resulting implementation remains useful and practical for applications in research and development. These performance improvements include extending prior work on implicit update transformations for basis products. Implicit update error in product representations is analyzed in order to identify and avoid cases in which such updates create numerical instability. Safe mechanisms are proposed to reduce the costs associated with `ortho` and `svqb`. Furthermore, an alternative computation for aggregate basis updating is proposed to eliminate a second call to `ortho` that is needed in each iteration of the version by Hetmaniuk and Lehoucq. This alternative formulation enables another optimization that substantially reduces communication costs through a simultaneous matrix-multiply-and-update mechanism.

1.1.3 Results

Stability improvements allow difficult matrices to be reliably processed where other implementations fail. This algorithm also exhibits a significant improvement in performance. Typical test cases finish between one sixth and one third of the time required by competing high-performance subroutines. These improvements to LOBPCG make it an attractive and reliable tool for researchers dealing with very large sparse symmetric matrices in related fields of science and engineering. The substantial reduction in processing time and increased robustness of this method will allow scientists and engineers to make better use of numerical eigenvalue tests for large finite-element models and ultimately improve the rate of progress in research.

1.2 Introduction to LOBPCG

Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) [41] is a widely used algorithm for computing a few algebraically smallest (or largest) eigenvalues and the corresponding eigenvectors of a symmetric definite matrix pencil (A, B) in which A is symmetric and B is symmetric positive definite. When B is identity, the problem becomes a standard symmetric eigenvalue problem.

There are three main advantages of this method when compared with the classical Krylov subspace-based methods. First, LOBPCG can employ a good preconditioner when one is available which sometimes dramatically reduces computation time. Second, the three-term recurrence used by the algorithm keeps memory required relatively low making it possible to tackle problems at a very large scale. Third, because the algorithm is blocked it can be implemented efficiently on modern parallel computers. Nearly every component of the algorithm can be formulated in level-3 dense or sparse BLAS operations that are highly tuned for efficient parallel scaling.

However, it has been observed that the algorithm can breakdown or suffer from numerical instability when it is not implemented carefully. In particular, basis vectors forming the subspace from which the approximate solution to the eigenvalue problem is extracted can become linearly dependent. This problem becomes progressively worse when the number of eigenpairs to be computed is relatively large (e.g., hundreds or thousands.)

A strategy proposed in the work of Hetmaniuk and Lehoucq [32] addresses this issue. Their strategy is based on performing additional orthogonalization to ensure that the preconditioned gradient is numerically B-orthogonal to both the current and the previous approximations to the desired eigenvectors. However, this strategy can become expensive when the number of eigenpairs to be computed is relatively large. More importantly, reliability can be severely compromised due to numerical instability within the orthogonalization steps.

This chapter presents an efficient and reliable implementation of LOBPCG. We develop a number of techniques to significantly enhance the Hetmaniuk–Lehoucq (HL) orthogonalization strategy in both efficiency and reliability. We also adopt an alternative convergence criterion to ensure achievable error control in computed eigenvalue–eigenvector pairs.

This chapter is organized as follows. Section 1.3 describes the basic LOBPCG algorithm. In Section 1.4, we discuss numerical difficulties one may encounter in LOBPCG and the HL strategy for overcoming these difficulties. Techniques for further improving stability are discussed in Section 1.5. In Section 1.6 we cover additional techniques for improving all other aspects of LOBPCG. In Section 1.7 error analysis is developed to justify these improvements. Section 1.8 presents numerical experimental results to illustrate the effectiveness of these techniques.

1.3 The basic LOBPCG algorithm

We denote the eigenvalues of the symmetric definite pencil (A, B) arranged in an increasing order by $\lambda_1 \leq \lambda_2 \leq \cdots \lambda_n$. Their corresponding eigenvectors are denoted by x_1, x_2, ..., x_n. The first $k \leq n$ eigenvectors and eigenvalues are

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_k \end{bmatrix} \quad \text{and} \quad \Lambda = \mathtt{diag}\left(\begin{bmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_k \end{bmatrix}\right)$$

$$\text{satisfying} \quad AX = BX\Lambda.$$

It is well known that X is the solution to the trace minimization problem

$$\min_{X^T BX = I} \operatorname{trace}(X^T AX). \tag{1.1}$$

The LOBPCG algorithm developed by Knyazev seeks to solve (1.1) by using the following eigenvector approximation update formula

$$X^{(i+1)} = \left[\begin{array}{ccc} X^{(i)} & W^{(i)} & P^{(i)} \end{array}\right] \left[\begin{array}{c} C_1^{(i+1)} \\ C_2^{(i+1)} \\ C_3^{(i+1)} \end{array}\right].$$

Parenthetical superscript indices indicate the matrix is stored in an array that will be overwritten by subsequent iterations. $W^{(i)}$ is the preconditioned gradient of the Lagrangian

$$\mathcal{L}(X, \Lambda) = \mathrm{trace}(X^T A X) - \mathrm{trace}\left[(X^T B X - I)\Lambda\right] \tag{1.2}$$

associated with (1.1) at $X^{(i)}$,

$$W^{(i)} = K^{-1}(AX^{(i)} - BX^{(i)}\Theta^{(i)}) \quad \text{with} \quad \Theta^{(i)} = X^{(i)T} A X^{(i)},$$

where K is any preconditioner. $P^{(i)}$ is an aggregated update direction from previous searches recursively defined as

$$P^{(i+1)} = \left[\begin{array}{cc} W^{(i)} & P^{(i)} \end{array}\right] \left[\begin{array}{c} C_2^{(i+1)} \\ C_3^{(i+1)} \end{array}\right].$$

Coefficient matrices $C_1^{(i+1)}$, $C_2^{(i+1)}$, and $C_3^{(i+1)}$ are determined at each step of LOBPCG by solving the constrained minimization problem (1.1) within the subspace $\mathcal{S}^{(i)}$ spanned by $X^{(i)}$, $W^{(i)}$, and $P^{(i)}$. That is,

$$\left(S^{(i)T} A S^{(i)}\right) C^{(i+1)} = \left(S^{(i)T} B S^{(i)}\right) C^{(i+1)} \Theta^{(i+1)}. \tag{1.3}$$

where $S^{(i)}$ is a matrix whose columns are a basis of $\mathcal{S}^{(i)}$ which is constructed

$$S^{(i)} = \left[\begin{array}{ccc} X^{(i)} & W^{(i)} & P^{(i)} \end{array}\right] \quad \text{with corresponding} \quad C^{(i+1)} = \left[\begin{array}{cc} C_1^{(i+1)} & C_{1\perp}^{(i+1)} \\ C_2^{(i+1)} & C_{2\perp}^{(i+1)} \\ C_3^{(i+1)} & C_{3\perp}^{(i+1)} \end{array}\right].$$

The leading k columns of $C^{(i+1)}$ form $C_x^{(i+1)} = [C_1^{(i+1)}; C_2^{(i+1)}; C_3^{(i+1)}]$ which are the components used to compute $X^{(i+1)}$. Remaining columns give the orthogonal complement within the search subspace.

$\Theta^{(i+1)}$ is a diagonal matrix containing approximations to the desired eigenvalues. If k smallest eigenpairs are sought then eigenvalues are sorted in ascending order. Otherwise if the largest eigenpairs are sought the order is reversed. Solving the projected eigenvalue problem (1.3) is often referred to as the *Rayleigh–Ritz procedure*. Combining these steps produces Algorithm 1, the basic LOBPCG algorithm. We leave details of convergence control to section 1.5.

Algorithm 1 The basic LOBPCG algorithm

```
Input:
    $X^{(0)}$ is $m \times n_x$ matrix of initial approximate eigenvectors.
    $n_v \leq n_x$ is the number of converged eigenvectors requested.
    $\tau$ is the threshold used to determine eigenpair convergence.
Output:
    $X$ is $m \times n_v$ matrix of approximate eigenvectors.
    $\Lambda$ is $n_v \times n_v$ diagonal matrix of approximate eigenvalues.
 1: function $[X, \Lambda]$=lobpcgKnyazev($X^{(0)}, n_v, \tau$)
 2:     $[C^{(1)}, \Theta^{(1)}] =$ RayleighRitz($X^{(0)}$);
 3:     $X^{(1)} = X^{(0)}C^{(1)}$;
 4:     $R^{(1)} = AX^{(1)} - BX^{(1)}\Theta^{(1)}$;
 5:     $P^{(1)} = []$;
 6:     do $i = 1, 2, \ldots$
 7:         $W^{(i)} = K^{-1}R^{(i)}$;
 8:         $S^{(i)} = \left[X^{(i)}, W^{(i)}, P^{(i)}\right]$;
 9:         $[C^{(i+1)}, \Theta^{(i+1)}] =$ RayleighRitz($S^{(i)}$);
10:         $X^{(i+1)} = S^{(i)}C^{(i+1)}(:, 1 : n_x)$;
11:         $R^{(i+1)} = AX^{(i+1)} - BX^{(i+1)}\Theta^{(i+1)}$;
12:         $P^{(i+1)} = S^{(i)}(:, n_x + 1 : \text{end})C^{(i+1)}(n_x + 1 : \text{end}, :)$;
13:         Determine number of converged eigenpairs $n_c$.
14:     while $n_c < n_v$
15:     Return converged eigenpairs in $X$ and $\Lambda$;
16: end function
```

1.4 Numerical stability and basis selection

In Algorithm 1, the Rayleigh–Ritz procedure for solving (1.3) is a critical point of numerical instability unless it is implemented carefully due to the fact that the projection S^TBS can be ill-conditioned or rank deficient. This holds regardless of the conditioning of B. The standard Rayleigh–Ritz procedure is outlined in Algorithm 2.

When S is not B-orthonormal, we must first perform a Cholesky factorization of S^TBS to obtain an upper triangular factor R that is used to transform the generalized eigenvalue problem into a standard eigenvalue problem

$$R^TR = S^TBS \quad \text{and} \quad \left[R^{-T}\left(S^TAS\right)R^{-1}\right]Z = Z\Theta. \tag{1.4}$$

This allows us to solve $C = R^{-1}Z$.

Algorithm 2 Rayleigh–Ritz analysis

```
Input:
    $S$ is $m \times n_s$ matrix basis for the search subspace.
    *Columns must be linearly independent and well-conditioned with respect to the metric $B$.
Output:
    $C$ is $n_s \times n_s$ solution components of $S$.
    $\Theta$ is $n_s \times n_s$ diagonal matrix of Rayleigh quotients.
    These satisfy $C^T(S^TBS)C = I_{n_s}$ and $C^T(S^TAS)C = \Theta$.
1: function $[C, \Theta]$=RayleighRitz($S$)
2:     $D = \left(\mathtt{diag}(S^TBS)\right)^{-1/2}$;
3:     Cholesky factorize $R^TR = DS^TBSD$.
4:     Solve symmetric eigenvalue problem $\left(R^{-T}DS^TASDR^{-1}\right)Z = Z\Theta$.
5:     $C = DR^{-1}Z$;
6: end function
```

When S^TBS is poorly conditioned or numerically singular, Cholesky factorization may fail. Even if factorization succeeds R may be poorly conditioned thus introducing significant roundoff error in the transformed problem (1.4) as well as the final transformation $C = R^{-1}Z$. This is often problematic since near linear dependence of columns in S naturally emerges when some columns of $X^{(i)}$ become accurate eigenvector approximations. Corresponding columns in both $W^{(i)}$ and $P^{(i)}$ become small in magnitude and thus exhibit large relative error. See section 1.7 for further analysis.

A proper implementation of the LOBPCG algorithm should deflate converged eigenvectors by keeping them in $X^{(i)}$ but exclude corresponding columns from $W^{(i)}$ and $P^{(i)}$. This technique is referred to as *soft locking*. In order to produce reliable results, converged eigenpairs should be counted consecutively beginning with the extreme eigenvalue. Some implementations allow out-of-order locking which can slightly improve performance, but care must be taken to avoid skipping eigenvalues when returning results. Another technique that helps overcome poor scaling is to normalize each column of $W^{(i)}$ and $P^{(i)}$ before performing the Rayleigh-Ritz procedure. This is equivalent to scaling S^TBS by a diagonal matrix D which should be rounded to integer powers of 2 to avoid truncation error. Although this form of scaling does not alter mantissa arithmetic in R, it can dramatically improve the condition number with respect to inversion, which is significant as R^{-1} is applied three times in this computation. [14]

Unfortunately, neither soft-locking nor simple scaling fundamentally solve the numerical instability that potentially leads to a breakdown. We observe that even with soft locking and diagonal scaling S^TBS can still become ill-conditioned. When the number of eigenpairs to be computed is relatively large, S^TBS can become ill-conditioned before any approximate eigenvectors in $X^{(i)}$ are sufficiently accurate. This failure is quite common and is observed in some of the cases we tested in Section 1.8. We also provide a detailed example of this phenomenon in Section 1.5.1.

1.4.1 Basis selection

Hetmaniuk and Lehoucq (HL) proposed a way to overcome the numerical difficulty associated with ill-conditioning in S^TBS [32]. Their basic approach is to keep X, W and P blocks in the subspace $\mathcal{S}$ mutually B-orthogonal. They refer to this as a basis selection strategy for $\mathcal{S}$.

Assuming the blocks $X^{(i)}$ and $P^{(i)}$ are B-orthonormal already, the basis selection scheme proposed by HL is performed in two steps on each iteration:

1. Before Rayleigh-Ritz has been performed, $W^{(i)}$ is obtained from residuals and then B-orthogonalized against both $X^{(i)}$ and $P^{(i)}$. Columns of $W^{(i)}$ are then B-orthonormalized.

2. After Rayleigh-Ritz has been performed, $P^{(i+1)}$ is implicitly B-orthogonalized against $X^{(i+1)}$. This is done by forming $C_p^{(i+1)}$ from $[0; C_2^{(i+1)}; C_3^{(i+1)}]$ which is orthogonalized against $C_x^{(i+1)}$ in the metric $S^{(i)T}BS^{(i)}$. The result is then orthonormalized in the same metric producing fully orthonormal blocks $X^{(i+1)} = S^{(i)}C_x^{(i+1)}$ and $P^{(i+1)} = S^{(i)}C_p^{(i+1)}$.

HL use the procedure `ortho` to carry out both of these orthogonalization steps. For implementation, they reference work by Stathopolous and Wu (SW) [65]. The procedure `ortho` operates in two nested loops. In the outer loop a candidate basis is orthogonalized against an existing orthonormal basis, called the external basis, using classical Gram–Schmidt. In the inner loop, the remainder is orthonormalized using the singular value decomposition. This is done by a function called `svqb`. These procedures are outlined in Algorithms 3 and 4. Both of these versions have been altered to incorporate a metric M which will be used as both $M = B$ and $M = S^TBS$. The original versions only considered $M = I$.

By constructing an orthonormal basis for S, HL are able to turn the generalized eigenvalue problem (1.3) into a standard eigenvalue problem in the Rayleigh–Ritz procedure. Thus Cholesky factorization of the projection S^TBS becomes unnecessary.

Algorithm 3 Block orthogonalization algorithm proposed by Stathopolous and Wu.

```
Input:
    $M$ is $m \times m$ symmetric positive definite metric.
    $U^{(\texttt{in})}$ is $m \times n_u$ candidate basis.
    $V$ is $M$-orthonormal external basis.
    $\tau > 0$ is relative orthogonality tolerance.
Output:
    $U^{(\texttt{out})}$ is $m \times n_u$ with $M$-orthonormal columns that are $M$-orthogonal to $V$.
    $\texttt{span}([U^{(\texttt{out})}, V]) \supseteq \texttt{span}([U^{(\texttt{in})}, V])$.
1: function $U^{(\texttt{out})}=\texttt{ortho}(M, U^{(\texttt{in})}, V, \tau)$
2:     do $i = 1, 2, \ldots$
3:         $U = U - V(V^T MU)$;
4:         do $j = 1, 2, \ldots$
5:             $U = \texttt{svqb}(M, U)$;
6:         while $\frac{\left\|U^T MU - I_{n_u}\right\|}{\|MU\|\|U\|} > \tau$
7:     while $\frac{\left\|V^T MU\right\|}{\|MV\|\|U\|} > \tau$
8: end function
```

Algorithm 4 Orthonormalization algorithm using SVD proposed by Stathopolous and Wu.

```
Input:
    $M$ is $m \times m$ symmetric positive definite metric.
    $U^{(\texttt{in})}$ is $m \times n_u$.
    $\tau > 0$ is tolerance.
Output:
    $U^{(\texttt{out})}$ is $m \times n_u$ with $M$-orthonormal columns.
    $\texttt{span}(U^{(\texttt{out})}) \supseteq \texttt{span}(U^{(\texttt{in})})$.
1: function $U^{(\texttt{out})}=\texttt{svqb}(M, U^{(\texttt{in})}, \tau)$
2:     $D = \left(\texttt{diag}\left(U^T MU\right)\right)^{-1/2}$;
3:     Solve $\left(DU^T MUD\right) Z = Z\Theta$ for $Z, \Theta$.
4:     for all $\theta_j < \tau \max_i(|\theta_i|)$ do
5:         $\theta_j = \tau \max_i(|\theta_i|)$;
6:     end for
7:     $U = UDZ\Theta^{-\frac{1}{2}}$;
8: end function
```

1.5 Stability improvements

1.5.1 Basis truncation

Although the HL basis selection algorithm is usually effective, its functionality hinges on the success of `ortho` in producing an orthonormal basis $S^{(i)}$.

When source columns for $S^{(i)}$ are ill-conditioned or linearly dependent, the orthogonalization procedure proposed by Stathapolous and Wu might not improve the basis. Depending on the implementation, `ortho` may fail to terminate because the orthogonality error threshold might never be satisfied. This is possible even when B is identity and becomes more vexing when B is ill-conditioned.

Even if `ortho` terminates after potentially numerous iterations, the returned basis might be so poorly conditioned that some eigenvalues of S^TAS are spurious. That is, they do not represent an approximation to any eigenvalue of A.

The following example illustrates how this problem could occur. Let

$$A = \begin{bmatrix} 3 & 1 & & & & \\ 1 & 3 & 1 & & & \\ & 1 & 3 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 3 & 1 \\ & & & & 1 & 3 \end{bmatrix}, \qquad B = K = I, \qquad X^{(0)} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Then $\Theta^{(0)} = \mathrm{diag}([2,\ 4])$ and $R^{(0)} = AX^{(0)} - X^{(0)}\Theta^{(0)} = [-e_3,\ e_3]/\sqrt{2}$. The span of $[X^{(0)},\ R^{(0)}]$ is equivalent to $\mathrm{span}([e_1,\ e_2,\ e_3])$. However, `svqb` returns an output S consisting of four linearly dependent vectors in both exact and floating-point arithmetic.

If we simply neglect the failure of `svqb` and assume orthonormality of S (i.e., $S^TS = I$) in the subsequent call to `RayleighRitz`, then we obtain $\theta_1 = 0$ as S is rank deficient. This is a spurious Ritz value since A is positive definite with smallest eigenvalue larger than 1. Note that this might appear to violate the Courant-Fischer min max theorem since the Ritz value is below the minimum eigenvalue of A. In fact, the corresponding Ritz vector reveals the linear dependency in the basis. That is, the primitive Ritz vector is the linear combination of basis vectors yielding zero which underscores the fact that it is spurious. We analyze the connection between orthogonality error and results of Rayleigh–Ritz in more detail in Section 1.7.

To overcome this difficulty, we modify `ortho` to truncate basis vectors below roundoff error threshold. We thank Professor Lin Lin for a useful discussion in which this idea was proposed [46]. The eigenvalue decomposition in `svqb` gives the form

$$U^T M U = \begin{bmatrix} z_1 & z_2 & \cdots & z_{n_u} \end{bmatrix} \begin{bmatrix} \theta_1 & & & \\ & \theta_2 & & \\ & & \ddots & \\ & & & \theta_{n_u} \end{bmatrix} \begin{bmatrix} z_1^T \\ z_2^T \\ \vdots \\ z_{n_u}^T \end{bmatrix}$$

where we have sorted eigenvalues $\theta_1 \geq \theta_2 \geq \cdots \geq \theta_{n_u}$. Let $k \leq n_u$ be the number of leading eigenpairs that are above the drop threshold: $\theta_k \geq \tau_{\texttt{drop}} \theta_1$ where $\tau_{\texttt{drop}}$ is a small multiple of machine epsilon. The retained basis becomes

$$U^{(\texttt{out})} = U \begin{bmatrix} z_1 & \cdots & z_k \end{bmatrix} \begin{bmatrix} \theta_1^{-1/2} & & \\ & \ddots & \\ & & \theta_k^{-1/2} \end{bmatrix}.$$

The span of the new basis is a good approximation of the span of the source basis.

$$\forall c_1 \in \mathbb{R}^{n_u} \quad \exists c_2 \in \mathbb{R}^k \quad \text{such that} \quad \frac{\|U c_1 - U^{(\texttt{out})} c_2\|_2}{\|U\|_2 \|c_1\|_2} \leq \tau_{\texttt{drop}}.$$

If maintaining a guaranteed minimum basis dimension is necessary, the basis can be padded with randomly generated B-orthogonalized columns, however randomized padding has never been necessary or useful in any of our experiments.

These modifications are outlined in Section 1.6, Algorithms 6 and 7. The implementation we tested still allows one call to `svqb` without dropping in order to extract potentially useful information, however subsequent iterations of the inner loop switch to the modified version to ensure a successful exit.

1.5.2 Detecting convergence

In some of our numerical experiments with the HL implementation of LOBPCG we observed both unexpectedly high and low iterations required to obtain convergence. One such example, Andrews, is a standard eigenvalue problem from the University of Florida Sparse Matrix Collection (UFSMC). Andrews is a $60,000 \times 60,000$ symmetric matrix with $760,154$ nonzero elements. We attempted to find the minimum 400 eigenpairs using a block size of 440 columns with a convergence tolerance of 10^{-4}. This test run was forced to exit without having converged after 1000 iterations.

Another test problem showed the opposite difficulty with convergence. The matrix pencil filter2D, which is also available from UFSMC, has symmetric sparse matrices A and B of dimension $1,668$. A has $10,750$ nonzero elements and B is diagonal. In this test case we also sought the lowest 400 eigenvalues using 440 columns per block and the same convergence tolerance. The algorithm reported convergence immediately after the first iteration. We emphasize that the first iteration is merely `RayleighRitz` on a random matrix.

These difficulties result from the relative residual computation that is typically used to detect convergence. Residual norms are measured relative to the magnitude of the eigenvalue which requires

$$\frac{\|r_i\|_2}{|\theta_i|\|x_i\|_2} \le \tau \quad \text{or} \quad \frac{\|r_i\|_2}{|\theta_i|\|x_i\|_B} \le \tau, \quad \text{where} \quad r_i = Ax_i - \theta_i B x_i, \tag{1.5}$$

to consider an eigenvalue–eigenvector pair converged. This criterion has two problems. The first problem is both versions lack scaling invariance. Scaling B does not change residuals or eigenvectors, but eigenvalues scale

$$AX = \frac{B}{\beta} X(\beta\Theta) \quad \text{giving} \quad \frac{\|r_i\|_2}{|\beta\theta_i|\|x_i\|_2} \le \tau \quad \text{or} \quad \frac{\|r_i\|_2}{|\beta\theta_i|\|x_i\|_{B/\beta}} = \frac{\|r_i\|_2}{\sqrt{\beta}|\theta_i|\|x_i\|_B} \le \tau$$

which both depend on the factor β. As a result, convergence detection using such a measure is somewhat arbitrary. Of course, this could be repaired by forcing a uniform scaling of B or including $\|B\|$ in the denominator.

The second problem with relative residual convergence measures persists even if B is identity. Section 1.7.2 shows that it may not be possible to compute a residual in floating point arithmetic that satisfies a criterion of the form (1.5). In the Andrews example, the smallest eigenvalue is less than 10^{-14} in magnitude and $\|A\| \approx 10$. Ill-conditioning of A does not permit the smallest eigenvalue to be known to 4 digits of precision. In the filter2D example, $\|B\| \approx 10^{-10}$ which makes convergence too easy to achieve.

We employ an alternative convergence criterion that we show to be backwards stable in Section 1.7.4.

$$\frac{\|r_i\|_2}{\left(\|A\|_2 + |\theta_i|\|B\|_2\right)\|x_i\|_2} \le \tau. \tag{1.6}$$

The 2-norms on A and B can be bound with little computational cost by using a Gaussian random $k \times m$ projection matrix Ω. The Frobenius norm is bound

$$\|\Omega A\|_F^2 \le \|\Omega\|_F^2 \|A\|_2^2 \quad \text{which allows us to define} \quad \|A\|_2^{(\Omega)} = \frac{\|\Omega A\|_F}{\|\Omega\|_F} \le \|A\|_2.$$

This guarantees our convergence criterion is satisfied if

$$\frac{\|r_i\|_2}{\left(\|A\|_2 + |\theta_i|\|B\|_2\right)\|x_i\|_2} \le \frac{\|r_i\|_2}{\left(\|A\|_2^{(\Omega)} + |\theta_i|\|B\|_2^{(\Omega)}\right)\|x_i\|_2} \le \tau.$$

Using this convergence test, Andrews converges after performing 56 iterations. Likewise, filter2D converges after performing 10 iterations.

1.6 Efficiency improvements

The following efficiency improvements can be safely included in LOBPCG without sacrificing algorithmic stability. Our version and related subroutines are outlined in Algorithms 5, 6, 7, and 8 at the end of this section.

1.6.1 Implicit product updates

When there is sufficient memory to store S, AS, and BS if $B \neq I$, HL suggest a possible improvement to efficiency by employing implicit product updates. Given block updates $X^{(i+1)} = S^{(i)}C_x^{(i+1)}$ and $P^{(i+1)} = S^{(i)}C_p^{(i+1)}$, matrix products can be implicitly updated using the same transformations.

$$\begin{aligned} AX^{(i+1)} &= AS^{(i)}C_x^{(i+1)} & AP^{(i+1)} &= AS^{(i)}C_p^{(i+1)} \\ BX^{(i+1)} &= BS^{(i)}C_x^{(i+1)} & BP^{(i+1)} &= BS^{(i)}C_p^{(i+1)} \end{aligned}$$

This is beneficial when direct matrix multiplication is expensive. Direct multiplication requires the number of nonzero elements in the matrix $\texttt{nnz}(A)$ operations per column while implicit updating requires $3mn_x$ per column. More importantly, dense matrix multiplication is highly tuned to optimize communication and parallel scalability. However, such updates accumulate roundoff error which can hinder stability and convergence. This is discussed in Section 1.7.1.

We can extend this tool to reduce direct computation of block inner products in the projection matrices

$$S^{(i)T}AS^{(i)} = \begin{bmatrix} X^{(i)T}AX^{(i)} & X^{(i)T}AW^{(i)} & X^{(i)T}AP^{(i)} \\ \cdots & W^{(i)T}AW^{(i)} & W^{(i)T}AP^{(i)} \\ \cdots & \cdots & P^{(i)T}AP^{(i)} \end{bmatrix}$$

and

$$S^{(i)T}BS^{(i)} = \begin{bmatrix} X^{(i)T}BX^{(i)} & X^{(i)T}BW^{(i)} & X^{(i)T}BP^{(i)} \\ \cdots & W^{(i)T}BW^{(i)} & W^{(i)T}BP^{(i)} \\ \cdots & \cdots & P^{(i)T}BP^{(i)} \end{bmatrix}.$$

Most implementations we have seen take advantage of some known structure within each block.

$$X^{(i)T}AX^{(i)} = \Theta^{(i)} \quad \text{and} \quad X^{(i)T}BX^{(i)} = I.$$

Because $P^{(i)}$ is in the orthogonal complement of the previous solution we also have

$$X^{(i)T}AP^{(i)} = 0, \quad X^{(i)T}BP^{(i)} = 0 \quad \text{and} \quad P^{(i)}BP^{(i)} = I.$$

Implicit updating can be used to avoid an additional block inner product

$$P^{(i)T}AP^{(i)} = C_p^T \left(S^{(i-1)T}AS^{(i-1)}\right) C_p.$$

This computation can be improved further by applying the technique described in Section 1.6.6.

Every W block must be directly computed unless $B = I$ and the preconditioner is $K = I$. In that case, $W^{(i)}$ is the block of residuals which must be orthogonal to the search subspace from which previous Rayleigh-Ritz solutions were formed including both $X^{(i)}$ and $P^{(i)}$ yielding $X^{(i)T}W^{(i)} = 0$ and $W^{(i)T}P^{(i)} = 0$.

Since every block inner product must move $\mathcal{O}(2mn_x)$ data through processors, every direct computation avoided significantly improves performance.

1.6.2 Implicit updates and `svqb`

As we have discussed, generalized eigenvalue problems or matrix pencils require orthogonalization in a nontrivial metric B performed by `ortho` and `svqb`. It is tempting to incorporate implicit product updates in these functions as well. This might take on the form

$$\begin{aligned} U^{(1)} &= U^{(0)} - V\left([BV]^T U^{(0)}\right), & U^{(2)} &= U^{(1)}\left(DZ\Theta^{-1/2}\right), \\ [BU]^{(1)} &= [BU]^{(0)} - [BV]\left([BV]^T U^{(0)}\right), \quad \text{and} & [BU]^{(2)} &= [BU]^{(0)}\left(DZ\Theta^{-1/2}\right). \end{aligned}$$

This is in general a bad idea. Numerical experiments using such updates show that when B is not identity, however well-conditioned, the inner loop of `ortho` often fails to terminate if the candidate basis $U^{(0)}$ has a condition number in the metric B of 10^6 or higher. Nearly all randomized tests will fail under this circumstance. Indeed, the original purpose for using `ortho` was to handle such ill-conditioned bases that cause LOBPCG to fail otherwise. The reason why these implicit updates are unstable while others are safe is explored in Section 1.7.1, but essentially cancellation in extraction dramatically magnifies accumulated error in this context.

1.6.3 Product purification

Even though other implicit product updates are relatively safe, error accumulation might still become problematic over many iterations. This is typically only a concern for high-precision computations of eigenvalues that require numerous iterations to converge. As such, we did not include the following test in our final algorithm, however this approach may be of interest for some high-precision applications.

Randomized projection can be used to efficiently approximate error in product representations [1, 13]. This follows from the well-known Johnson–Lindenstrauss lemma [38]. To determine when product arrays need to be recomputed we simply apply a Gaussian $N(0,1)$ matrix $\Omega \in \mathbb{R}^{k\times m}$ to obtain the norm approximation

$$\|AX^{(i)} - [AX]^{(i)}\|_F \approx \frac{\|[\Omega A]X^{(i)} - \Omega[AX]^{(i)}\|_F}{\sqrt{k}}.$$

This computation is reliable and we provide an elementary analysis here. Let $E = AX^{(i)} - [AX]^{(i)}$ be the matrix of errors in the product representation. Since the Frobenius norm is invariant under orthogonal transformations, as is the distribution of Ω, we can easily simplify the error approximation by using the singular values $E = U\Sigma V^T$.

$$\|\Omega E\|_F^2 = \|\Omega U\Sigma V^T\|_F^2 = \|\hat{\Omega}\Sigma\|_F^2 = \sum_{i=1}^{k}\sum_{j=1}^{n} \hat{\omega}_{i,j}^2 \sigma_j^2.$$

The expected Frobenius norm is scaled to match the true norm

$$\mathbb{E}\left(\left\|\frac{\Omega E}{\sqrt{k}}\right\|_F^2\right) = \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{n} \mathbb{E}\left(\hat{\omega}_{i,j}^2\right)\sigma_j^2 = \sum_{j=1}^{n}\sigma_j^2 = \|E\|_F^2$$

and the corresponding variance is bound above using the Cauchy-Schwarz inequality

$$\mathbb{V}\left(\left\|\frac{\Omega E}{\sqrt{k}}\right\|_F^2\right) = \frac{2}{k}\sum_{j=1}^{n}\sigma_j^4 \le \frac{2}{k}\|E\|_F^4.$$

We only need the magnitude of $\|E\|_F$ to decide if an update of the product array is necessary. To obtain an approximation $\|\frac{\Omega E}{\sqrt{k}}\|_F \ge \phi\|E\|_F$, where $\phi \in (0,1)$ is a safe fraction of the true magnitude, requires $\|E\|_F^2 - \|\frac{\Omega E}{\sqrt{k}}\|_F^2 \le (1-\phi^2)\|E\|_F^2$. At n_σ-sigma confidence this is bound by

$$n_\sigma\sqrt{\mathbb{V}\left(\left\|\frac{\Omega E}{\sqrt{k}}\right\|_F^2\right)} \le n_\sigma\sqrt{\frac{2}{k}}\,\|E\|_F^2 \le (1-\phi^2)\|E\|_F^2 \quad \text{which requires} \quad k \ge \frac{2n_\sigma^2}{(1-\phi^2)^2}.$$

We can achieve 5-sigma confidence that the approximation holds to a fraction $\phi = 1/8$ using $k = 52$. A sparse compression matrix would likely improve performance further [60].

1.6.4 When to use basis orthogonalization

Although constructing an orthonormal basis for S guarantees `RayleighRitz` will not fail, the construction process itself can be costly and sometimes unnecessary. The principal extra cost is contained in the call to `ortho` used to complete block W. Even if we assume each step within Algorithm 3 succeeds on the first iteration, the corresponding basis update computations would be

$$W^{(1)} = W^{(0)} - [X,\ P]([X,\ P]^T BW^{(0)}) \quad \text{and} \quad W^{(2)} = W^{(1)}(DZ\Theta^{1/2}).$$

As a result, the memory block containing W must be accessed at least twice.

If `ortho` is skipped, which forces `RayleighRitz` to construct and apply Cholesky factors of $S^T BS$, the transformations that would have been performed in `ortho` are subsumed by updates to X and P. As a result, iterations that skip `ortho` reduce memory movement by more than two full passes over W. Furthermore, `svqb` requires solving an $n_x \times n_x$ eigenvalue problem which is more time consuming than the corresponding Cholesky decomposition.

In order to take advantage of this possible performance improvement without sacrificing stability, we need to determine when the Cholesky decomposition $R^T R = S^T BS$ becomes unreliable. As R^{-1} is applied three times, we simply require the condition number to be bound $\mu_\epsilon \texttt{cond}(R)^3 < \tau$.

Knowing this allows us to skip orthogonalization of W initially. When the condition number exceeds the a safe threshold we switch to iterations that apply full orthogonalization. Note that our method to construct orthogonal $P^{(i)}$ blocks is not expensive and therefore not worth skipping. After $\texttt{cond}(R)$ passes the safe threshold, subsequent iterations tend to remain above threshold. As a result, we never attempt to switch back to iterations that skip `ortho`.

1.6.5 Early `ortho` exiting

Just as conditioning of $S^T BS$ allows us to identify when `ortho` may be safely skipped, we can also use the condition number of $W^T BW$ within `svqb` to predict acceptable orthogonality error.

The update $W^{(2)} = W^{(1)}(DZ\Theta^{-1/2})$ produces relative orthogonality error of magnitude $\mu_\epsilon \texttt{cond}(\Theta^{-1/2})$, therefore we can avoid computing the resulting orthogonality error in $W^{(2)T} BW^{(2)}$ when $\texttt{cond}(\Theta^{-1/2})$ is small. Likewise, relative orthogonality error testing in the outer loop of `ortho` may be skipped if the `svqb` transformation was well-conditioned on the *first* iteration of the inner loop. This improvement usually allows us to avoid three block inner products for each call to `ortho`.

1.6.6 Alternative orthogonal basis updates

We propose an alternative construction of the aggregate solution component matrix C_p that enables both X and P to be constructed simultaneously which leads to a further performance improvement discussed in Section 1.6.7. `RayleighRitz` always computes orthonormal primitive eigenvectors Z satisfying

$$S^TBS = R^TR, \quad \left(R^{-T}S^TASR^{-1}\right)Z = Z\Theta, \quad \text{and} \quad Z = RC$$

partitioned as

$$Z = \begin{bmatrix} Z_1 & Z_{1\perp} \\ Z_2 & Z_{2\perp} \\ Z_3 & Z_{3\perp} \end{bmatrix}, \quad R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix},$$

$$C = \begin{bmatrix} C_1 & C_{1\perp} \\ C_2 & C_{2\perp} \\ C_3 & C_{3\perp} \end{bmatrix}, \quad \text{and} \quad \Theta = \begin{bmatrix} \Theta_x & 0 \\ 0 & \Theta_\perp \end{bmatrix}.$$

The orthogonal complement to X within the subspace spanned by S corresponds to primitive components

$$Z_\perp = \begin{bmatrix} Z_{1\perp} \\ Z_{2\perp} \\ Z_{3\perp} \end{bmatrix} \quad \text{which give} \quad C_\perp = R^{-1}Z_\perp = \begin{bmatrix} C_{1\perp} \\ C_{2\perp} \\ C_{3\perp} \end{bmatrix}.$$

$Z_\perp$ can be transformed into orthonormal Z_p to combine basis updates

$$\begin{bmatrix} X & P \end{bmatrix} = S\begin{bmatrix} C_x & C_p \end{bmatrix} = S\left(R^{-1}\begin{bmatrix} Z_x & Z_p \end{bmatrix}\right).$$

The unorthogonalized aggregate update components used by HL have corresponding primitives

$$\hat{C}_p = \begin{bmatrix} 0 \\ C_2 \\ C_3 \end{bmatrix} \quad \text{and} \quad \hat{Z}_p = \begin{bmatrix} \hat{Z}_1 \\ Z_2 \\ Z_3 \end{bmatrix}$$

$$\text{where} \quad \hat{Z}_1 = \begin{bmatrix} R_{12} & R_{13} \end{bmatrix}\begin{bmatrix} R_{22} & R_{23} \\ 0 & R_{33} \end{bmatrix}^{-1}\begin{bmatrix} Z_2 \\ Z_3 \end{bmatrix}.$$

$\hat{Z}_p$ can easily be orthogonalized against Z_x by projecting it onto $Z_\perp$, which is already available. QR factorization of projection components produces the orthogonal transformation that yields orthonormal Z_p.

$$Q_pR_p = Z_\perp^T\hat{Z}_p \quad \text{and} \quad Z_p = Z_\perp Q_p.$$

This improves communication efficiency by combining BLAS-3 operations used to form X and P. We also avoid the second call to `ortho` needed by HL. Also note how this simplifies the update to the P inner product $P^TAP = Q_p^T\Theta_\perp Q_p$.

1.6.7 Communication efficient matrix multiply-and-update

Previous versions of LOBPCG require us to store $X^{(i+1)}$ and $S^{(i)}$ in separate arrays. $S^{(i)}$ is needed to construct $P^{(i+1)}$, but $X^{(i+1)}$ is also needed to compute residuals which determine columns of $P^{(i+1)}$ to be formed. This forces memory stored in S to be accessed at least three times per update.

$$X^{(i+1)} = S^{(i)}C_x, \quad P^{(i+1)} = S^{(i)}C_p, \quad \text{and} \quad S^{(i+1)}(:, 1 : n_x + n_p) = \left[\begin{array}{cc} X^{(i+1)} & P^{(i+1)} \end{array}\right].$$

We reduce this expense to a single pass over S. This is done by delaying soft locking by one iteration. If columns of P are computed for eigenpairs that had not converged on the previous iteration then both $X^{(i+1)}$ and $P^{(i+1)}$ can be computed before residual norms are known, which eliminates the need to store results in separate arrays. This allows results to be stored in S during the same memory pass used to compute the product. We do this by partitioning S into row panels which are multiplied into local temporary arrays that are copied back before moving on to the next row panel. Partitioning

$$S^{(i)} = \begin{bmatrix} S_1^{(i)} \\ S_2^{(i)} \\ \vdots \end{bmatrix}, \quad \text{take} \quad T = S_j^{(i)} \left[\begin{array}{cc} C_x^{(i+1)} & C_p^{(i+1)} \end{array}\right] \quad \text{then} \quad S_j^{(i+1)}(:, 1 : n_x + n_p) = T.$$

T is a temporary array owned by the processor handling the given panel. Panel thickness should be chosen to allow T, $S_j^{(i)}$, and $[C_x\ C_p]$ to reside in cache simultaneously. Note that we have moved P to be adjacent to X in the array S which has a secondary benefit; if trailing columns of W are subsequently dropped by `ortho`, P does not need to be moved.

Algorithm 5 Our LOBPCG algorithm

Input:

$X^{(0)}$ is $m \times n_x$ initial approximate eigenvectors.

$n_v \leq n_x$ is the number of converged eigenvectors requested.

τ is the threshold used to determine eigenpair convergence.

Output:

X is $m \times n_v$ matrix of approximate eigenvectors.

Λ is $n_v \times n_v$ diagonal matrix of approximate eigenvalues.

```
1: function $[X, \Lambda]$=lobpcgDGSY($X^{(0)}, n_v, \tau$)
2:     $[C^{(1)}, \Theta^{(1)}]$ = RayleighRitz($X^{(0)}$)
3:     $X^{(1)} = X^{(0)} C^{(1)}$;
4:     $R^{(1)} = AX^{(1)} - BX^{(1)}\Theta^{(1)}$;
5:     $n_c = 0$; useOrtho = false; $P^{(1)} = []$;
6:     do $i = 1, 2, \ldots$
7:         if useOrtho $W^{(i)}$ = orthoDrop($B, R^{(i)}, [X^{(i)}\ P^{(i)}]$);
8:         else $W^{(i)} = R^{(i)}$;
9:         $S^{(i)} = \left[X^{(i)}\ P^{(i)}\ W^{(i)}\right]$;
10:        $[C^{(i+1)}, \Theta^{(i+1)}]$ = RayleighRitzModified($S^{(i)}, n_x, n_c$);
11:        $[X^{(i+1)}\ P^{(i+1)}] = S^{(i)}[C_x\ C_p]$;
12:        $R^{(i+1)} = AX^{(i+1)} - BX^{(i+1)}\Theta^{(i+1)}$;
13:        Determine number of converged eigenpairs $n_c$.
14:    while $n_c < n_v$
15:    Return converged eigenpairs in $X$ and $\Lambda$;
16: end function
```

Algorithm 6 Modified orthogonalization procedure.

```
Input:
    $M$ is $m \times m$ symmetric positive definite metric.
    $U^{(\texttt{in})}$ is $m \times n_u^{(\texttt{in})}$ candidate basis.
    $V$ is $M$-orthonormal external basis.
    $\tau > 0$ is relative orthogonality tolerance.
Output:
    $U^{(\texttt{out})}$ is $m \times n_u^{(\texttt{out})}$. $n_u^{(\texttt{out})} \le n_u^{(\texttt{in})}$.
    $\texttt{span}([U^{(\texttt{out})}, V]) \approx \texttt{span}([U^{(\texttt{in})}, V])$.
 1: function $U^{(\texttt{out})} = \texttt{orthoDrop}(M, U^{(\texttt{in})}, V)$
 2:     do $i = 1, 2, 3$
 3:         $U = U - V(V^T MU)$;
 4:         do $j = 1, 2, 3$
 5:             if $j == 1$ then
 6:                 $U = \texttt{svqb}(M, U, \tau_{\texttt{replace}})$;
 7:             else
 8:                 $U = \texttt{svqbDrop}(M, U, \tau_{\texttt{drop}})$;
 9:             end if
10:         while $\frac{\|U^T MU - I_{n_u}\|}{\|MU\|\|U\|} > \tau_{\texttt{relOrthoErr}}$
11:     while $\frac{\|V^T MU\|}{\|MV\|\|U\|} > \tau_{\texttt{relOrthoErr}}$
12: end function
```

Algorithm 7 SVQB with dropping

```
Input:
    $M$ is $m \times m$ symmetric positive definite metric.
    $U^{(\texttt{in})}$ is $m \times n_u$.
    $\tau > 0$ is tolerance.
Output:
    $U^{(\texttt{out})}$ is $m \times n_u$. $n_u^{(\texttt{out})} \le n_u^{(\texttt{in})}$.
    $\texttt{span}(U^{(\texttt{out})}) \approx \texttt{span}(U^{(\texttt{in})})$.
 1: function $U^{(\texttt{out})} = \texttt{svqbDrop}(M, U^{(\texttt{in})}, \tau)$
 2:     $D = \left(\texttt{diag}\left(U^T MU\right)\right)^{-1/2}$;
 3:     Solve $\left(DU^T MUD\right) Z = Z\Theta$ for $Z, \Theta$.
 4:     Determine columns to keep $J = \{j : \theta_j > \tau \max_i(|\theta_i|)\}$.
 5:     $U = UDZ(:, J)\Theta(J, J)^{-1/2}$;
 6: end function
```

Algorithm 8 Rayleigh–Ritz with alternative basis update

```
Input:
    $S$ is an $m \times n_s$ matrix forming a basis for the search subspace.
    $n_x$ is the number of extreme eigenpair approximations to return.
    $n_c$ is the number of converged eigenpairs from the previous iteration.
Output:
    $C$ is $n_s \times (2n_x - n_c)$. First $n_x$ columns are $C_x$ followed by $n_x - n_c$ giving $C_p$.
    Output satisfies $C^T(S^TBS)C = I$ and $C^T(S^TAS)C = \Theta$.
 1: function $[C,\Theta]$=RayleighRitzModified($S$,$n_x$,$n_c$)
 2:     if useOrtho then
 3:         Solve $\left(S^TAS\right) Z = Z\Theta$.
 4:         QR factorize
                $$Q_pR_p = \begin{bmatrix} Z_{2\perp}^T & Z_{3\perp}^T \end{bmatrix} \begin{bmatrix} Z_2(:, n_c+1:n_x) \\ Z_3(:, n_c+1:n_x) \end{bmatrix}.$$
 5:         $[C_x\ C_p] = [Z_x\ Z_\perp Q_p(:, 1:n_x - n_c)]$;
 6:     else
 7:         $D = \left(\mathtt{diag}(S^TBS)\right)^{-1/2}$                ▷ Round to integer powers of 2.
 8:         Cholesky factorize $R^TR = DS^TBSD$.
 9:         if $(\mathtt{cond}(R) > \tau_R)$ useOrtho = true; Exit and restart this iteration.
10:         Solve $\left(R^{-T}DS^TASDR^{-1}\right) Z = Z\Theta$.
11:         QR factorize
                $$Q_pR_p = \left( Z_{1\perp}^T \begin{bmatrix} R_{12} & R_{13} \end{bmatrix} \begin{bmatrix} R_{22} & R_{23} \\ 0 & R_{33} \end{bmatrix}^{-1} + \begin{bmatrix} Z_{2\perp}^T & Z_{3\perp}^T \end{bmatrix} \right) \begin{bmatrix} Z_2(:, n_c+1:n_x) \\ Z_3(:, n_c+1:n_x) \end{bmatrix}.$$
12:         $[C_x\ C_p] = DR^{-1}[Z_x\ Z_\perp Q_p(:, 1:n_x - n_c)]$;
13:     end if
14:     Update $\Theta$ to represent partial inner products
                $$\Theta = \begin{bmatrix} \Theta_x & 0 \\ 0 & Q_p(:, 1:n_x - n_c)^T \Theta_\perp Q_p(:, 1:n_x - n_c) \end{bmatrix}.$$
15: end function
```

1.7 Analysis

1.7.1 Implicit update error

Suppose we have arrays representing a basis X and product $\mathtt{fl}(AX) = AX + \epsilon_1\|AX\|E_1$ where we chose $\epsilon_1 > 0$ to normalize the representation error matrix $\|E_1\| = 1$. All such error matrices below are normalize the same way $\|E_i\| = 1$ by definition of $\epsilon_i > 0$. We wish to apply the transformation T. Floating point arithmetic gives

$$\hat{X} = \mathtt{fl}(XT) = XT + \epsilon_2\|X\|\|T\|E_2,$$

however we can define the new basis to be exactly represented by the resulting array $\hat{X}$. Computing the product directly in floating point arithmetic leaves representation error

$$\mathtt{fl}(A\hat{X}) = A\hat{X} + \epsilon_3\|A\|\|\hat{X}\|E_3.$$

Alternatively, we could apply the transformation to the stored product $\mathtt{fl}(AX)$ to obtain

$$\mathtt{fl}(\mathtt{fl}(AX)T) = AXT + \epsilon_1\|AX\|E_1T + \epsilon_4\|AX\|\|T\|E_4.$$

We can express this error in the new basis by substituting $AXT = A\hat{X} - \epsilon_2\|X\|\|T\|AE_2$ to write the error as

$$\mathtt{fl}(\mathtt{fl}(AX)T) = A\hat{X} + \epsilon_5\|A\|\|\hat{X}\|E_5$$

where we have defined the composite error

$$\epsilon_5\|A\|\|\hat{X}\|E_5 = \epsilon_1\|AX\|E_1T - \epsilon_2\|X\|\|T\|AE_2 + \epsilon_4\|AX\|\|T\|E_4.$$

We can provide an upper bound for any sub-multiplicative matrix norm using the triangle inequality with $\|E_1T\| \leq \|T\|$, $\|AE_2\| \leq \|A\|$, and $\|AX\| \leq \|A\|\|X\|$ giving

$$\epsilon_5 \leq (\epsilon_1 + \epsilon_2 + \epsilon_4)\frac{\|X\|\|T\|}{\|\hat{X}\|}.$$

The implicit basis updates in our algorithm use orthogonal transformations for T. Thus we have a very slight increase in representation error of the product, however using implicit updates in `svqb` would be disastrous. If $X = U\Sigma V^T$ and we take the transformation $T = V\Sigma^{-1}$, then the representation inconsistency bound in the 2-norm is substantially magnified compared to the direct product

$$\epsilon_5 \leq (\epsilon_1 + 2\mu_\epsilon)\sigma_1\sigma_{n_x}^{-1} = (\epsilon_1 + 2\mu_\epsilon)\mathtt{cond}(X) \quad \text{versus} \quad \epsilon_3 \leq \mu_\epsilon.$$

1.7.2 Residual construction error

When we use our modified convergence test, the convergence tolerance must be set $\tau > 2\mu_\epsilon$. This is due to representation constraints. Let x and θ be an exact eigenpair so that the residual vanishes, $r = Ax - \theta Bx = 0$. Take the nearest floating-point representation to be $\hat{x} = x + \epsilon_x \|x\| e_1$ where we choose the scaling factor $\epsilon_x > 0$ to normalize the corresponding vector of roundoff errors $\|e_x\| = 1$. When the residual is computed in floating-point arithmetic, the matrix products are

$$\texttt{fl}(A\hat{x}) = A\hat{x} + \epsilon_x \|\hat{x}\| A e_x + \epsilon_A \|A\| \|\hat{x}\| e_A \quad \text{and}$$

$$\texttt{fl}(\theta B\hat{x}) = \theta B\hat{x} + \epsilon_x \|\hat{x}\| \theta B e_x + \epsilon_B |\theta| \|B\| \|\hat{x}\| e_B.$$

Again, scaling factors ϵ_A and ϵ_B are chosen so that $\|e_A\| = 1$ and $\|e_B\| = 1$. Since both product vectors are nearly equal, subtraction is exact

$$\texttt{fl}(r) = \texttt{fl}(A\hat{x}) - \texttt{fl}(B\hat{x}\theta) = \epsilon_x \|\hat{x}\| (A - \theta B) e_x + \epsilon_A \|A\| \|\hat{x}\| e_A - \epsilon_B |\theta| \|B\| \|x\| e_B.$$

This produces an upper bound on the computed residual 2-norm

$$\|\texttt{fl}(r)\|_2 \le 2\mu_\epsilon \left(\|A\|_2 + |\theta| \|B\|_2\right) \|\hat{x}\|_2 \quad \text{and} \quad \frac{\|\texttt{fl}(r)\|_2}{\left(\|A\|_2 + |\theta| \|B\|_2\right) \|\hat{x}\|_2} \le 2\mu_\epsilon.$$

Therefore we see that even the best possible representation $\hat{x}$ can only be verified to pass our convergence test if the tolerance is set above $2\mu_\epsilon$. This computation also shows that the relative residual corresponding to $\hat{x}$ would be bound

$$\frac{\|\texttt{fl}(r)\|_2}{|\theta| \|\hat{x}\|_2} \le 2\mu_\epsilon \left(\frac{\|A\|_2}{|\theta|} + \|B\|_2 \right).$$

While it is true that floating point truncation error could be smaller that the bounds above or even zero in contrived circumstances, truncation error cannot be guaranteed to produce smaller results. Therefore convergence detection thresholds must be set above these bounds. Even if $B = I$, the ratio $\frac{\|A\|_2}{|\theta|}$ could be quite large which is the reason why the relative residual is sometimes ill-suited for detecting convergence.

1.7.3 Rayleigh–Ritz orthogonality error

Suppose we have a basis stored in memory $\texttt{fl}(S)$ that we wish to use in `RayleighRitz`. Let us suppose that the Cholesky decomposition exists in exact arithmetic. That is, assume $\texttt{fl}(S)$ is linearly independent in the metric B.

$$\texttt{fl}(S)^T B \texttt{fl}(S) = R^T R \quad \text{and define} \quad S^* = \texttt{fl}(S) R^{-1}.$$

This would give the exact result in `RayleighRitz` since it is now a standard eigenvalue problem.

$$S^{*T}AS^*C^* = S^{*T}BS^*C^*\Theta^* = C^*\Theta^*.$$

We apply the Relative Weyl Theorem restated here: Let X be a nonsingular square matrix. Let $\lambda_i(A)$ and $\lambda_i(X^TAX)$ return the ith eigenvalues of A and X^TAX respectively, having been sorted in ascending order. It follows

$$\frac{\left|\lambda_i(X^TAX) - \lambda_i(A)\right|}{|\lambda_i(A)|} \leq \|X^TX - I\|_2$$

$$\frac{\left|\lambda_i\left(\mathtt{fl}(S)^TA\mathtt{fl}(S)\right) - \lambda_i\left(S^{*T}AS^*\right)\right|}{|\lambda_i\left(S^{*T}AS^*\right)|} \leq \|R^TR - I\|_2$$

$$\frac{|\theta_i - \theta_i^*|}{|\theta_i^*|} \leq \|\mathtt{fl}(S)^TB\mathtt{fl}(S) - I\|_2.$$

If we substitute the approximated Rayleigh quotient into our convergence criterion with $\theta_i = \theta_i^*(1+\epsilon)$ where relative error ϵ is bound as above then error in the convergence criterion computation would be

$$\epsilon_r \leq \frac{\|Bx_i\|\,|\theta_i^*\epsilon|}{(\|A\| + |\theta_i^*|\,\|B\|)\,\|x_i\|} \leq |\epsilon| \leq \|\mathtt{fl}(S)^TB\mathtt{fl}(S) - I\|_2.$$

This shows that for the convergence criterion to be possible to satisfy we must ensure B-orthogonality in $\mathtt{fl}(S)$ below the convergence tolerance τ.

1.7.4 Backwards stability

Let the eigenpair (x, θ) satisfy our convergence condition. Then perturbation matrices Δ_A and Δ_B exist for which the eigenpair is exact

$$(A + \Delta_A)\,x = \theta\,(B + \Delta_B)\,x \quad \text{or} \quad r = Ax - \theta Bx = (\theta\Delta_B - \Delta_A)\,x.$$

Minimal 2-norm perturbations can be constructed as rank-1 matrices

$$\Delta_A^* = \frac{-\alpha r x^T}{\|x\|_2^2} \quad \text{and} \quad \Delta_B^* = \frac{(1-\alpha)rx^T}{\theta\|x\|_2^2} \quad \text{satisfying} \quad \min_{\Delta_A,\Delta_B} \|\Delta_A\|_2 + |\theta|\|\Delta_B\|_2 = \frac{\|r\|_2}{\|x\|_2}$$

for $\alpha \in [0, 1]$. Substituting this into the convergence condition gives an upper bound on the relative perturbation magnitude

$$\frac{\|\Delta_A^*\|_2 + |\theta|\|\Delta_B^*\|_2}{\|A\|_2 + |\theta|\|B\|_2} \leq \tau.$$

1.8 Numerical examples

1.8.1 Convergence comparisons

The number of iterations performed by an implementation of LOBPCG will depend on the method used to detect convergence and the corresponding threshold. This introduces a difficulty when we attempt to compare our best algorithms against other algorithms that are available. Because we use the modified convergence criterion Equation 1.6, it is possible that our algorithm benefits from performing fewer iterations, which obfuscates the meaning of direct time comparisons.

In order to construct meaningful tests we have altered the source code of both Blopex and Anasazi implementations of LOBPCG to use our convergence condition. This ensures that when we compare performance, we are doing so for the same effective workload. However, in order to show that our convergence criterion is a fair improvement, we first compare each implementation using the original convergence (–OC) method and our modification. We also test our own Conservative implementation of the Hetmaniuk-Lehoucq algorithm (CHL). This version does not perform any implicit updates or employ other performance improvements. All test matrices are available from the University of Florida Sparse Matric Collection.

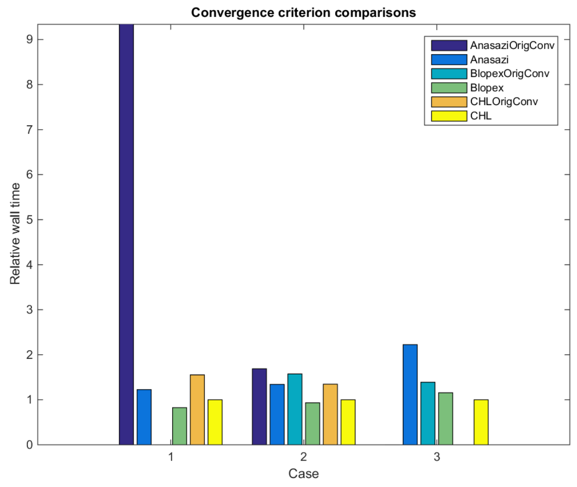
Case	**Matrix**	**Block width**	**Eigenpairs**	**Convergence threshold**
1	C60	194	176	1.0×10^{-4}
2	Si5H12	219	199	1.0×10^{-4}
3	Andrews	550	500	1.0×10^{-4}

We seek 1% of the eigenpairs with minimal eigenvalues. Basis blocks are padded by 10% and the convergence tolerance is set to 10^{-4}. These tests are performed using 24-core shared-memory Intel processors. Timing results are

Matrix	**AnasaziOC**	**Anasazi**	**BlopexOC**	**Blopex**	**CHLOC**	**CHL**
C60	947.40	21.02	**failed**	14.15	26.63	17.15
Si5H12	53.48	42.49	49.84	29.57	42.66	31.67
Andrews	**failed**	614.10	383.70	318.90	**failed**	276.01

The BlopexOC fail case generated an unrecoverable error during operation due to a negative pivot within Cholesky factorization. Both AnasaziOC and CHLOC fail cases had not converged after running for 4 hours.

Each implementation benefits from the new convergence criterion. In the case of Andrews, Blopex succeeds where the other algorithms fail because it allows for soft-locking out of order. As a result, Blopex misses the minimum eigenpair but succeeds in returning other eigenpairs within the allowed number of iterations. Since CHLOC and AnasaziOC force convergence in order, iteration cannot terminate as the minimum eigenvalue is too small to satisfy the relative residual threshold in



floating point arithmetic.

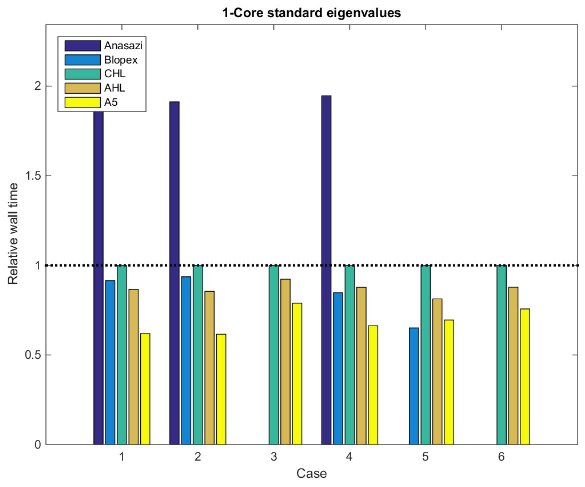
1.8.2 Fixed core performance

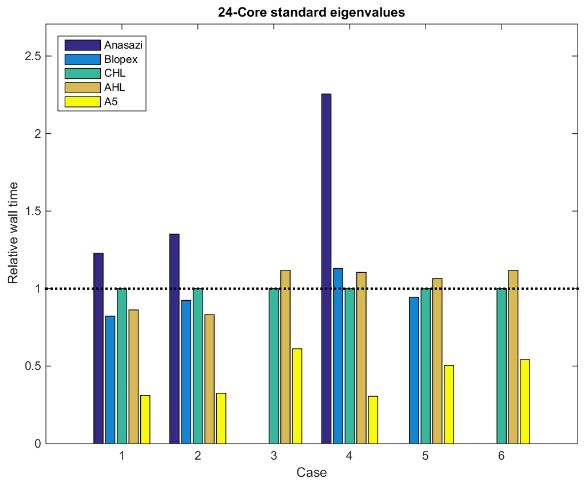
For all remaining comparisons we use the modified convergence condition. During the course of this research five main algorithm variations were developed and tested. A5 is our fifth and final version using the stability and performance modifications discussed. We also test an Aggressive implementation of Hetmaniuk-Lehoucq (AHL) using the implicit updates they suggest. We seek the minimum of 500 or 1% of the eigenpairs for each matrix. As before, basis blocks are padded by 10% and convergence tolerance is set to 10^{-4}.

Case	**Matrix**	**Block width**	**Eigenpairs**	**Conv. threshold**
1	C60	194	176	1.0×10^{-4}
2	Si5H12	219	199	1.0×10^{-4}
3	c_65	529	481	1.0×10^{-4}
4	Andrews	550	500	1.0×10^{-4}
5	Ga3As3H12	550	500	1.0×10^{-4}
6	Ga10As10H30	550	500	1.0×10^{-4}

These tests are performed on both 1 and 24 cores of 24-core shared memory Intel processors. Wall times are plotted relative to CHL because it is a reliable reference. The fail cases observed in these experiments are due to Cholesky factorization failure during orthogonalization.

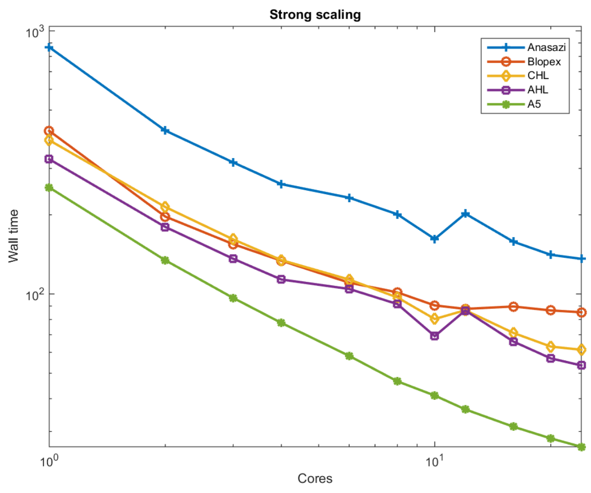
1-Core results:





Case	Anasazi	Blopex	CHL	AHL	A5
1	126.10	59.08	64.58	55.93	39.99
2	282.60	138.40	147.80	126.34	91.01
3	**failed**	**failed**	4535.37	4184.92	3577.42
4	3230.00	1406.00	1659.73	1456.42	1100.81
5	**failed**	2752.00	4228.64	3436.80	2938.86
6	**failed**	**failed**	11430.91	10034.57	8650.59

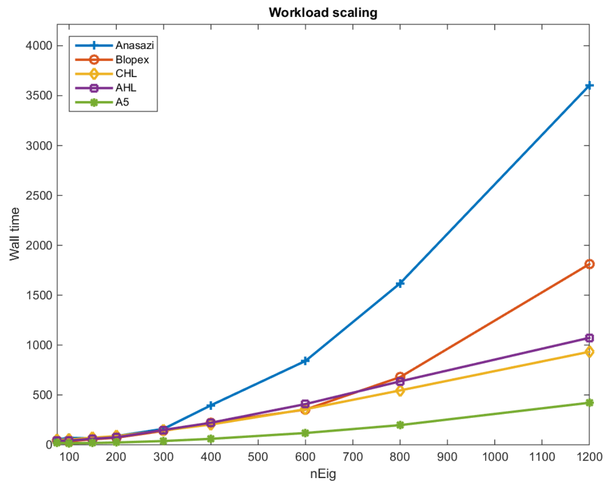
24-Core results:



Case	Anasazi	Blopex	CHL	AHL	A5
1	21.01	14.06	17.10	14.75	5.32
2	42.75	29.22	31.63	26.31	10.26
3	**failed**	**failed**	751.26	839.59	459.68
4	624.00	312.50	276.69	305.69	84.55
5	**failed**	569.30	602.70	642.13	304.34
6	**failed**	**failed**	1715.36	1918.28	930.69

1.8.3 Strong scaling

We compare strong scaling parallel performance using the standard symmetric eigenvalue problem Si5H12. This dimension of this matrix is 19896. We seek 199 eigenpairs which is 1% of the spectrum. We pad basis blocks by 10% to give 219 columns per block. The convergence threshold is set to 10^{-4}.



Cores	Anasazi	Blopex	CHL	AHL	A5
1	867.00	416.90	384.20	325.38	254.03
2	418.00	196.90	214.10	179.81	134.31
3	316.30	154.70	161.30	136.14	96.60
4	261.50	133.80	134.52	113.51	77.64
6	232.20	110.40	113.46	104.49	58.16
8	200.80	101.40	96.99	91.54	46.56
10	161.80	90.47	80.32	69.14	41.11
12	202.00	87.77	86.76	86.55	36.43
16	158.00	89.63	71.11	65.94	31.29
20	140.90	86.59	63.11	56.92	28.25
24	136.00	85.23	61.38	53.50	26.25

1.8.4 Scaling number of eigenpairs

We also examine the effect of scaling the number of sought eigenpairs and the corresponding block dimension using Andrews because all of the algorithms we test are successful on this matrix. As before, the block dimension is 10% larger than the number of eigenpairs sought and convergence tolerance is 10^{-4}.

Eigenpairs	**Anasazi**	**Blopex**	**CHL**	**AHL**	**A5**
75	52.93	40.47	50.75	41.51	19.48
100	69.92	34.06	52.40	42.95	15.68
150	64.65	55.44	71.05	60.58	18.01
200	87.37	71.02	87.01	74.80	23.00
300	160.10	139.70	143.77	148.98	37.99
400	396.20	218.70	203.77	220.89	59.83
600	841.00	353.20	356.82	408.53	118.32
800	1619.00	678.60	545.03	635.83	197.37
1200	3602.00	1811.00	932.56	1072.21	421.47

1.8.5 Parallel performance fitting

The following model is used to analyze strong scaling performance. Solution time using n processors is assumed to have the form

$$t(n) := \alpha\frac{t_1}{n} + (1-\alpha)t_1 + \beta(n-1)t_1.$$

This time can be separated into workload components undertaken by the most heavy-laden core. The first term is the fraction $\alpha \in [0,1]$ of single-core workload time t_1 that is efficiently divided among all n cores. The second term is remaining overhead that does not parallelize. The third term is the relative increase in workload due to communication and coordination with each other processors. $\beta \geq 0$.

Parameters α and β are solved using a nonnegative linear least-squares fit to experimental results. The optimal number of processors, $n_{\texttt{opt}} = \min(24, \sqrt{\frac{\alpha}{\beta}})$, and the optimal wall time $t(n_{\texttt{opt}})$ are compared.

	Anasazi	**Blopex**	**CHL**	**AHL**	**A5**
t_1	867.0	416.9	384.2	325.4	254.0
α	92.6%	93.3%	86.6%	85.0%	93.3%
β	0.286%	0.518%	0.000%	0.000%	0.000%
$n_{\texttt{opt}}$	18	13	24	24	24
$t_{\texttt{opt}}$	151.2	83.6	65.3	60.3	27.0

The meaning of these parameters and corresponding optima is clearly limited by the applicability of the model which is both problem-dependent and architecture-dependent. However, they still provide insight into how well each algorithm utilizes available processing capacity of the given machine.

We can also fit perfomance scaling as the number of eigenvectors sought changes. That is $\log(t(n_{\texttt{Eig}}))$ to $\log\left(\frac{n_{\texttt{Eig}}}{75}\right)$ approximates the order of leading term in time dependence.

$$t(n) := t_{75} \left(\frac{n}{75} \right)^p$$

	Anasazi	**Blopex**	**CHL**	**AHL**	**A5**
t_{75}	48.48	35.89	50.74	42.14	14.59
p	1.58	1.40	1.10	1.25	1.19

We believe the value of p is smallest for CHL because it avoids implicit updates. Implicit updates require matrix multiplication which scales as $n_{\texttt{Eig}}^2$. For large block dimensions, it would be more efficient to use direct updates which scale linearly with block dimension. However, in this range of problem sizes implicit updating is still faster.

Chapter 2

Spectral Target Residual Descent

2.1 Contributions and results

This work introduces a new algorithm, based on LOBPCG, that is designed to compute eigenvectors with eigenvalues near a spectral target in the interior of the spectrum of a symmetric matrix. The previous approach using LOBPCG to approximate such eigenvectors applies the algorithm to a shift-squared matrix. This technique produces a new matrix in which eigenvectors of interest have smallest eigenvalues which are suitable for approximation with LOBPCG. Unfortunately, convergence can be slow unless the target is set at a large gap in the spectrum. Furthermore, the shift-and-square approach using LOBPCG cannot be used to solve generalized symmetric eigenvalue problems.

2.1.1 Improving convergence

The purpose of this work is to improve convergence towards interior eigenvalue targets of large symmetric matrices. Eigenpairs of interest are framed as solutions to a spectral target objective function which is then minimized through gradient descent. This objective formulation is needed to ensure that eigenpair approximations do not retreat from the intended spectral target. Convergence is then accelerated by formulating a second set of objective functions based on the convergence detection computation. These additional objectives provide gradients that improve subspace expansion efficiency.

2.1.2 Generalized spectral targeting

The objective formulations used to derive this method generalize to non-identity metrics which share solutions with generalized eigenvalue problems. A generalized method is proposed and possible implementation strategies are discussed.

2.1.3 Results

Numerical experiments on standard symmetric eigenvalue problems show the proposed method typically converges in roughly half of the number of iterations required by LOBPCG on a shift-squared matrix.

This work will extend to naively parallel implementations of windowed eigenvalue solvers for large sparse symmetric matrices. Since interior targeting can be efficiently pursued, multiple processes can be configured to simultaneously and independently work on different spectral targets with very little synchronization among processes.

2.2 Introduction to interior eigenvalue targeting

The symmetric eigenvalue problem is often encountered in engineering and physics. Given a large symmetric matrix $A \in \mathbb{R}^{m\times m}$, the standard eigenvalue problem is written

$$AX = X\Lambda \quad \text{with unit-length eigenvectors} \quad X = \begin{bmatrix} x_1 & \cdots & x_m \end{bmatrix}$$

and eigenvalues $\Lambda = \text{diag}([\lambda_1, \cdots, \lambda_m])$. These eigenpairs can be sorted by distance from a spectral target σ. When the matrix A is very large the full eigenvalue decomposition becomes impractical and may be unnecessary. For many applications, only a small window of eigenvalues is needed near a spectral target of interest. These target-shifted eigenvalues can be written as $\delta_i = \lambda_i - \sigma$ for $i = 1, 2, \ldots, m$ and ordered so that $|\delta_1| \leq |\delta_2| \leq \cdots \leq |\delta_m|$. We explore the task of seeking a small window of $k \ll m$ eigenvector-eigenvalue pairs nearest σ in value [53]. These solutions (X^*, Δ^*) satisfy the shifted eigenvalue problem

$$(A - \sigma I)\, X^* = X^* \Delta^* \quad \text{where} \quad X^* = \begin{bmatrix} x_1 & \cdots & x_k \end{bmatrix} \quad \text{and} \quad \Delta^* = \begin{bmatrix} \delta_1 & & \\ & \ddots & \\ & & \delta_k \end{bmatrix}$$

An early attempt to solve interior eigenvalues of sparse symmetric matrices was submitted by Morgan in 1991.[51] Morgan notes that approximate eigenpair extraction based on Rayleigh-Ritz analysis fails for interior spectral targets. The alternative proposed uses a form of Rayleigh–Ritz analysis with implicit shift-inversion.

A related approach approximates shift-inversion using block Lanczos iteration [27]. Sleijpen *et al.* introduced the Jacobi–Davidson method in 1996, which approximates Rayleigh-Quotient Iteration [62, 63]. This method requires approximating the solution to a linear system for every eigenpair sought on every iteration.

An obvious approach based on Knyzev's Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) [41] simply approximates the lowest eigenvalues of the Shifted-Squared matrix (SSLOBPCG):

$$(A - \sigma I)^2 X^* = X^* \Delta^{*2}.$$

LOBPCG is an iterative method that computes a sequence of approximations

$$(X^{(j)}, \Delta^{(j)}) \quad \text{for} \quad j = 1, 2, \ldots$$

until all sought eigenpairs satisfy a convergence threshold

$$\frac{\|r_i^{(j)}\|_2}{\left(\|(A - \sigma I)\|_2 + |\delta_i^{(j)}|\right) \|x_i^{(j)}\|_2} \leq \tau_c \quad \text{for} \quad i = 1, \ldots, k. \tag{2.1}$$

This method is well-suited to resolving eigenpairs at either extreme of the spectrum. This research proposes a modification that attempts to improve convergence over what can be obtained by SSLOBPCG.

In Section 2.3, analysis and motivation are developed by examining relevant objective functions to be optimized. Using two residual-minimizing objectives, subspace extraction and expansion methods are derived. These methods are then implemented in an algorithm titled Spectral Target Residual Descent (STRD). Section 2.4 examines analogous objective functions for a generalized spectral target $(A - \sigma B) X^* = BX^* \Delta^*$. An algorithm requiring internal iterations of Preconditioned Conjugate Gradient (PCG) is proposed. In Section 2.5, numerical experimental results comparing STRD with SSLOBPCG are given.

2.3 Spectral targeting analysis

Each iteration of LOBPCG operates in two phases: subspace extraction and subspace expansion. Iteration j begins with $X^{(j)} \in \mathbb{R}^{m \times n_x}$ representing current approximate eigenvectors. Convergence may be improved by including padding in the dimension of approximate eigenvectors over the number of eigenvectors sought so that $n_x > k$. A basis $S^{(j)}$ is constructed for a subspace that will be searched to obtain better approximations $X^{(j+1)} \in \operatorname{span}(S^{(j)})$. That is,

$$X^{(j+1)} = \texttt{extractionFunction}(S^{(j)}) \quad \text{where} \quad S^{(j)} = \begin{bmatrix} X^{(j)} & Y^{(j)} \end{bmatrix}.$$

Composition and analysis of additional search basis vectors $Y^{(j)}$ are considered in Section 2.3.4. To maintain both simplicity and numerical stability, columns of $S^{(j)}$ are always constructed to be orthonormal $S^{(j)T}S^{(j)} = I$ [33]. Typically extraction is accomplished with the Rayleigh–Ritz procedure, although variations on this technique have been explored by Hochstenbach and Sleijpen [35]. Solutions have the form $X^{(j+1)} = S^{(j)}{C_x}^{(j+1)}$ and are substituted into the eigenvalue problem which is projected onto $S^{(j)}$. This produces the reduced symmetric eigenvalue problem

$$\left(S^{(j)T}AS^{(j)}\right){C_x}^{(j+1)} = {C_x}^{(j+1)}\Theta^{(j+1)} \quad \text{solving for} \quad {C_x}^{(j+1)} \quad \text{and} \quad \Theta^{(j+1)}.$$

Eigenpairs are then sorted by eigenvalue to obtain n_x primitive vectors ${C_x}^{(j+1)}$ from an extreme end of the spectrum. Taking the lowest eigenvalues guarantees that Rayleigh–Ritz produces monotonicaly non-increasing approximations. This easily follows from the Courant-Fischer min-max theorem restated here. Let α_i be eigenvalues of A *sorted in ascending order* $\alpha_1 \le \alpha_2 \le \cdots \le \alpha_m$. If U^i and V^{m-i+1} are i and $m-i+1$ dimensional subspaces of $\mathbb{R}^m$ respectively then

$$\alpha_i = \min_{U^i \subset \mathbb{R}^m} \max_{0 \ne z \in U^i} \frac{z^TAz}{z^Tz} = \max_{V^{m-i+1} \subset \mathbb{R}^m} \min_{0 \ne z \in V^{m-i+1}} \frac{z^TAz}{z^Tz}.$$

By taking the k minimal eigenvalues from the Rayleigh–Ritz projection, we have

$$\alpha_i \le \theta_i^{(j+1)} = \min_{U^i \subset S^{(j)}} \max_{0 \ne z \in U^i} \frac{z^TAz}{z^Tz} \le \frac{x_i^{(j)T}Ax_i^{(j)}}{x_i^{(j)T}x_i^{(j)}} = \theta_i^{(j)} \quad \text{since} \quad x_i^{(j)} \in S^{(j)} \subset \mathbb{R}^m.$$

Likewise, taking the maximal eigenvalues would give a non-decreasing sequence of eigenvalue approximations. This is the extremizing property of iterative Rayleigh–Ritz on a sequence of subspaces each of which contains previous extrema.

2.3.1 Phantom eigenvalues

In order to target the interior of the spectrum of A, an enticing strategy is to perform Rayleigh–Ritz and sort by distance from the spectral target σ. Unfortunately, this technique is unsuccessful.

To understand why interior convergence fails, consider a unit-length linear combination z of two eigenvectors x_a and x_b such that λ_a and λ_b respectively fall below and above the spectral window of interest:

$$z = \cos(\varphi)x_a + \sin(\varphi)x_b \quad \text{where} \quad \lambda_a < \min_{i=1,\dots,n_x} \lambda_i \quad \text{and} \quad \lambda_b > \max_{i=1,\dots,n_x} \lambda_i.$$

The corresponding Rayleigh quotient is

$$\theta(z) = z^T A z = \cos(\varphi)^2 \lambda_a + \sin(\varphi)^2 \lambda_b \quad \text{which yields} \quad \theta(z) \in [\lambda_a, \lambda_b].$$

This is an example of what might be called a phantom Ritz vector. These are useless combinations of eigenvectors with both large and small eigenvalues. Although z has no linear dependence with any eigenvector of interest, it can duplicate any eigenvalue within the spectral window depending on φ. As a result, Rayleigh-Ritz sorted by distance from σ will dispose of more useful approximations in favor of vectors such as z simply because the Rayleigh quotient is close to the target.

2.3.2 Spectral targeting objective

Clearly, the spectral target distance from the Rayleigh quotient of an approximate eigenvector is not an adequate objective to minimize. Suppose we construct a phantom z to have a Rayleigh-quotient that exactly matches the target σ. The residual would be given by $r = (A - \sigma I)z$, which would give a residual 2-norm

$$\|r\|_2^2 = z^T (A - \sigma I)^2 z = \cos(\varphi)^2(\lambda_a - \sigma)^2 + \sin(\varphi)^2(\lambda_b - \sigma)^2 \geq \min\left((\lambda_a - \sigma)^2, (\lambda_b - \sigma)^2\right)$$

We see that even if the Rayleigh quotient were an exact match, the residual 2-norm would have shown the vector z to contain components that are far from the spectral target. This observation leads to the definition of the spectral target residual

$$t_i^{(j)} = (A - \sigma I)\, x_i^{(j)}. \tag{2.2}$$

Note that σ is the spectral target and is held constant for all target residual computations rather than using the Rayleigh quotient as is the case for the standard residual definition.

The sum of squares of target residual 2-norms corresponding to an approximation $X^{(j)}$ would be

$$f(X^{(j)}) = \operatorname{trace}\left(X^{(j)T} (A - \sigma I)^2 X^{(j)}\right). \tag{2.3}$$

The objective function 2.3—based on spectral target residual minimization—is equivalent to the LOBPCG objective on the shift-squared matrix. The minimizer of this objective gives eigenvectors with values nearest σ. To show this, let $X^* = XC^*$ where $X \in \mathbb{R}^{m\times m}$ is the full matrix of unit-eigenvectors and $C^* \in \mathbb{R}^{m\times k}$ is the minimizing orthogonal column matrix so that $C^{*T}C^* = I$. We take corresponding components $H \in \mathbb{R}^{m\times k}$ to be arbitrary orthonormal perturbations which are only constrained to remain orthogonal to the minimizer $H^T H = I$ and $H^T C^* = 0$. This gives perturbed candidates of the form $\hat{X} = X(C^* + \epsilon H)$ where $\epsilon \in \mathbb{R}$ is a scaling constant.

The variational principle requires that all derivatives of the objective function, represented by derivatives with respect to ϵ, must vanish at the optimizer. That is

$$\frac{\partial}{\partial \epsilon} f(\hat{X}) = 0 \quad \text{or} \quad \operatorname{trace}\left(H^T(\Lambda - \sigma I)^2 C^*\right) = 0.$$

Since H could be any basis within the complement of span(C^*), the only way this condition can hold is if $(\Lambda - \sigma I)^2 C^* = C^* \hat{C}$ or $C^{*T}(\Lambda - \sigma I)^2 C^* = \hat{C}$ for some component matrix $\hat{C}$. This would give the objective function $f(X^*) = \operatorname{trace}(\hat{C})$. The objective is minimized by sequentially constructing orthonormal columns c_j^* in C^* that result in the smallest magnitude row scaling $(\Lambda - \sigma I)c_j^*$. This produces components corresponding to eigenvalue offsets $\delta_1 = \lambda_1 - \sigma$ through $\delta_k = \lambda_k - \sigma$ sorted by distance from the target σ.

2.3.3 Subspace extraction

A reliable subspace extraction method makes use of the target residual minimization objective 2.3. We simply seek optimal candidates from within the search subspace span $S^{(j)}$. That is, $X^{(j+1)} = S^{(j)} {C_x}^{(j+1)}$ where ${C_x}^{(j+1)} \in \mathbb{R}^{n_s \times n_x}$ gives optimal candidate components. Following the same analysis from the previous section, these optimal components correspond to the eigenvectors of $S^{(j)T} \left(A - \sigma I\right)^2 S^{(j)}$ with minimal eigenvalues.

Because this process is equivalent to what would occur if LOBPCG were applied to the target shift-squared matrix $\left(A - \sigma I\right)^2$, it naturally guarantees that subsequent approximations of eigenvectors within the target window are no worse than previous approximations by a trivial modification of the proof using the Courant-Fischer min-max theorem. However unlike LOBPCG, this style of analysis generalizes to interior targeting of matrix pencils or generalized eigenvalue problems discussed further in Section 2.4.1.

A second extraction, Rayleigh-Ritz, is then performed using only the n_x minimizers from the first extraction with the shifted—but not squared—matrix $(A - \sigma I)$. This isolation phase decouples eigenvectors with positive and negative offsets. Although this isolation does not change the search subspace used on subsequent iterations, it is required to compute residuals and detect convergence.

2.3.4 Subspace expansion

To achieve convergence towards eigenpairs near the target, we need to capture the local gradient of the objective function 2.3. Let $X^{(j)}$ be our last approximate orthonormal eigenvectors. As before, we consider an arbitrary perturbation $H \in \mathbb{R}^{n \times n_x}$ under the only restrictions $H^T H = I$ and $H^T X^{(j)} = 0$, which are sensible because we are only interested in dimensions that would expand the search subspace $\text{span}(S^{(j)})$. Perturbations on the local approximation are then $\hat{X} = X^{(j)} + \epsilon H$ where $\epsilon \in \mathbb{R}$ is a scaling constant.

Differentiating the objective function with respect to ϵ produces

$$\frac{\partial}{\partial \epsilon} f(\hat{X}) = \text{trace}\left(H^T \left(A - \sigma I\right)^2 X^{(j)}\right)$$

which indicates that the direction of steepest descent is maximized by some linear combination

$$H_{\text{sd}} \in \text{span}\left(\begin{bmatrix} X^{(j)} & \left(A - \sigma I\right)^2 X^{(j)} \end{bmatrix}\right)$$

that isolates components of $\left(A - \sigma I\right)^2 X^{(j)}$ that are orthogonal to $X^{(j)}$. The residual matrix corresponding to shift-squared LOBPCG equivalently computes

$$R_{\text{ss}} = \left(A - \sigma I\right)^2 X^{(j)} - X^{(j)} (\Delta^{(j)})^2 \quad \text{giving} \quad H_{\text{sd}} \in \text{span}\left(\begin{bmatrix} X^{(j)} & R_{\text{ss}} \end{bmatrix}\right).$$

Notably, each shift-squared residual is a critical direction of the quadratic form

$$q(x_i^{(j)}) = x_i^{(j)T} \left(\left(A - \sigma I\right)^2 - \delta_i^{(j)^2} \right) x_i^{(j)}.$$

However, this quadratic form is not a useful objective since it is already zero whenever $(\delta_i^{(j)})^2$ is the Rayleigh quotient. Although this basis expansion provides descent towards eigenpairs near the target, it misses a second key measurement in the algorithm: the convergence condition (2.1).

The objectives required to claim that each eigenpair has converged scale with the ordinary residual 2-norms. We require gradients of these residual norms to speed satisfaction of the convergence conditions. Residual 2-norm-squared objectives are

$$g_i^{(j)}(x) = x^T (A - \theta_i^{(j)} I)^2 x \quad \text{where} \quad \theta_i^{(j)} = \frac{x_i^{(j)T} A x_i^{(j)}}{x_i^{(j)T} x_i^{(j)}}.$$

Each corresponding dimension of steepest descend is then

$$h_{\text{sd},i}^{(j)} \in \text{span}\left(\begin{bmatrix} x_i^{(j)} & (A - \theta_i^{(j)} I)^2 x_i^{(j)} \end{bmatrix}\right)$$

which further expands into components $h_{\mathrm{sd},i}^{(j)} = \alpha x_i^{(j)} + \beta r_i^{(j)} + \gamma \left(A - \sigma I\right) r_i^{(j)}$. The two expansion basis blocks below ensure that the gradients of *both* the ordinary residual norms and target residual norms are captured in the subsequent search subspace. LOBPCG applied to a shift-squared matrix only expands the search basis by n_x vectors $R_{\mathrm{SS}} = \left(A - \sigma I\right)^2 X^{(j)} - X^{(j)}(\Delta^{(j)})^2$ rather than $2n_x$ vectors

$$R^{(j)} = AX^{(j)} - X^{(j)}\Theta^{(j)} \quad \text{and} \quad V^{(j)} = \left(A - \sigma I\right) R^{(j)}. \tag{2.4}$$

As usual, both of these basis expansions need to be orthonormalized before the next extraction phase. Furthermore, aggregate eigenvector updates P are still retained from prior extraction results as is the case for LOBPCG. STRD is outlined in Algorithms 9, 10, and 11 for the standard eigenvalue problem.

Algorithm 9 Spectral Target Residual Descent

Input:

$X^{(0)}$ is an $m \times k$ matrix of initial approximate eigenvectors.

$k \leq m$ is the number of converged eigenvectors requested.

σ is the spectral target. Eigenvectors are sought with eigenvalues nearest σ.

Output:

X^* is the $m \times k$ matrix of approximate eigenvectors.

Λ^* is the $k \times k$ diagonal matrix of approximate eigenvalues.

```
 1: function $[X^*, \Lambda^*]=\texttt{strd}(X^{(0)}, k, \sigma)$
 2:     $[C^{(1)}, \Delta^{(1)}] = \texttt{TargetRayleighRitz}(X^{(0)}, \sigma)$
 3:     $X^{(1)} = X^{(0)}C^{(1)};$
 4:     $P^{(1)} = [\,];$
 5:     $R^{(1)} = (A - \sigma I)\,X^{(1)} - X^{(1)}\Delta^{(1)};$
 6:     $V^{(1)} = (A - \sigma I)\,R^{(1)};$
 7:     $n_c = 0;$
 8:     do $j = 1, 2, \ldots$
 9:         $Y^{(j)} = \texttt{orthoDrop}\left(I, [R^{(j)}\ V^{(j)}], [X^{(j)}\ P^{(j)}]\right);$
10:         $S^{(j)} = \left[X^{(j)}\ P^{(j)}\ Y^{(j)}\right];$
11:         $[C^{(j+1)}, \Delta^{(j+1)}] = \texttt{standardTargetExtract}(S^{(j)}, k, n_c);$
12:         $[X^{(j+1)}\ P^{(j+1)}] = S^{(j)}C^{(j+1)};$
13:         $R^{(j+1)} = (A - \sigma I)\,X^{(j+1)} - X^{(j+1)}\Delta^{(j+1)};$
14:         $V^{(j+1)} = (A - \sigma I)\,R^{(j+1)};$
15:         Determine number of converged eigenpairs $n_c$.
16:     while $n_c < k$
17:     Return converged eigenpairs $X^* = X^{(j+1)}$ and $\Lambda^* = \Delta^{(j+1)} + \sigma I;$
18: end function
```

Algorithm 10 Initial spectral target Rayleigh–Ritz

```
Input:
    $X^{(0)}$ is $m \times k$ matrix basis for the initial search subspace.
    $\sigma$ is the spectral target of $A$.
    Columns must be linearly independent and well-conditioned.
Output:
    $C^{(1)}$ is $k \times k$ solution components for $X^{(1)}$.
    $\Delta^{(1)}$ is $k \times k$ diagonal matrix of Rayleigh quotients for target-shifted matrix.
    These satisfy $C^{(1)T}X^{(0)T}X^{(0)}C^{(1)} = I$ and $C^{(1)T}X^{(0)T}(A-\sigma I)X^{(0)}C^{(1)} = \Delta^{(1)}$.
1: function $[C^{(1)}, \Delta^{(1)}]$=TargetRayleighRitz($X^{(0)}$,$\sigma$)
2:     $D = \left(\texttt{diag}(X^{(0)T}X^{(0)})\right)^{-\frac{1}{2}}$;
3:     Cholesky factorize $R^TR = DX^{(0)T}X^{(0)}D$.
4:     Solve $\left(R^{-T}DX^{(0)T}(A-\sigma I)X^{(0)}DR^{-1}\right)Z = Z\Delta^{(1)}$.
5:     $C^{(1)} = DR^{-1}Z$;
6: end function
```

Algorithm 11 Standard spectral target extraction and isolation

```
Input:
    $S^{(j)}$ is $m \times n_s$ matrix forming an orthonormal basis for the search subspace.
    $n_c$ is the number of previous eigenpairs satisfying the convergence condition.
Output:
    $C^{(j+1)}$ is $n_s \times (2n_x - n_c)$. First $n_x$ columns are ${C_x}^{(j+1)}$ followed by ${C_p}^{(j+1)}$.
    $\Delta^{(j+1)}$ is $n_x \times n_x$ diagonal matrix giving $\delta_1^{(j+1)}, \cdots, \delta_{n_x}^{(j+1)}$.
    Output satisfies $C^{(j+1)T}C^{(j+1)} = I$ and $C^{(j+1)T}\left(S^{(j)T}(A-\sigma I)S^{(j)}\right)C^{(j+1)} = \Delta^{(j+1)}$.
1: function $[C^{(j+1)}, \Delta^{(j+1)}]$=standardTargetExtract($S^{(j)}$,$k$,$n_c$)
2:     Solve $\left(S^{(j)T}(A-\sigma I)^2 S^{(j)}\right)Z = Z\Theta$.
3:     QR factorize aggregate update components
           $$Q_pR_p = \begin{bmatrix} Z_{2\perp}^T & Z_{3\perp}^T \end{bmatrix} \begin{bmatrix} Z_2(:, n_c+1:k) \\ Z_3(:, n_c+1:k) \end{bmatrix}.$$
4:     Isolate eigenpairs from non-squared target
           $$Z_x^T\left(S^{(j)T}(A-\sigma I)S^{(j)}\right)Z_xQ_x = Q_x\Delta^{(j+1)}.$$
5:     $[{C_x}^{(j+1)}\ {C_p}^{(j+1)}] = [Z_xQ_x\ Z_\perp Q_p(:, 1:n_x-n_c)]$;
6: end function
```

2.4 Notes on generalized spectral targeting

2.4.1 Generalized targeting objective

Solving the target-shifted and squared system using LOBPCG is not an option for generalized eigenvalue problems with a symmetric positive definite metric B such that

$$AX = BX\Lambda \quad \text{where} \quad X^T BX = I$$

because the eigenpairs of $(A - \sigma B)^2$ are *not* equivalent to eigenvectors X having shift-squared eigenvalues $(\Lambda - \sigma I)^2$. Observe that

$$X^T (A - \sigma B)^2 X = \Delta X^T B^2 X \Delta \neq \Delta^2.$$

This would correspond to minimizing 2-norms $\|t_i^{(j)}\|_2$ of the generalized spectral target residuals $t_i^{(j)} = (A - \sigma B)\, x_i^{(j)}$.

We require a generalized objective function that can be optimized to extract components from a search subspace. However, it must retain the property that the minimizer produces generalized eigenvectors with eigenvalues nearest the spectral target σ. The objective function must satisfy

$$\arg\min_X f(X) = X^* \quad \text{where} \quad (A - \sigma B)\, X^* = BX^* \Delta^* \quad \text{and} \quad |\delta_i| \leq |\delta_j|$$

for $i = 1, \ldots, n_x$ and $j = n_x + 1, \ldots, m$. A modification meeting this requirement minimizes generalized spectral target residuals in the B^{-1} norm.

$$f(X) = \operatorname{trace}\left(X^T (A - \sigma B)\, B^{-1} (A - \sigma B)\, X\right) \quad \text{where} \quad X^T BX = I \tag{2.5}$$

Using analysis identical to Section 2.3.2, we find that the minimizer produces eigenpairs of the matrix $((A - \sigma B)\, B^{-1} (A - \sigma B))$ which coincide exactly with the generalized eigenpairs nearest σ:

$$(X^*)^T (A - \sigma B)\, B^{-1} (A - \sigma B)\, X^* = \Delta^* (X^*)^T B B^{-1} B X^* \Delta^* = \Delta^{*2}.$$

Note that $B = I$ recovers the standard objective function from the previous section.

2.4.2 Generalized extraction

The analogous extraction method for generalized spectral targeting solves the objective optimization problem 2.5 restricted to the search subspace span$(S^{(j)})$. This requires solving a standard eigenvalue problem, provided $S^{(j)}$ is orthonormal in the metric B:

$$\left(S^{(j)T}\left(A-\sigma B\right)B^{-1}\left(A-\sigma B\right)S^{(j)}\right)C = C\Theta \quad \text{where} \quad S^{(j)T}BS^{(j)} = I.$$

The difficulty with this method is in constructing inner products in the metric B^{-1}. Let $Y = \left(A-\sigma B\right)S^{(j)}$. We require Ψ such that $B\Psi = Y$. This gives the inner product matrix we need as $\mathbb{X} = Y^T\Psi$. If inverting B were feasible then it would be easier to simply express this as a standard eigenvalue problem. Since that is not the case, each generalized extraction iteration will require an internal iteration sequence to approximate Ψ.

Solving this problem will likely require a strong preconditioner $K \approx B$—where K^{-1} is computable—combined with a slight variation on Preconditioned Conjugate Gradient (PCG). A substantial amount of research has gone into efficient computation of such preconditioners [20, 59]. If we have a complete B-orthonormal basis U given in blocks then we can construct a solution

$$\Psi = UC \quad \text{where} \quad U = \begin{bmatrix} U_1 & \cdots & U_{n_u} \end{bmatrix} \quad \text{and} \quad U^TBU = I \quad \text{with} \quad C = \begin{bmatrix} C_1 \\ \vdots \\ C_{n_u} \end{bmatrix}.$$

This would produce the inner product matrix $\mathbb{X} = C^TC$. Fortunately, this problem will become easier to solve near convergence. If approximate eigenvectors have the form $X^{(j)} = X^* + E$ where $\|E\| \ll \|X^{(j)}\|$, then the inverse needed can be well-approximated

$$B^{-1}\left(A-\sigma B\right)X^{(j)} = X^*\Delta^* + B^{-1}AE - \sigma E = X^{(j)}\Delta^* + B^{-1}AE - E\Lambda^* \approx X^{(j)}\Delta^*.$$

Since $X^{(j)} \subset S^{(j)}$ and $S^{(j)}$ is already B-orthonormal, taking the first basis as $U_1 = S^{(j)}$ should provide a useful initial approximate solution. However, stopping at this approximate solution would simply square the matrix used in Rayleigh–Ritz analysis and reproduce phantom eigenpairs that prevent convergence to an internal spectral target. Several iterations of PCG will likely be needed to obtain a strong enough approximation of $\mathbb{X}$. Another benefit of using PCG follows in Section 2.4.3.

Alternatively, the problem could be considered in terms of selected inversion. [47] Taking the compact QR decomposition $Y = Q_YR_Y$ so that $Q_Y \in \mathbb{R}^{m\times n_s}$ has orthonormal columns, we simply require components of B^{-1} corresponding to $\mathbb{X} = R_Y^T\left(Q_Y^TB^{-1}Q_Y\right)R_Y$.

As in the standard eigenvalue case, the first extraction would be followed by a second extraction using ordinary Rayleigh-Ritz with the n_x minimizers from the first extraction.

2.4.3 Generalized expansion

The subspace expansion employed in the general case must include gradients for both the targeted residual objective and the ordinary residual objective used to detect convergence. Analysis analogous to the standard targeting case produces directions of steepest descent for targeted residuals:

$$H_{\text{sd}} \in \operatorname{span}\left(\begin{bmatrix} X^{(j)} & (A-\sigma B)\, B^{-1}\, (A-\sigma B)\, X^{(j)} \end{bmatrix}\right).$$

It may be possible to incorporate results from generalized extraction if PCG had been used to approximate $\hat{\Psi} \approx \Psi = B^{-1}\,(A-\sigma B)\, S^{(j-1)}$. Then the expansion basis could be approximated as

$$\hat{H}_{\text{sd}} \in \operatorname{span}\left(\begin{bmatrix} X^{(j)} & U^{(j)} \end{bmatrix}\right) \quad \text{with} \quad U^{(j)} = (A-\sigma B)\, \hat{\Psi} S^{(j-1)} {C_x}^{(j)}.$$

Fortunately, the generalized gradients corresponding to the convergence condition objective 2.1 are easier to compute. Generalized residual objective functions are

$$g_i^{(j)}(x) = x^T (A - \theta_i^{(j)} B)^2 x \quad \text{where} \quad \theta_i^{(j)} = \frac{x_i^{(j)T} A x_i^{(j)}}{x_i^{(j)T} B x_i^{(j)}}.$$

Each dimension of steepest descend is $h_{\text{sd},i}^{(j)} \in \operatorname{span}([x_i^{(j)}\ (A - \theta_i^{(j)} B)^2 x_i^{(j)}])$, which can be expressed using the bases $V^{(j)}$ where

$$R^{(j)} = AX^{(j)} - BX^{(j)}\Theta^{(j)} \quad \text{and} \quad V^{(j)} = AR^{(j)} - BR^{(j)}\Theta^{(j)} \tag{2.6}$$

Including both $U^{(j)}$ and $V^{(j)}$ in expansion blocks would be sufficient to capture the gradients of each objective.

These ideas are combined in a proposed outline for generalized STRD in Algorithms 12, 13, and 14.

Algorithm 12 Generalized Spectral Target Residual Descent

```
Input:
    $X^{(0)}$ is an $m \times k$ matrix of initial approximate eigenvectors.
    $k \leq m$ is the number of converged eigenvectors requested.
    $\sigma$ is the spectral target. Eigenvectors are sought with eigenvalues nearest $\sigma$.
Output:
    $X^{*}$ is the $m \times k$ matrix of approximate eigenvectors.
    $\Lambda^{*}$ is the $k \times k$ diagonal matrix of approximate eigenvalues.
 1: function $[X^{*}, \Lambda^{*}]$=strd($X^{(0)}, k, \sigma$)
 2:     $[C^{(1)}, \Delta^{(1)}] = $ GeneralTargetRayleighRitz($X^{(0)}, \sigma$)
 3:     $X^{(1)} = X^{(0)} C^{(1)}$;
 4:     $P^{(1)} = [\,]$;
 5:     $U^{(1)} = [\,]$;
 6:     $R^{(1)} = (A - \sigma B)\, X^{(1)} - B X^{(1)} \Delta^{(1)}$;
 7:     $V^{(1)} = (A - \sigma B)\, R^{(1)}$;
 8:     $W^{(1)} = B R^{(1)}$;
 9:     $n_c = 0$;
10:     do $j = 1, 2, \ldots$
11:         $Y^{(j)} = $ orthoDrop $\left(I, [U^{(j)}\ V^{(j)}], [X^{(j)}\ P^{(j)}]\right)$;
12:         $S^{(j)} = \left[X^{(j)}\ P^{(j)}\ Y^{(j)}\right]$;
13:         $[C^{(j+1)}, \Delta^{(j+1)}, U^{(j+1)}] = $ GeneralTargetExtract($S^{(j)}, k, n_c$);
14:         $[X^{(j+1)}\ P^{(j+1)}] = S^{(j)} C^{(j+1)}$;
15:         $R^{(j+1)} = (A - \sigma B)\, X^{(j+1)} - B X^{(j+1)} \Delta^{(j+1)}$;
16:         $V^{(j+1)} = (A - \sigma B)\, R^{(j+1)} - B R^{(j+1)} \Delta^{(j+1)}$;
17:         Determine number of converged eigenpairs $n_c$.
18:     while $n_c < k$
19:     Return converged eigenpairs $X^{*} = X^{(j+1)}$ and $\Lambda^{*} = \Delta^{(j+1)} + \sigma I$;
20: end function
```

Algorithm 13 General Spectral Target Rayleigh–Ritz procedure

Input:

$X^{(0)}$ is $m \times k$ matrix basis for the initial search subspace.
σ is the spectral target of A.
Columns must be linearly independent and well-conditioned.

Output:

$C^{(1)}$ is $k \times k$ solution components for $X^{(1)}$.
$\Delta^{(1)}$ is $k \times k$ diagonal matrix of Rayleigh quotients for target-shifted matrix.
These satisfy $C^{(1)T} X^{(0)T} B X^{(0)} C^{(1)} = I$ and $C^{(1)T} X^{(0)T} \left(A - \sigma B\right) X^{(0)} C^{(1)} = \Delta^{(1)}$.

```
1: function [$C^{(1)}, \Delta^{(1)}$]=GeneralTargetRayleighRitz($X^{(0)}$,$\sigma$)
2:     $D = \left(\mathtt{diag}(X^{(0)T} B X^{(0)})\right)^{-\frac{1}{2}}$;
3:     Cholesky factorize $R^T R = D X^{(0)T} B X^{(0)} D$.
4:     Solve $\left(R^{-T} D X^{(0)T} \left(A - \sigma B\right) X^{(0)} D R^{-1}\right) Z = Z \Delta^{(1)}$.
5:     $C^{(1)} = D R^{-1} Z$;
6: end function
```

Algorithm 14 General spectral target extraction

```
Input:
    $S^{(j)}$ is $m \times n_s$ matrix forming an orthonormal basis for the search subspace.
    $n_c$ is the number of previous eigenpairs satisfying the convergence condition.
Output:
    $C^{(j+1)}$ is $n_s \times (2k - n_c)$. First $k$ columns are ${C_x}^{(j+1)}$ followed by ${C_p}^{(j+1)}$.
    $\Delta^{(j+1)}$ is $k \times k$ diagonal matrix giving $\delta_1^{(j+1)}, \cdots, \delta_k^{(j+1)}$.
1:  $U^{(j+1)}$ is prepared basis for next expansion phase.
    Output satisfies $C^{(j+1)T}C^{(j+1)} = I$ and $C^{(j+1)T}\left(S^{(j)T}(A - \sigma B)S^{(j)}\right)C^{(j+1)} = \Delta^{(j+1)}$.
2:  function $[C^{(j+1)}, \Delta^{(j+1)}, U^{(j+1)}]$=GeneralTargetExtract($S^{(j)}$,$k$,$n_c$)
3:      Use PCG to obtain $\hat{\Psi} \approx B^{-1}(A - \sigma B)S^{(j)}$.
4:      Solve $\left(S^{(j)T}(A - \sigma B)\hat{\Psi}\right)Z = Z\Theta$.
5:      QR factorize aggregate update components
            $$Q_p R_p = \left[\begin{array}{cc} Z_{2\perp}^T & Z_{3\perp}^T \end{array}\right]\left[\begin{array}{c} Z_2(:, n_c+1:k) \\ Z_3(:, n_c+1:k) \end{array}\right].$$
6:      Isolate eigenpairs from non-squared target $Z_x^T\left(S^{(j)T}(A - \sigma B)S^{(j)}\right)Z_x Q_x = Q_x\Delta^{(j+1)}$.
7:      $[{C_x}^{(j+1)}\ {C_p}^{(j+1)}] = [Z_x Q_x\ Z_\perp Q_p(:, 1:n_x - n_c)]$;
8:      $U^{(j+1)} = (A - \sigma B)\hat{\Psi}{C_x}^{(j+1)}$.
9:  end function
```

2.5 Numerical experiments

The first three tests are performed on the matrix Si5H12 available on UFSMC. These tests set the spectral targets at $\sigma = 2.5$, $\sigma = 2.8$, and $\sigma = 3.1$, demonstrating how a naively parallel solution might be employed on multiple nodes of a distributed-memory machine. The number of iterations of STRD required to achieve convergence are compared with Shift-Squared LOBPCG.

STRD tests use an individual block basis size $n_x = 60$. Since STRD uses 4 basis blocks in each subspace extraction whereas SSLOPBCG uses only 3, a fair point of contention would be that STRD benefits from simply having a larger extraction subspace on each iteration. In order to provide counterbalance and make an abundantly fair comparison, SSLOBPCG is tested with a larger block-size giving subspaces of equivalent dimension $3 \times 80 = 4 \times 60$ for each method. Both methods only iterate until $k = 50$ eigenpairs have converged to $\tau_c = 10^{-4}$.

In the interest of producing comparisons that are as clear as possible, LOBPCG is modified to perform an additional isolation extraction restricted to approximate eigenvectors $X^{(j)}$ as

$$X^{(j)T}\left(A - \sigma I\right) X^{(j)T} C_x = C_x \Delta.$$

These results are used to show approximate eigenvalues and to compute the convergence measures in Equation 2.1. Results are shown in Figures 2.1, 2.2, and 2.3.

The last test uses a larger matrix CO, which is also available on UFSMC. As before, SSLOBPCG uses 3×80 basis columns while STRD uses 4×60 and both seek $k = 50$ eigenpairs at a convergence threshold $\tau_c = 10^{-4}$. Since iterations were quite time consuming for this test case, SSLOBPCG was halted after the same number of iterations as STRD. It is clear from the graph that no eigenpairs in SSLOBPCG were near the convergence threshold at that time. Results are shown in Figure 2.4.

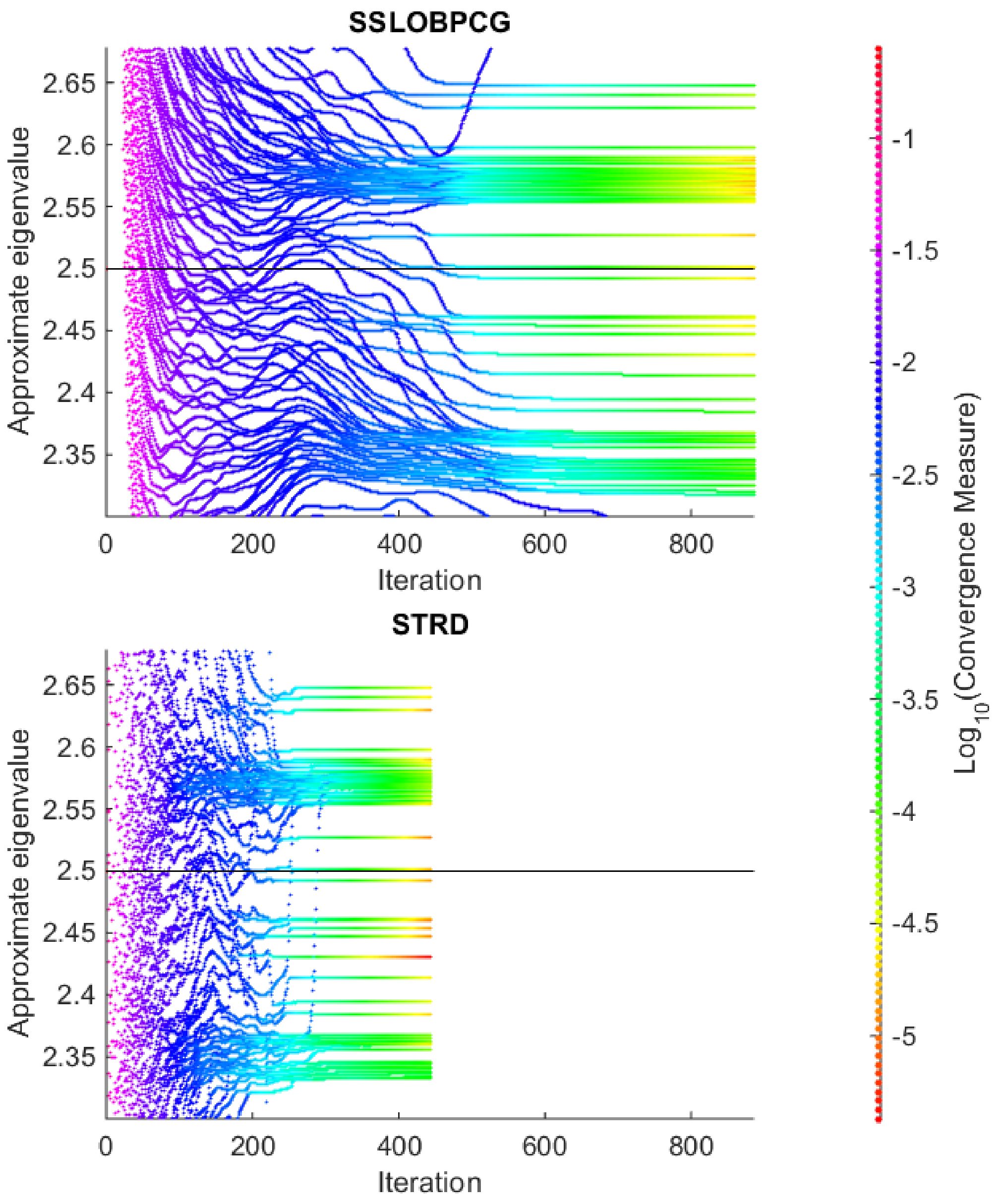


Figure 2.1: Si5H12 $\sigma = 2.5, \quad \text{and} \quad \tau_c = 10^{-4}$

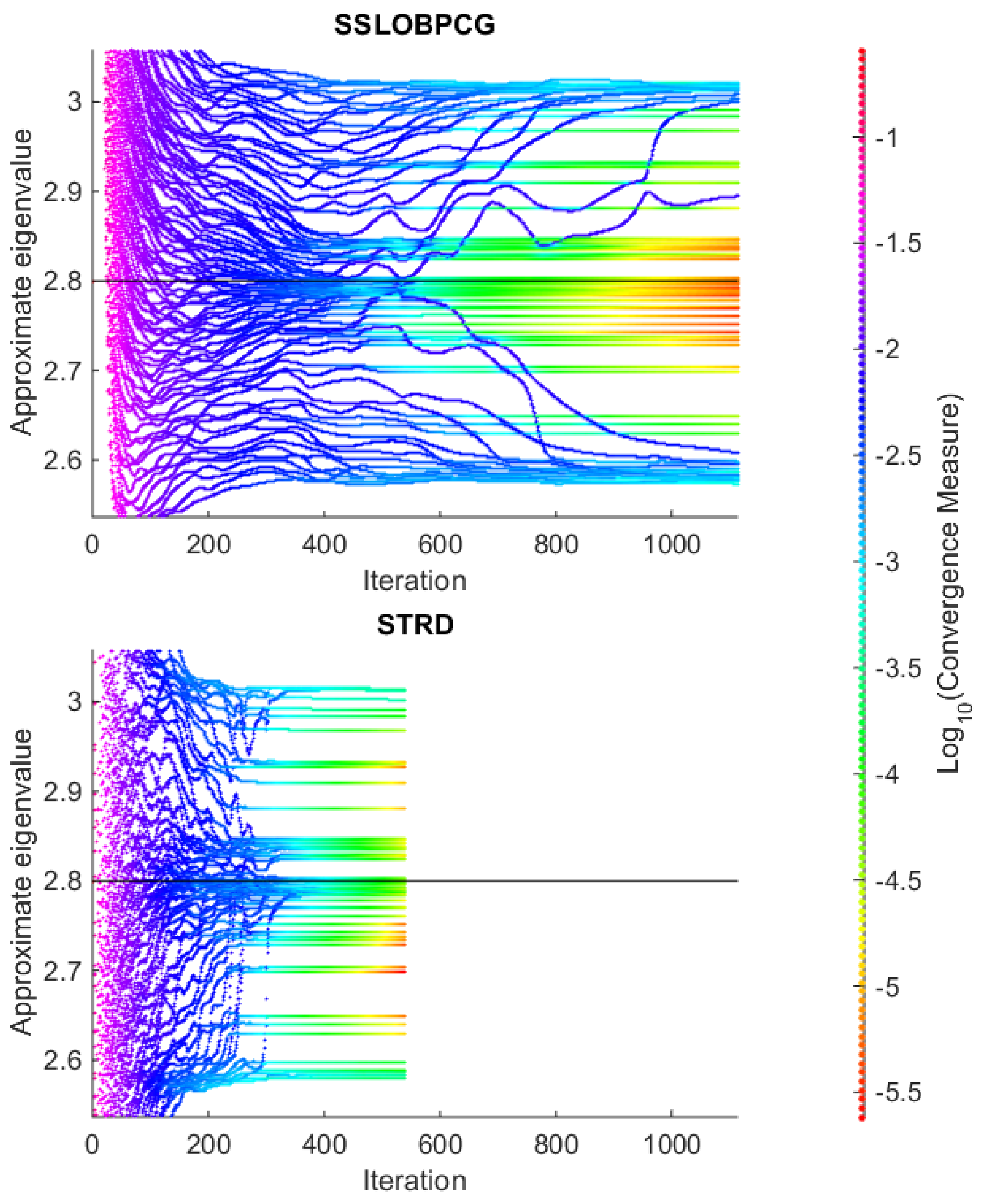


Figure 2.2: Si5H12 $\sigma = 2.8$, and $\tau_c = 10^{-4}$

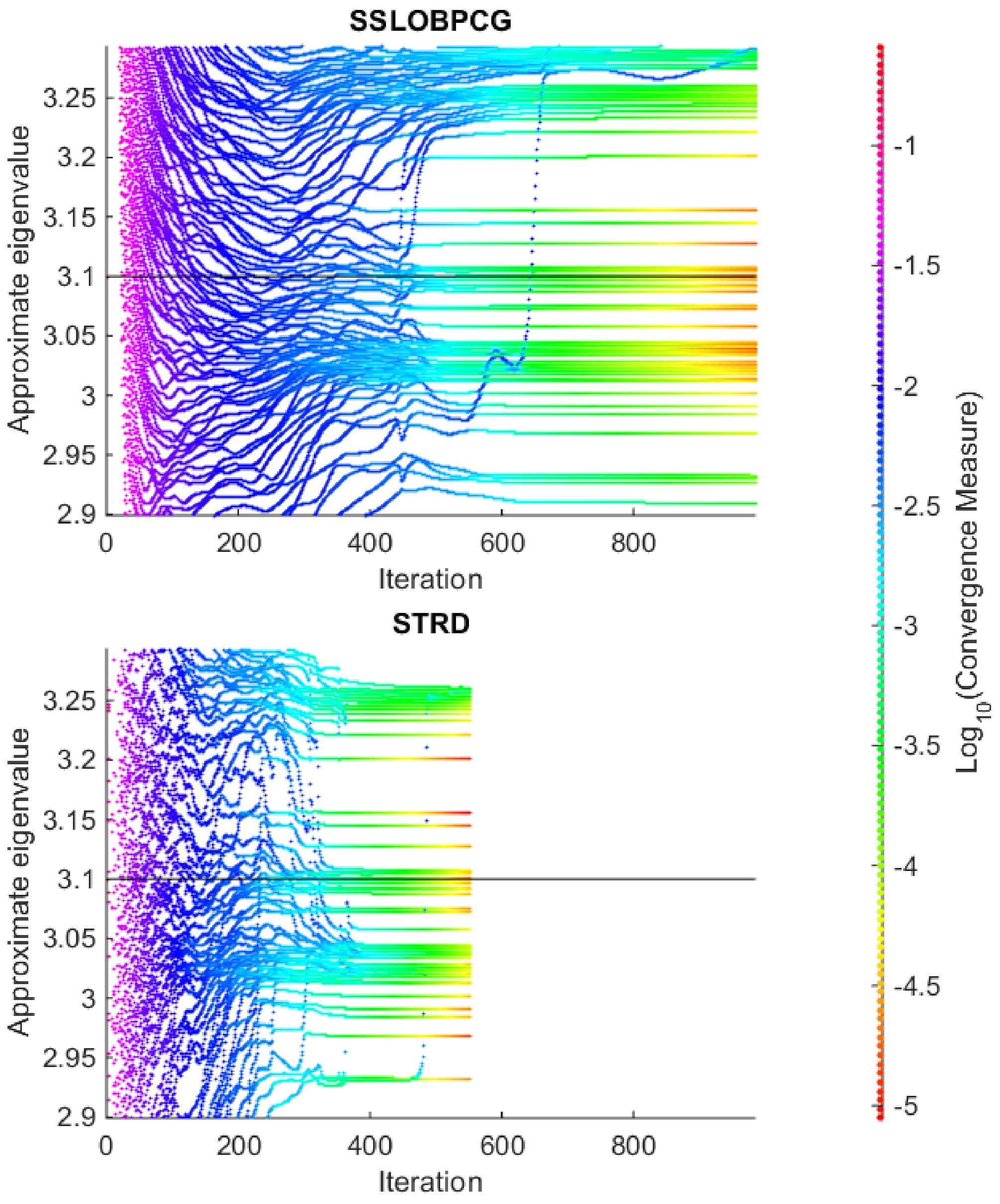


Figure 2.3: Si5H12 $\sigma = 3.1$, and $\tau_c = 10^{-4}$

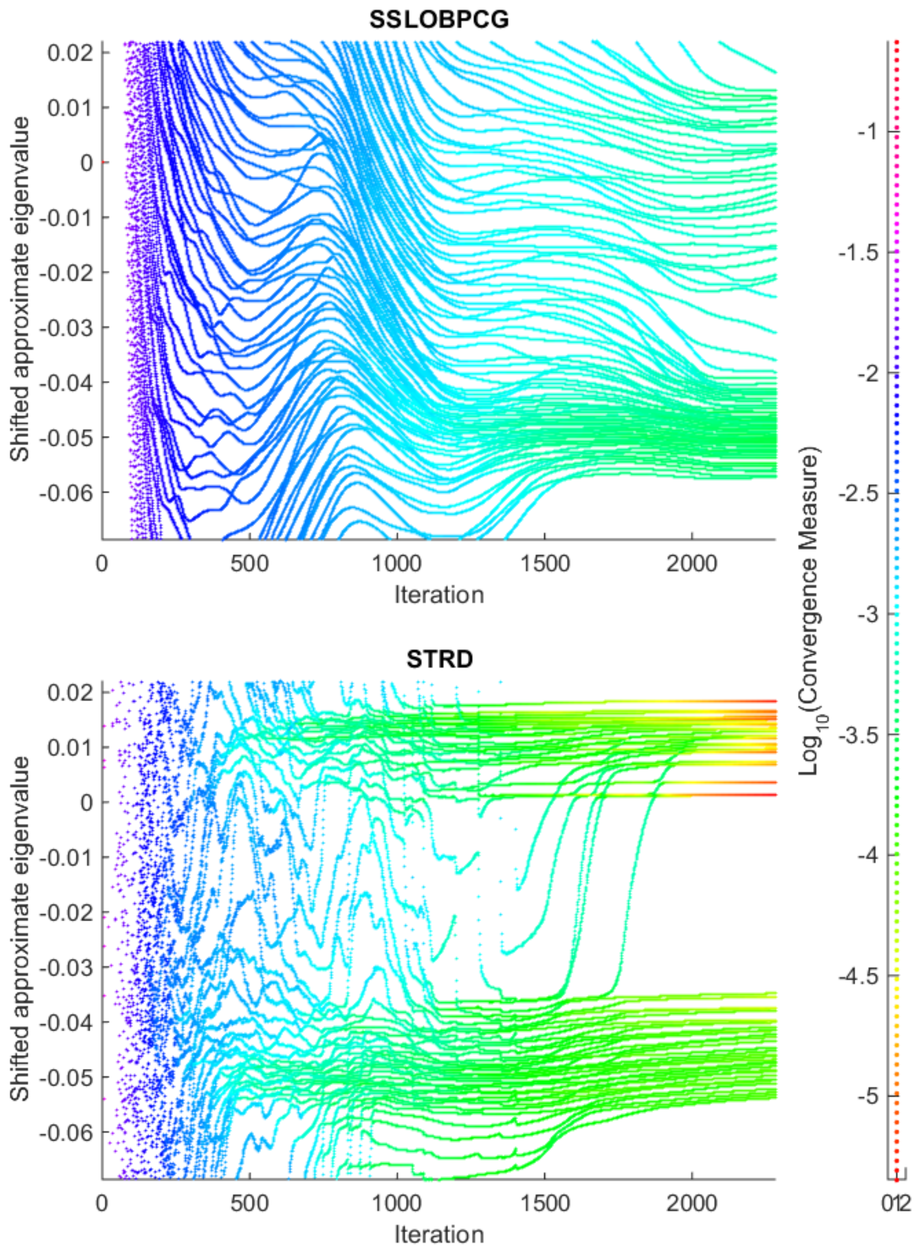


Figure 2.4: CO $\sigma = 3.0$, and $\tau_c = 10^{-4}$

Chapter 3

True BLAS-3 Performance QRCP using Random Sampling

3.1 Contributions and results

For large matrices, QR with Column Pivoting requires an order of magnitude longer to process than blocked QR without column pivoting. This is due to a communication bottleneck in processing pivoting decisions that cannot be avoided using previous approaches, although this bottleneck can be significantly reduced in parallel implementations [15]. By using randomized sampling to process pivoting decisions with a much smaller matrix, the leading-order term of communication complexity can be eliminated entirely.

3.1.1 Random sampling and update formulas

This work explores variations of QRCP using randomized sampling to process blocks of pivots. Magnitudes of trailing column norms are detected using Gaussian random compression matrices to produce smaller sample matrices. We analyze the probability distributions of sample column norms to justify the internal updating computation used to select blocks of column pivots.

This work also proposes two sample update computations that reduce BLAS-3 communication required to process full matrices by one third of what would be required by sample-based QRCP using a new sample compression for every block of column pivots.

3.1.2 Low-rank approximations

We extend this method of factorization to produce truncated low-rank approximations. We propose an implementation using random-sample pivots that avoids the trailing update computation on the full matrix. This further reduces BLAS-3 communication to nearly half of the truncated version employing a trailing matrix update.

The Truncated Randomized QR with Column Pivoting algorithm immediately extends to approximate the truncated Singular Value Decomposition using a variation on Stewart's QLP algorithm [66].

3.1.3 Results

We are able to achieve matrix factorizations of similar quality to standard QRCP while retaining communication complexity of unpivoted QR. Algorithms have been implemented and tested in Fortran 90 with OpenMP on shared-memory 24-core systems. Our performance experiments compare these algorithms against LAPACK subroutines complied using the Intel Math Kernel Library and verify computation time nearly as short as unpivoted QR (dgeqrf) and substantially shorter than QRCP (dgeqp3).

We also examine performance and quality of low-rank truncated approximations. The truncated algorithms proposed operate in the same time domain as an implementation of truncated QR, which provides an experimental lower bound on processing time. Truncated Randomized QRCP gives nearly the same approximation error as Truncated QRCP. Similarly, the approximate truncated singular value decomposition proposed (TUXV) yields error nearly as small as truncated SVD.

These algorithms have the potential to dramatically reduce factorization times in a wide variety of applications in numerical linear algebra. By eliminating the leading-order communication complexity term in QRCP, problems that have been too large to process with QRCP-dependent subroutines will now become feasible. This has the potential to increase the usefulness of computational modeling in a wide variety of fields of research in science and engineering.

3.2 Introduction to QRCP

The QR decomposition is one of the most well known and useful tools in numerical linear algebra. An input matrix A is expressed as the product of an orthogonal matrix Q and a right-triangular factor R,

$$A = QR.$$

QR is particularly stable in that the decomposition always exists regardless of the conditioning of the input matrix. Furthermore, it has finely tuned implementations that operate at the BLAS-3 level of performance. Unfortunately, the standard QR algorithm is not suitable for purposes requiring rank detection or low-rank approximations. These objectives require a column permutation scheme to process more representative columns earlier in the decomposition [8].

QR with Column Pivoting (QRCP) is a standard solution that is usually adequate for such purposes with a few rare exceptions such as the Kahan matrix [25]. A permutation matrix P is introduced to rearrange columns into a more beneficial ordering which is then decomposed as before,

$$AP = QR.$$

QRCP is an effective alternative to the much more costly Singular Value Decomposition (SVD) and it has a number of applications including least-squares approximations which are discussed in detail by Chan *et al.* [9]. Furthermore, some applications for which unpivoted QR is usually sufficient occasionally encounter ill-conditioned matrices. If QRCP can be implemented at a level of performance similar to QR, stability safeguards can be included to safely handle problematic cases without burdening performance.

3.2.1 QRCP performance

Early BLAS-2 implementations of QR and QRCP gave very similar performance results until reflector blocking was employed in QR [6]. For large matrices, blocking improves performance by reducing memory passes over matrix data. Instead, Householder reflectors are constructed in groups using the compact WY formulation. These groups can be applied to remaining columns in a single pass over matrix memory rather than repeating communication for each column processed. This reduces the leading order of communication complexity by a factor of the block size.

The primary obstacle to high-performance QRCP is the additional communication that is required to make column pivoting decisions. In order to understand how our algorithms improve performance, we first review the reasons why additional communication could not be avoided with previous approaches. An outline of BLAS-2 QRCP is provided in Algorithm 15.

QRCP can be understood as a greedy procedure intended to maximize the magnitude of the determinant of the upper left block at every iteration. At the end of iteration j we can represent the matrix A as a partial factorization using the pivots

Algorithm 15 Householder QR with Column Pivoting (QRCP).

```
Input:
    $A$ is $m \times n$.
Output:
    $Q$ is $m \times m$ orthogonal matrix.
    $R$ is $m \times n$ right triangular matrix with diagonals in order of descending magnitude.
    $P$ is $n \times n$ permutation matrix such that $AP = QR$.
 1: function $[Q, R, P]$=qrcp($A$)
 2:     Compute initial column 2-norms which will become trailing column norms.
 3:     do $j = 1, 2, \dots \min(m, n)$
 4:         Swap column $j$ with a subsequent column having max trailing norm.
 5:         Update composite permutation with last swap $S_j$, $P^{(j)} = P^{(j-1)}S_j$.
 6:         Form Householder reflection $H_j$ from the new column $j$.
 7:         Apply reflection $A^{(j)} = H_j A^{(j-1)} S_j$.
 8:         Update trailing column norms by removing the contribution of row $j$.
 9:     end do
10:     $Q = H_1 H_2 \dots H_{\min(m,n)}$ is the product of all reflections.
11:     $R = A^{(\min(m,n))}$
12: end function
```

$P^{(j)}$ and transformations $Q^{(j)} = H_1 \cdots H_j$ as

$$AP^{(j)} = Q^{(j)} \begin{bmatrix} R_{11}^{(j)} & R_{12}^{(j)} \\ 0 & \hat{A}^{(j)} \end{bmatrix}.$$

On the next iteration the trailing column norm selected will become the magnitude of the next diagonal element in $R_{1,1}^{(j+1)}$. This gives the new determinant magnitude

$$|\det R_{11}^{(j+1)}| = |\det R_{11}^{(j)}| \|\hat{A}^{(j)}(:, p_{j+1})\|_2.$$

Provided prior pivot decisions are considered immutable, this scheme maximizes the determinant magnitude at every iteration. Note that true determinant maximization would require exchanging prior columns and adjusting the factorization accordingly [28].

In order to produce a correct pivot decision on iteration $(j + 1)$, trailing column norms must be updated to remove the contribution of row j, labeled ρ_j below, which depends on the transformation $H_j = I - y_j \tau_j y_j^T$. The full update requires two BLAS-2 operations on the whole trailing matrix on every iteration. The first

BLAS-2 operation computes the scaled inner products $w_j^T = \tau_j y_j^T A^{(j-1)}$. The second operation is the outer product $y_j w_j^T$ subtracted from the trailing matrix:

$$\begin{bmatrix} r_{jj} & \rho_j \\ 0 & A^{(j)} \end{bmatrix} = (I - y_j \tau_j y_j^T) A^{(j-1)}$$

These BLAS-2 operations are the performance bottleneck of QRCP.

3.2.2 Attempts to achieve BLAS-3 QRCP

Quintana-Orti *et al.* were able to halve these BLAS-2 operations with the insight that the trailing norm update only requires the scaled inner products on each iteration [57]. That is, only ρ_j is needed at the end of iteration j. Likewise, once the next swap is selected, prior transformations are applied only to that column which then forms the basis for the next reflector. Thus reflectors can be gathered and applied in blocks similar to BLAS-3 QR.

Unfortunately, the remaining 50% of BLAS-2 operations still dominates communication complexity for large matrices. Including only the leading term in reflector inner product computations gives complexity of BLAS-2 operations $\frac{1}{2}\min(m,n)mn$ to complete the factorization. This is why even the most efficient implementations of this style of reflection blocking still run substantially slower than unpivoted QR on both sequential and parallel architectures.

3.2.3 Communication Avoiding Rank-Revealing QR

Several mechanisms have been put forward to avoid repeating full BLAS-2 operations over the trailing matrix for every iteration. Bischof proposed pivoting restricted to local blocks [7] and Demmel *et al.* propose a strong parallel process called Communication Avoiding Rank-Revealing QR (CARRQR) [15, 16]. CARRQR proceeds by partitioning the trailing matrix into column sets that are processed independently. Groups of b candidate pivots are selected by using QRCP within each partition of columns. Groups are combined into $2b$ candidates which are refined again by repeating QRCP to return b candidates. This refinement process repeats until only one group of b pivots remains which is then processed and applied to the trailing matrix with BLAS-3 operations.

Both of these mechanisms perform at least as many BLAS-2 operations over each column, however performance is improved by restricting these BLAS-2 operations to local blocks of memory. With enough parallel processors, these blocks of memory can be small enough to fit in cache and thus be regarded as BLAS-3 operations.

3.2.4 Randomized sampling

We are able to entirely eliminate these BLAS-2 passes over the matrix by using random sampling to make pivot decisions. As a result, we are able to achieve performance in the class of BLAS-3 QR. We sacrifice knowing exact trailing norms when pivot decisions are made and settle for approximations that have extremely high probability of detecting magnitudes needed to reveal rank or construct reliable low-rank approximations.

We emphasize that the pivots obtained from QRCP are not intrinsically superior to those we obtain with randomized sampling for the purpose of low-rank approximation. On the contrary, our experiments show that the Frobenius norm of truncation error is often lower using our approach.

3.3 Sample QRCP

Randomized sampling is a computational tool that has recently gained traction in a number of applications in numerical linear algebra [48, 68, 50, 45]. Random sampling reduces communication complexity via dimensional reduction while simultaneously maintaining high probability of safe error bounds on the approximations that follow. This is the result of the well-known Johnson-Lindenstrauss Lemma [39].

Let $0 < \tau < 1/2$ be a relative error threshold and $a_j \in \mathbb{R}^m$ represent columns of A for $j = 1, \dots n$. Using a randomized compression matrix $\Omega \in \mathbb{R}^{\ell \times m}$ with unit-variance Gaussian Independent Identially Distributed (GIID) elements, the probability of successfully detecting column norms is bound by

$$\texttt{Pr}\left(\left|\frac{\|\Omega^{(\ell)} a_j\|_2^2}{\ell \|a_j\|_2^2} - 1\right| \leq \tau\right) \geq 1 - 2e^{\frac{-\ell\tau^2}{4}(1-\tau)}.$$

We use the sample matrix $B = \Omega A$ to select the column with largest approximate norm. Subsequent columns are selected by continuing QRCP on the sample as seen in the single-sample version, Algorithm 16. We show in Section 3.3.1 that the trailing columns computed during sample QRCP correspond to randomized compressions of corresponding trailing columns in the full matrix factorization. Thus sample column norms remain suitable for continued column selection.

The leading order BLAS-2 complexity required to process a block of pivots becomes $k\ell n$ versus kmn in Quintana-Orti's version of QRCP. However, if the $\ell \times n$ sample matrix dimension is calibrated to the cache size of the machine, the sample communication is effectively BLAS-3. Algorithm 16 was first implemented as class project at UC Berkeley in April 2014 and it is the simplest example of this approach [18]. P.G. Martinsson independently developed similar work [49].

This formulation is acceptable for very-low-rank approximations in which the required sample size is small enough to maintain communication efficiency. For larger approximations we will resort to a more comprehensive algorithm including a sample update formulation that subsumes this version. However the single-sample algorithm illuminates the performance advantage gained from this approach, so we examine it first.

Algorithm 16 Single-sample randomized QRCP.

```
Input:
    $A$ is $m \times n$.
    $k$ the desired approximation rank. $k \ll \min(m, n)$.
Output:
    $Q$ is $m \times k$ orthogonal column matrix in the form of $k$ reflectors.
    $R$ is $k \times n$ truncated upper trapezoidal matrix.
    $P$ is $n \times n$ permutation matrix such that $AP \approx QR$.
1: function $[Q, R, P]$=ssrqrcp$(A, k)$
2:     Compute required number of sample rows $l = k + p$ to obtain acceptable sample error.
3:     Generate random $l \times m$ GIID compression matrix $\Omega$.
4:     Form the sample $B = \Omega A$.
5:     Get $k$ column pivots from sample, $[Q_b, R_b, P] = \texttt{qrcp}(B)$.
6:     Apply permutation $A^{(1)} = A^{(0)}P$.
7:     Construct $k$ reflectors from new leading columns, $[Q, R_{1,1}] = qr(A^{(1)}(:, 1 : k))$.
8:     Finish $k$ rows of $R$ in remaining columns, $R_{1,2} = Q(:, 1 : k)^T \hat{A}(:, k + 1 : n)$.
9: end function
```

Although a GIID matrix Ω is more computationally expensive than some alternatives, it only contributes a small fraction to the total time. For example, randomization comprises 7% of processing time on a 12000 by 12000 matrix with $k = 32$ and $\ell = 40$ on a 24-core test machine. Furthermore, this fractional contribution becomes even smaller in algorithms that follow. Therefore we believe optimization of Ω is premature at this stage. It is also a robust choice because it is both dense and invariant in distribution under independent orthogonal transformations. These characteristics support our analysis.

Permutation ordering is determined all at once by applying QRCP to the much smaller sample matrix B instead of A. CARRQR could be applied to B to further improve performance, however this has not been necessary up to this point since our experiments show it is not a performance bottleneck.

After the permutation is known, A is rearranged once and the leading columns processed contiguously without referencing the rest of the matrix just as is done in BLAS-3 QR. Resulting reflectors are then applied to the trailing matrix with BLAS-3 matrix multiplies.

3.3.1 Sample norm distribution updates during QRCP

Initially sample columns are proportional to columns of A with a constant of proportionality following the Chi-squared distribution. To see this we examine a sample column $b = \Omega a$. Because GIID sample matrices are invariant in distribution under independent orthogonal transformations, we represent Ω and a as

$$\Omega = \begin{bmatrix} w_1 & W_2 \end{bmatrix} \begin{bmatrix} q_a^T \\ Q_{\perp a}^T \end{bmatrix} \quad \text{and} \quad a = q_a \|a\|_2 \quad \text{giving} \quad b = w_1 \|a\|_2$$

where q_a is the unit vector formed from a and $Q_{\perp a}$ is an $m \times m-1$ complement giving a full orthogonal basis. The components of w_1 remain GIID giving a 2-norm-squared following the Chi-squared distribution with ℓ degrees of freedom. That is,

$$\|b\|_2^2 = x\|a\|_2^2 \quad \text{where} \quad \rho_\ell(x) = \frac{\left(\frac{x}{2}\right)^{\frac{\ell}{2}} e^{-\frac{x}{2}}}{x\Gamma(\frac{\ell}{2})} \quad \text{giving} \quad \mathbb{E}(x) = \ell \quad \text{and} \quad \mathbb{V}(x) = 2\ell.$$

As a result, we can expect the initial sample to approximately resolve column norms with a relative error that scales as $\frac{1}{\sqrt{\ell}}$.

Because the QR factorization always exists, at iteration j we can represent A and B as partial factorizations

$$AP^{(j)} = Q \begin{bmatrix} R_{11}^{(j)} & R_{12}^{(j)} \\ 0 & R_{22}^{(j)} \end{bmatrix} \quad \text{and} \quad BP^{(j)} = Q_b \begin{bmatrix} S_{11}^{(j)} & S_{12}^{(j)} \\ 0 & S_{22}^{(j)} \end{bmatrix} \quad \text{where}$$

$R_{11}^{(j)}$ and $S_{11}^{(j)}$ are both upper triangular. This allows us to represent Ω as a partition of components W of the orthogonal bases Q_b and Q.

$$\Omega = Q_b \begin{bmatrix} W_{11}^{(j)} & W_{12}^{(j)} \\ W_{21}^{(j)} & W_{22}^{(j)} \end{bmatrix} Q^T \quad \text{which gives}$$

$$\begin{bmatrix} S_{11}^{(j)} & S_{12}^{(j)} \\ 0 & S_{22}^{(j)} \end{bmatrix} = \begin{bmatrix} W_{11}^{(j)} R_{11}^{(j)} & W_{11}^{(j)} R_{12}^{(j)} + W_{12}^{(j)} R_{22}^{(j)} \\ W_{21}^{(j)} R_{11}^{(j)} & W_{21}^{(j)} R_{12}^{(j)} + W_{22}^{(j)} R_{22}^{(j)} \end{bmatrix}.$$

As long as j linearly independent columns of A exist, Johnson-Lindenstrauss provides virtual certainty that j linearly independent columns of B also exist. Then $S_{11}^{(j)}$ is non-singular upper-triangular. It immediately follows that both $W_{11}^{(j)}$ and $R_{11}^{(j)}$ are also non-singular upper-triangular which also implies $W_{21}^{(j)} = 0$. In other words, we have implicitly formed a QR factorization of ΩQ using the same orthogonal matrix Q_b that corresponds to BP. As such, the trailing column norms of B in the remaining lower right partition are $S_{22}^{(j)} = W_{22}^{(j)} R_{22}^{(j)}$, which is a sample of the trailing matrix in the corresponding partial factorization of AP.

3.3.2 Sample bias

If the sample had not been used to make pivot decisions then $W_{22}^{(j)}$ would be equivalent in distribution to a new GIID matrix. That is because Q would be formed from columns of A independent of Ω. Similarly, Q_b would be formed from the leading columns of ΩQ, which would be independent of remaining columns. As such, the trailing column norms in B would remain unbiased predictors for the trailing column norms in A. Unfortunately, using the sample to make pivot decisions, which is indeed the entire purpose of random sampling, results in two forms of sampling bias.

The first is selection bias. This is similar to publication bias of experimental results. Statistical anomalies are more likely to be noticed, reported, and then published than less interesting but more common outcomes. We use the sample to estimate true norms in order to permute the largest column to the front. The selected sample column will be more likely to exhibit an unusually large norm—larger than we would expect from an unbiased sample—due to the fact that it was specifically selected as the maximum. This bias would be present even if a new compression matrix were used for every column selection.

This bias is most pronounced when many columns are nearly tied for having the largest norm. In such situations, sample noise out-weighs the true distinctions between columns. As a result, the 2-norm of the selected sample column will over-represent the true 2-norm and the ordinary Chi-squared expectation value will fail to hold.

Fortunately, when several columns are nearly tied for having the largest 2-norm, the particular choice is not critical to the quality of the final factorization. Conversely, if one column has a substantially greater norm than all other columns, statistical anomalies are less likely to dictate the outcome of the selection. In other words, selection bias is unlikely to obscure important distinctions.

The following numerical experiment computes the expected 2-norm of a column selected by this method. The matrix A is constructed to have orthogonal columns with scaled 2-norms according to a decaying exponential so that column j has 2-norm

$$\|a_j\|_2 = \phi^{j-1} \quad \text{where} \quad \phi \in [0,1].$$

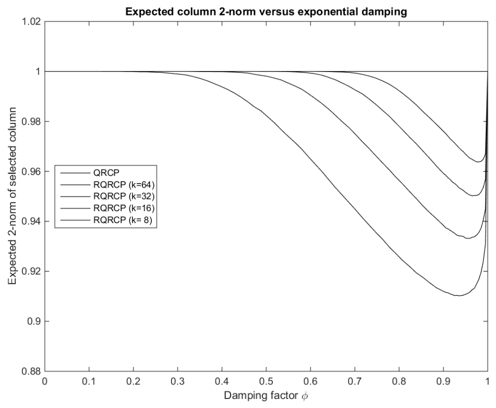


Figure 3.1: Expectation values of selected columns using sample QRCP.

When the damping factor ϕ is close to 1 we see that sub-optimal choices are more likely, but they do little damage. When column norms are damped more dramatically damaging choices are highly suppressed. As expected, higher rank samples do a better job of detecting optimal choices.

3.3.3 Norm distribution truncation

The second form of bias arises after partial factorization. Because prior column selections depended on the sample, the remaining low-right partition of the sample factorization is no longer equivalent in distribution to a GIID compression. Suppose we have a partial factorization of the sample matrix after j rows and columns are complete:

$$Q_b^{(j)T} B P^{(j)} = \begin{bmatrix} s_{1,1} & \cdots & s_{1,j} & s_{1,j+1} & \cdots & s_{1,n} \\ 0 & \ddots & \vdots & & & \vdots \\ 0 & & s_{j,j} & s_{j,j+1} & \cdots & s_{j,n} \\ 0 & \cdots & 0 & \hat{b}_{j+1} & \cdots & \hat{b}_n \end{bmatrix}.$$

QRCP produces right-triangular matrices with the descending norm property:

$$s_{i,j'}^2 + s_{i+1,j'}^2 + \cdots s_{j,j'}^2 + \|\hat{b}_{j'}\|_2^2 \le s_{i,i}^2 \quad \text{for all} \quad i = 1, \cdots, j \quad \text{and } j' = j+1, \cdots, n.$$

Again we can write the sample norm as a factor of the true norm $\|\hat{b}_{j'}\|_2^2 = x_{j'}\|\hat{a}_{j'}\|_2^2$ which gives the equivalent expression

$$x_{j'} \leq \tau_{j'} \quad \text{where} \quad \tau_{j'} = \frac{1}{\|\hat{a}_{j'}\|_2^2} \min_{i=1}^{j} \left(s_{i,i}^2 - (s_{i,j'}^2 + s_{i+1,j'}^2 + \cdots s_{j,j'}^2) \right).$$

Theorem: Given a partial factorization as above, the trailing 2-norm-squared for column j' of the sample corresponds to a truncated Chi-squared distribution with $\ell - j$ degrees of freedom. The normalization factor $\gamma()$ is the lower-incomplete gamma function corresponding to the cutoff threshold and the remaining degrees of freedom.

$$\rho_{\ell-j,\tau_{j'}}(x_{j'}) = \begin{cases} \frac{\left(\frac{x_{j'}}{2}\right)^{\frac{\ell-j}{2}} e^{-\frac{x_{j'}}{2}}}{x_{j'}\gamma(\frac{\ell-j}{2},\frac{\tau}{2})} & x_{j'} \leq \tau_{j'} \\ 0 & x_{j'} > \tau_{j'} \end{cases}.$$

Proof: Given a particular permutation P^*, the conditional probability density function for $x_{j'}$ satisfies Bayes' theorem in the form

$$\rho(x_{j'}|P^*)\mathtt{Pr}(P^*) = \mathtt{Pr}(P^*|x_{j'})\rho(x_{j'})$$

where we have a discrete probability functions in the variable P^* and continuous distributions in the variable $x_{j'}$.

If the descending norm property is not satisfied then QRCP would not have produced the permutation P^*. It follows that $\mathtt{Pr}(P^*|x_{j'}) = 0$ for $x_{j'} > \tau_{j'}$. Conversely, as long as $x_{j'} < \tau_{j'}$ the particular value of x_j does not affect any branch decisions in QRCP thus far. Within this region $\mathtt{Pr}(P^*|x_{j'})$ is a constant independent of $x_{j'}$. It follows

$$\rho(x_{j'}|P^*) \propto \begin{cases} \rho(x_{j'}) & x_{j'} < \tau_{j'} \\ 0 & x_{j'} > \tau_{j'} \end{cases}.$$

Solving for unit-normalization gives the stated result.

Two interesting effects follow from this computation. The sample column 2-norm-squared expectation value drops as the cutoff threshold drops. If prior sample columns are relatively large in comparison to remaining columns, which occurs at gaps in the spectrum, then the cutoff thresholds are also relatively large. Thus sample QRCP behaves more like an unbiased sample on remaining columns at spectral gaps.

Secondly, columns that have large magnitudes of components that are linearly dependent with previously selected columns have lower cutoffs. In other words, when multiple columns have roughly similar true 2-norms, sample QRCP is biased towards the column that originally had a larger linearly independent fraction. In contrast, standard QRCP is only concerned with the magnitude of the remaining linearly independent component, regardless of the magnitude of prior dependence.

3.3.4 Proposals to further reduce bias

Bias effects could be reduced by increasing the sample rank and by over-pivoting additional columns beyond those needed for the desired approximation rank. These extra columns could be permuted to the front of the matrix and standard QRCP would be locally applied to extract the optimal subset. Similarly, one could apply determinant maximizing column swaps within the set of over-pivoted columns [28].

3.4 Sample updates

The original sample matrix $B = \Omega A$ is constructed with rank $\ell = k + p$ where p is additional padding required to ensure the sample error remains below an acceptable threshold. Accordingly, we can only safely select k pivots from the initial sample. If a decomposition of rank greater than k is required then we need a new sample of the trailing columns of A to continue the factorization. Martinsson's approach continues by multiplying trailing columns by a new compression matrix. We propose a sample update formulation that does not require accessing trailing columns of A. Instead we use a less expensive transformation on the previous sample to obtain a new rank ℓ compression. As a result, the proposed update reduces BLAS-3 communication in the overall factorization by at least one third of Martinsson's approach. The remaining BLAS-3 communication corresponds to blocked reflector inner products and trailing updates.

3.4.1 First update formula

The original update formula we derive is an extension of the implicit update mechanism described in the prior section. After performing QRCP on the sample matrix B, the array is left in the transformed state (3.1). Note that Algorithms 17 and 18 that follow will proceed in blocks of pivots. The bracket superscripts denote results of a computation that occurred on the indicated block iteration. For example, the first block iteration uses the sample $B^{[1]} = \Omega^{[1]} A^{[0]}$ where $A^{[0]}$ represents the matrix before any block iterations have been performed.

At the end of block iteration J the sample has been transformed into the partial upper-trapezoidal matrix:

$$\begin{gathered} S^{[J]} = \left(Q_b^{[J]T} \Omega^{[J]} Q^{[J]}\right) \left(Q^{[J]T} A^{[J-1]} P^{[J]}\right) \\ \begin{bmatrix} S_{11}^{[J]} & S_{12}^{[J]} \\ 0 & S_{22}^{[J]} \end{bmatrix} = \begin{bmatrix} W_{11}^{[J]} & W_{12}^{[J]} \\ 0 & W_{22}^{[J]} \end{bmatrix} \begin{bmatrix} R_{11}^{[J]} & R_{12}^{[J]} \\ 0 & A^{[J]} \end{bmatrix}. \end{gathered} \tag{3.1}$$

We require a rank ℓ compression of the trailing matrix $A^{[J]}$ to continue the next iteration. This can be implicitly constructed from $W^{[J]}$. Take the new compression matrix $\Omega^{[J+1]}$ to be:

$$\Omega^{[J+1]} = \begin{bmatrix} W_{12}^{[J]} \\ W_{22}^{[J]} \end{bmatrix} \quad \text{and} \quad B^{[J+1]} = \Omega^{[J+1]} A^{[J]} = \begin{bmatrix} S_{12}^{[J]} - W_{11}^{[J]} R_{12}^{[J]} \\ S_{22}^{[J]} \end{bmatrix}.$$

We only need to remove the contribution of $W_{11}^{[J]} R_{12}^{[J]}$ from result of the last iteration. Both $R_{11}^{[J]}$ and $R_{12}^{[J]}$ will be computed in blocked BLAS-3 operations using the previous k pivots of A. Since $W_{11}^{[J]}$ can be recovered from $S_{11}^{[J]}$, we can avoid any direct computations on Ω. This gives the sample update formula

$$\begin{bmatrix} B_1^{[J+1]} \\ B_2^{[J+1]} \end{bmatrix} = \begin{bmatrix} S_{12}^{[J]} - S_{11}^{[J]} R_{11}^{[J]-1} R_{12}^{[J]} \\ S_{22}^{[J]} \end{bmatrix}. \tag{3.2}$$

3.4.2 Alternative update

A second approach avoids absorbing Q into the compression matrix to form W. Instead we can formulate an update that simply removes the contribution of the leading k columns from the previous compression matrix. Note that we still absorb the orthogonal transformation Q_b^T—which does not alter column norms—into the subsequent compression matrix. The leading k columns to be removed are labeled $\Psi^{[J+1]}$ and remaining columns give the next compression matrix $\Omega^{[J+1]}$. We desire $B^{[J+1]} = \Omega^{[J+1]} A^{[J]}$ where

$$\begin{bmatrix} \Psi^{[J+1]} & \Omega^{[J+1]} \end{bmatrix} = Q_b^{[J]T} \begin{bmatrix} \Omega_1^{[J]} & \Omega_2^{[J]} \end{bmatrix} \quad \text{and} \quad A^{[J-1]} P^{[J]} = Q^{[J]} \begin{bmatrix} R_{11}^{[J]} & R_{12}^{[J]} \\ 0 & A^{[J]} \end{bmatrix}.$$

The orthogonal transformation is implemented using a block reflection in the compact WY formulation $Q^{[J]} = (I - Y^{[J]} T^{[J]} Y^{[J]T})$ which allows us to express the new sample as

$$\begin{aligned} B^{[J+1]} &= \begin{bmatrix} \Psi^{[J+1]} & \Omega^{[J+1]} \end{bmatrix} \begin{bmatrix} R_{12}^{[J]} \\ A^{[J]} \end{bmatrix} - \Psi^{[J+1]} R_{12}^{[J]} \\ &= Q_b^{[J]T} \Omega^{[J]} (I - Y^{[J]} T^{[J]T} Y^{[J]T}) A^{[J-1]} P_2^{[J]} - \Psi^{[J+1]} R_{12}^{[J]}. \end{aligned}$$

We compute and store the matrix of reflector inner products on all remaining columns $W^{[J]T} = T^{[J]T} Y^{[J]T} A^{[J-1]} P_2^{[J]}$ in the course of computing $R_{12}^{[J]}$. This gives the alternative update formula:

$$B^{[J+1]} = S_2^{[J]} - \left(Q_b^{[J]T} \Omega^{[J]} Y^{[J]} \right) W^{[J]T} - \Psi^{[J+1]} R_{12}^{[J]}. \tag{3.3}$$

Note that this update formula requires retaining Ω and applying the orthogonal transformations from sample QRCP in order to compute the second and third terms.

3.4.3 Subtle bias

The original update formula absorbs Q—which depends on the columns selected—into subsequent compression matrices. This clearly creates an interdependence between previously selected columns and subsequent samples and may be a cause for concern regarding sample bias. In contrast, the alternative update formula uses a sequence of compression matrices that exist—modulo benign left-hand orthogonal transformations—before any columns of A are known or selected. One may be tempted to conclude that the alternative update formula must produce independent random samples of the trailing matrix at each iteration. Unfortunately, that is not true. The fact remains that pivot outcomes subtly constrain subsequent samples.

A simple way to understand this is to consider the possibility of observing extremely large sample norms at intermediate stages. If the compression matrix were newly generated there would be nonzero probability of observing an arbitrarily large sample norm for any particular remaining column, but that would be inconsistent with prior pivot selections. If a component of Ω producing an extremely large observation had been present in prior stages it would have altered a prior pivot decision. That means remaining sample column norms must be bound by previous pivoting decisions in a similar fashion to the distribution truncation discussed in Section 3.3.3. However, we do not undertake such a computation at this time. Our numerical experiments have shown both update formulas to be reliable.

3.5 Full randomized QRCP

Full Randomized QR with Column Pivoting (RQRCP) can be structured as a modification to BLAS-3 QR. The algorithm must simply interleave processing blocks of reflectors with permutations obtained from sample QRCP. This is described in detail in Algorithm 17 which uses the second update formulation (3.3).

When QRCP is applied to the sample matrix B, only a partial decomposition is necessary. This second argument b in the subroutine call $\texttt{qrcp}(B^{[J]}, b)$ indicates that only b column permutations are required. Although the additional cost of processing all ℓ columns is very small, halting the computation early is a trivial modification.

After sample pivots have been applied to the array containing both A and R, we perform QR factorization on the new leading b columns of the trailing matrix $\hat{A}_1^{[J]}$. The result is expressed in compact WY notation. Reflectors are then applied to the trailing matrix and used to form the sample update $B^{[J+1]}$ in preparation for the next iteration.

3.5.1 Truncated RQRCP with trailing update

Algorithm 17 can be trivially modified to produce a low-rank truncated approximation of the original matrix. Simply include approximation rank k as a user-defined argument instead of setting $k = \min(m, n)$. The rest of the algorithm remains identical. This modification is acceptable if, after processing k columns, the user requires the trailing matrix $A^{(k)}$ in the partial factorization

$$AP = Q \begin{bmatrix} R_{11} & R_{12} \\ 0 & A^{(k)} \end{bmatrix}.$$

However, if the trailing matrix $A^{(k)}$ is not needed, the simple modification suggested does not take advantage of a performance improvement that potentially halves factorization time.

3.5.2 Truncated RQRCP without trailing update

The trailing matrix is usually not needed for low-rank approximations and the algorithm can be reformulated to run roughly twice as fast on large matrices, provided $k \ll \min(m, n)$. This is accomplished by avoiding the trailing update which reduces BLAS-3 passes over the trailing matrix by half.

The technique is analogous to the method Quintana-Ort et al. used to halve BLAS-2 operations in QRCP. In their version of QRCP, all reflector inner products are computed, but rows and columns are only updated as needed. In order to compute correct reflector inner products without having updated the trailing matrix, we review block reflector composition. Block reflections are combined as

$$(I - Y_1T_1Y_1^T)(I - Y_2T_2Y_2^T) = I - YTY^T \quad \text{where}$$

$$Y = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix} \quad \text{and} \quad T = \begin{bmatrix} T_1 & -T_1Y_1^TY_2T_2 \\ 0 & T_2 \end{bmatrix}.$$

Corresponding reflector inner products $W^T = T^TY^TA$ are partitioned

$$W^T = \begin{bmatrix} W_1^T \\ W_2^T \end{bmatrix} \quad \text{with} \quad W_1^T = T_1^TY_1^TA \quad \text{and} \quad W_2^T = T_2^TY_2^TA - T_2^TY_2^TY_1T_1^TY_1^TA$$

$$\text{which simplifies to} \quad W_2^T = T_2^T\left(Y_2^TA - (Y_2^TY_1)W_1^T\right).$$

Thus the trailing update that is needed in Algorithm 17 can be avoided if all reflector inner products are stored. This allows columns that are selected by sample pivots to be constructed as needed. Likewise, the resulting rows of R are constructed as needed by the sample update formula.

Since the trailing update is avoided, we also avoid overwriting A. That is because the partial factorization would leave the lower right submatrix somewhat meaningless. Therefore we store both Y and R in new arrays. As usual, Y can be stored in the strictly lower triangle, however this leaves the upper triangle available to store T which saves additional computation in the future when the orthogonal matrix is applied. This also leaves the array of reflector coefficients τ already stored on the diagonal of the array that stores Y instead of being carried in a separate array.

Truncated Randomized QRCP (TRQRCP) is outlined without the trailing update in Algorithm 18 and employs the first update formula (3.2).

Algorithm 17 Randomized QRCP

```
Input:
    $A$ is $m \times n$.
Output:
    $Y$ is $m \times \min(m, n)$ lower trapezoidal reflector matrix.
    $\tau$ $\min(m, n)$ array of reflector coefficients used to form $T$.
    $R$ is $m \times n$ upper trapezoidal matrix.
    $P$ is $n \times n$ permutation matrix.
    $AP = (I - YTY^T)R$.
1: function $[Y, \tau, R, P]$=rqrcp($A$)
2:     Determine optimal block dimension $k$ and sample padding $p$.
3:     Set sample dimension $\ell = k + p$.
4:     Set maximum factorization rank $r = \min(m, n)$.
5:     Generate $\ell \times m$ GIID matrix $\Omega^{[1]}$.
6:     Sample $B^{[1]} = \Omega^{[1]} A^{[0]}$.
7:     Set composite permutation $P^{[0]} = I$.
8:     Set completed column counter $c = 0$.
9:     do J=1,2,...
10:        $[Q_b^{[J]}, S^{[J]}, P_b^{[J]}]$ = qrcp($B^{[J]}, k$).
11:        Permute $R(1:c, c+1:n) = R(1:c, c+1:n)P_b^{[J]}$.
12:        Permute trailing matrix $\hat{A}^{[J]} = A^{[J-1]} P_b^{[J]}$.
13:        Update composite permutation $P^{[J]} = P^{[J-1]} P_b^{[J]}$.
14:        Get new reflectors $[Y^{[J]}, T^{[J]}, R_{11}^{[J]}] = qr(\hat{A}_1^{[J]})$.
15:        Store reflector coefficients $\tau^{[J]} = \mathrm{diag}(T^{[J]})$.
16:        BLAS-3 reflector inner products $W^{[J]T} = T^{[J]T} Y^{[J]T} \hat{A}_2^{[J]}$.
17:        BLAS-3 trailing update $[R_{12}^{[J]}; A^{[J]}] = \hat{A}_2^{[J]} - Y^{[J]} W^{[J]T}$.
18:        if $(c + k = r)$ then exit
19:        Update sample with original or alternative formula.
20:        Increment completed column counter $c = c + k$.
21:    end do
22: end function
```

Algorithm 18 Truncated Randomized QRCP without trailing update

```
Input:
    $A$ is $m \times n$.
    $r$ approximation rank. $r \ll \min(m,n)$.
Output:
    $Y$ is $m \times r$ reflector matrix.
    $\tau$ $r$ array of reflector coefficients used to form $T$.
    $R$ is $r \times n$ upper trapezoidal matrix.
    $P$ is $n \times n$ permutation matrix.
    $AP \approx (I - YTY^T)R$.
1: function $[Y,\tau,R,P]$=trqrcp$(A,k)$
2:     Determine optimal block dimension $k$ and sample padding $p$.
3:     Set sample dimension $\ell = k + p$.
4:     Generate $\ell \times m$ GIID matrix $\Omega^{[1]}$.
5:     Sample $B^{[1]} = \Omega^{[1]}A^{[0]}$.
6:     Set composite permutation $P^{[0]} = I$.
7:     Set completed column counter $c = 0$.
8:     do J=1,2,...
9:         $[Q_b^{[J]}, S^{[J]}, P_b^{[J]}] = \texttt{qrcp}(B^{[J]}, b)$.
10:        Apply $P_b$ to columns $c+1:n$ of $A$, $R$, and $W^T$.
11:        Update composite permutation $P^{[J]} = P^{[J-1]}P_b^{[J]}$.
12:        Form selected columns $\hat{A}(c+1:m, c+1:c+k)$ from prior reflections.
13:        Factorize selected columns $[Y^{[J]}, T^{[J]}, R(c+1:c+k, c+1:c+k)] = qr(\hat{A})$.
14:        Merge new reflector inner products in $W^T$.
15:        Complete $R(c+1:c+k, c+k+1:n)$ from all reflections.
16:        if $(c+k=r)$ then exit
17:        Update sample with original or alternative update formula.
18:        Increment completed column counter $c = c + k$.
19:    end do
20: end function
```

3.6 Parallel implementation notes

3.6.1 Memory affinity

The primary concern in designing an efficient parallel algorithm is minimizing slow memory movement. Memory movement is handled automatically in OpenMP. When a thread directs a processor to perform a computation, relevant memory is loaded into that processor's cache hierarchy. If we wish to minimize slow memory movement, threads must be written to avoid computations that pick up distant memory. This leads us to a notion of soft ownership.

We consider matrices to be partitioned into blocks that are owned by the last processor to perform a computation using that block. Code is written to restrict computations to particular blocks that are already owned based on thread identity which corresponds to a physical processor identity. Avoiding ownership changes reduces communication. Since a processor can only effectively own the limited amount of memory that fits in cache, we can understand efficiency improvements of parallel processing as resulting from increased ownership efficiency. Engaging more processors allows more relevant data to already reside near a processor before computation. This is most effective if a sequence of computations can be structured to maximize continuous ownership of the relevant matrix partitions.

3.6.2 Load balancing

An alternative approach that we briefly tested is dynamic scheduling. Dynamic scheduling distributes portions of each computation to available processors in an effort to keep all processors busy. We find that this is not as efficient as ownership balancing for our purposes. Dynamic scheduling requires additional synchronization points between processors which can be very costly at run time. Furthermore, dynamic scheduling necessarily results in dynamic ownership.

Because we know what data must be handled at each stage of the algorithm, as well as computational complexity, we can predetermine an efficient ownership balance and write code accordingly. As a result, balance is achieved without unnecessary synchronization points and changes in ownership. Dynamic scheduling is a better solution when workloads are difficult to predict and balance at design time.

3.6.3 Ownership layout

When choosing a memory affinity layout, it is useful to consider the effect on the most expensive computational stages of the algorithm. In each of the blocked algorithms we discussed these are the block reflector inner products $W^T = T^T(Y^TA)$ and trailing matrix updates $A := A - YW^T$. These matrix multiplies dominate processing time because the entire trailing matrix A must filter through the set of processors. Fortunately, both of these computations can be performed on blocks of columns independently of other blocks. If I_p represents a column index set corresponding to processor p then a block reflection of the form

$$W^T(:, I_p) = T^T\left(Y^TA^{[J]}(:, I_p)\right) \quad \text{and} \quad A^{[J+1]}(:, I_p) = A^{[J]}(:, I_p) - YW^T(:, I_p)$$

can be completed without any intermediate communication or synchronization points between processors. Only Y and T need to be distributed to all processors before hand. We found this simple form of column-based ownership affinity to be quite successful at achieving desirable performance in parallel scaling.

When the trailing update is required, the column index set I_p is further partitioned into panels based on available cache size. Each panel inner product and update is finished before proceeding to the next panel. Ideally, this allows a single load and store operation per panel rather than a load/store pair for the inner product followed by a second load/store pair for the update. Otherwise if the trailing update can be avoided, which is the case for low-rank approximations, this additional layer of panelling is unnecessary.

We tested two 1D column-based affinity structures. The first version uses a contiguous division layout. At every stage, all remaining columns are equally divided into contiguous regions for each processor. For example, if we have P processors and 3 subpanels per processor block then the affinity structure would be

$$\left[\begin{array}{c|ccc|ccc|c|ccc} \text{completed columns} & I_{1,1} & I_{1,2} & I_{1,3} & I_{2,1} & I_{2,2} & I_{2,3} & \cdots & I_{P,1} & I_{P,2} & I_{P,3} \end{array}\right].$$

The first subscript denotes the processor used to operate on the given partition. The second subscript indicates sub-panelling structure for the trailing update computation.

The second style we tested uses a fixed 1D block-cyclic layout, which was suggested by Professor Laura Grigori [26]. An example of this affinity structure might be

$$\left[\begin{array}{cccc|cccc|ccc} I_{1,1} & I_{2,1} & \cdots & I_{P,1} & I_{1,2} & I_{2,2} & \cdots & I_{P,2} & \cdots & I_{P-1,3} & I_{P,3} \end{array}\right].$$

As columns are completed the affinity structure does not change. Processing loads remain nearly balanced while limiting changes in column ownership to only those that must be permuted. We believe this block-cyclic layout will be superior on very large distributed memory systems. However, on the shared memory system we tested, this layout ran somewhat slower. This is likely a consequence of reduced memory prefetching efficiency.

Since a processor's collection of panels is too large to reside in cache, additional boundary communication is subsumed by cache transfers that could not be avoided in either layout. The contiguous layout allows hardware to correctly predict memory that will be needed since addresses are requested in consecutive sequence. However the block cyclic layout interleaves large sections of consecutive access with very large jumps thus causing hardware predictions to fail periodically.

A distributed memory implementation could be designed to enjoy the communication advantage of the fixed block-cyclic structure while also retaining prefetching efficiency. Each node would simply store panels from the fixed block-cyclic layout in consecutive blocks of a single local array. Within a shared-memory node, processors would still utilize contiguous divisions.

3.6.4 Tall-skinny matrix operations

Performance can be improved further by optimizing ownership in tall-skinny matrix operations such as reflector construction. Row-based affinity structures improve parallel efficiency in these cases because the resulting partitions have better matrix dimension to computational dimension ratios than would be the case for column-based affinity. That is, the computational density of corresponding matrix operations is higher [2]. Again, we find the contiguous division layout produces the best results.

3.7 Truncated approximate Singular Value Decomposition

TRQRCP naturally extends to a truncated approximation of the Singular Value Decomposition (SVD). This follows the QLP method proposed by Stewart [66]. The QLP decomposition proceeds by first applying QRCP to obtain $AP_0 = Q_0R$. Then the right triangular matrix R is factored again using an LQ factorization $P_1R = LQ_1$ where row-pivoting is an optional safeguard (otherwise $P_1 = I$). This gives the factored form $A = (Q_0P_1^T)L(Q_1P_0^T)$. The diagonal elements of L give a very good approximation of the singular values of A. Analysis is done by Huckaby and Chan [36].

The truncated approximate SVD proposed here simply applies low-rank versions of the steps in QLP. The rank-k approximation that results is exactly the same as the truncated approximation that would be obtained if QLP had been processed to completion using RQRCP—without secondary row-pivoting—and then truncated to a rank-k approximation.

We begin by using TRQRCP to produce k left reflectors to obtain the initial left orthogonal matrix $U^{(0)}$. Results are simultaneously compared to what would have been obtained by full RQRCP-QLP:

$$A \begin{bmatrix} P_1^{(0)} & P_2^{(0)} \end{bmatrix} \approx \begin{bmatrix} U_1^{(0)} & U_2^{(0)} \end{bmatrix} \begin{bmatrix} R_{11}^{(0)} & R_{12}^{(0)} \\ 0 & 0 \end{bmatrix}.$$

Continuing the full factorization would produce additional pivoting to factorize the rest of the matrix:

$$A \begin{bmatrix} P_1^{(0)} & P_2^{(0*)} \end{bmatrix} = \begin{bmatrix} U_1^{(0)} & U_2^{(0*)} \end{bmatrix} \begin{bmatrix} R_{11}^{(0)} & R_{12}^{(0*)} \\ 0 & R_{22}^{(0*)} \end{bmatrix}.$$

Clearly the first k pivots in $P^{(0)}$ and corresponding reflectors in $U^{(0)}$ are the same. The corresponding rows in $R^{(0)}$ are also the same modulo additional column permutations. We can reverse these permutations to obtain $R^{(1)} = R^{(0)} P^{(0)T}$:

$$A \approx \begin{bmatrix} U_1^{(0)} & U_2^{(0)} \end{bmatrix} \begin{bmatrix} R_{11}^{(1)} & R_{12}^{(1)} \\ 0 & 0 \end{bmatrix} \quad \text{versus} \quad A = \begin{bmatrix} U_1^{(0)} & U_2^{(0*)} \end{bmatrix} \begin{bmatrix} R_{11}^{(1)} & R_{12}^{(1)} \\ R_{21}^{(1*)} & R_{22}^{(1*)} \end{bmatrix}.$$

Taking the LQ factorization $L^{(1)} V^{(1)T} = R^{(1)}$ instead of from $R^{(0)}$ simply absorbs the permutation $P^{(0)T}$ into the definition of $V^{(1)T}$. That gives

$$A \approx \begin{bmatrix} U_1^{(0)} & U_2^{(0)} \end{bmatrix} \begin{bmatrix} L_{11}^{(1)} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^{(1)T} \\ V_2^{(1)T} \end{bmatrix}.$$

Again, continuing the full factorization would produce

$$A = \begin{bmatrix} U_1^{(0)} & U_2^{(0*)} \end{bmatrix} \begin{bmatrix} L_{11}^{(1)} & 0 \\ L_{21}^{(1*)} & L_{22}^{(1*)} \end{bmatrix} \begin{bmatrix} V_1^{(1)T} \\ V_2^{(1*)T} \end{bmatrix}.$$

For consistency with the form that follows, we could label $X^{(0)} = L_{11}^{(1)}$. The leading k reflectors in $V^{(1)}$ and $V^{(1*)}$ are identical because they are only based on the leading k rows of $R^{(1)}$ as long as no secondary row-pivoting is considered.

At this point, the rank-k approximation of RQRCP-QLP would require $L_{21}^{(1*)}$, which is unknown. Fortunately, the leading k columns of $(U^{(0*)} L^{(1*)})$ can be reconstructed with one BLAS-3 matrix multiply:

$$A V_1^{(1)} =: Z = \begin{bmatrix} U_1^{(0)} & U_2^{(0*)} \end{bmatrix} \begin{bmatrix} L_{11}^{(1)} \\ L_{21}^{(1*)} \end{bmatrix} \quad \text{in both cases.}$$

The resulting $m \times k$ matrix is QR-factorized as $Z = U^{(1)} X^{(1)}$ to produce the rank-k approximation desired:

$$A \approx U^{(1)} \begin{bmatrix} X^{(1)} & 0 \\ 0 & 0 \end{bmatrix} V^{(1)^T}. \tag{3.4}$$

Further iterations could be computed to produce $X^{(2)}$, $X^{(3)}$, etc. which would oscillate between upper triangular and lower triangular $k \times k$ matrices. One would simply multiply the leading rows of U^T or columns of V on the left and right of A respectively until the norm of X converged. This type of iterative scheme is outlined in Algorithm 19, however as Stewart observed, only one QR-LQ iteration is needed to produce a strong approximation of the SVD.

Algorithm 19 Truncated Approximate SVD (TUXV)

Input:

A is $m \times n$ matrix to approximate.
k approximation rank. $k \ll \min(m, n)$.
$0 < \tau < 1$ is tolerance of consecutive norm ratios.
$i_{\max}$ is maximum number of LQ-QR iterations. Usually $i_{\max} = 1$.

Output:

U is orthogonal matrix represented by $m \times k$ compact Y_U and T_U.
V is orthogonal matrix represented by $n \times k$ compact Y_V and T_V.
X is $k \times k$ upper or lower triangular matrix.
$A \approx U(:, 1:k)XV(:, 1:k)^T$.

```
1: function $[U, X, V]$=tuxv$(A, k, \tau, i_{\max})$
2:     $[Y_U, T_U, Z, p]$ = rqrcp$(A, k)$
3:     Revert permutation $A(:, p) = A$.
4:     Compute norm of leading upper triangle $n_0 = \|Z(:, 1:k)\|$.
5:     Revert permutation $Z(:, p) = Z$.
6:     LQ-Factorize $[Y_V, T_V, X^T] = qr(Z^T)$.
7:     Compute norm of leading lower triangle $n_1 = \|X\|$.
8:     Initialize LQ-QR iteration count $i = 0$.
9:     do while $\frac{n_0}{n_1} < \tau$ and $i < i_{\max}$
10:        $Z = A(1:m, 1:k) - AY_V T_V Y_V(1:k, 1:k)^T$
11:        QR-Factorize $[Y_U, T_U, X] = qr(Z)$.
12:        Update norm sequence $n_0 = n_1$ and upper triangle norm $n_1 = \|X\|$.
13:        i=i+1;
14:        if $\frac{n_0}{n_1} \geq \tau$ or $i = i_{\max}$ then exit
15:        $Z = A(1:k, 1:n) - Y_U(1:k, 1:k)T_U^T Y_U^T A$
16:        LQ-Factorize $[Y_V, T_V, X^T] = qr(Z^T)$.
17:        Update norm sequence $n_0 = n_1$ and lower triangle norm $n_1 = \|X\|$.
18:        i=i+1;
19:    end do
20: end function
```

3.8 Experimental performance

The first set of experiments test scaling of decomposition time versus various problem dimensions for several full matrix decompositions. These experiments were run on a single node of the NERSC machine Edison. Each node has two 12-core Intel processors. Subroutines were compiled using Intel's Math Kernel Library. Unless stated otherwise, all tests were performed using 24 cores. Each matrix used is randomly generated 12000×12000 unless indicated otherwise. The same random matrix is submitted to each algorithm. These tests show that the full RQRCP decomposition can be computed an order of magnitude faster than QRCP as implemented in `dgeqp3`.

The pivots resulting from RQRCP are not identical to QRCP, however we show that at every stage of the partial factorization the resulting basis is indistinguishably well-suited to represent the matrix thus far. In the second set of experiments we compare the Frobenius norm of the partial factorization of rank-k, R_k, resulting from QR, QRCP, and RQRCP to the corresponding rank-k truncated SVD, Σ_k. Since the Frobenius norms are often nearly equivalent we plot the relative deficiency in log-scale:

$$\log\left(\frac{\|\Sigma_k\|_F - \|R_k\|_F}{\|\Sigma_k\|_F}\right).$$

In these quality tests we examine three structured matrices (images) and one random matrix. These tests demonstrate that both QRCP and RQRCP result in a much stronger basis for representing the matrix than unpivoted QR. Pivot quality is also visually demonstrated in the truncated decomposition image reconstruction experiments that follow.

The third set of tests compare truncated approximations using a default rank $k = 1200$ unless stated otherwise. We include an implementation of Truncated QR (TQR) that avoids the trailing matrix update. This version uses the same subroutines as TRQRCP, but without sample-based pivoting. In practice, TQR is worthless because it produces very poor approximations, however we include it to provide an experimental lower bound on decomposition performance. In effect, TQR reveals the overhead in TRQRCP due to sample computations and column pivoting.

Truncated timing results also include TRQRCP-TU, which is the version that computes the Trailing Update. This is the simple modification of RQRCP discussed in Section 3.5.1. Algorithm 17 is simply halted after k columns have been processed. These experiments verify that avoiding the trailing update reduces computation time by half, provided the truncation rank is small enough.

The version of TUXV tested only applies a single LQ-QR iteration ($i_{\max} = 1$), which returns X as an upper triangular matrix. No threshold testing for convergence is attempted. As a result, TUXV performs just one additional BLAS-3 matrix multiply with a reflector block of dimension $n \times k$ over what is performed by TRQRCP.

A fourth set of tests are intended to compare truncated decomposition approximation quality on structured matrices. Three grayscale images are decomposed and reconstructed at 10% of the original rank. Image reconstruction error is measured in the Frobenius norm relative to the original image norm. These tests show that RQRCP gives comparable results to QRCP. Unfavorable columns are easily rejected early on in favor of a column basis that provides a good low rank approximation of the matrix. TQR is included to verify that it produces poor approximations, however quickly. The truncated SVD is also included in image reconstructions in order to compare against the theoretically optimal low-rank approximation.

These test demonstrate the dramatic performance improvement these randomized methods are capable of achieving for large matrices. These image reconstruction experiments show that just one additional pass over the matrix in TUXV significantly improves low-rank approximations for these structured matrices.

3.8.1 Full decomposition row scaling

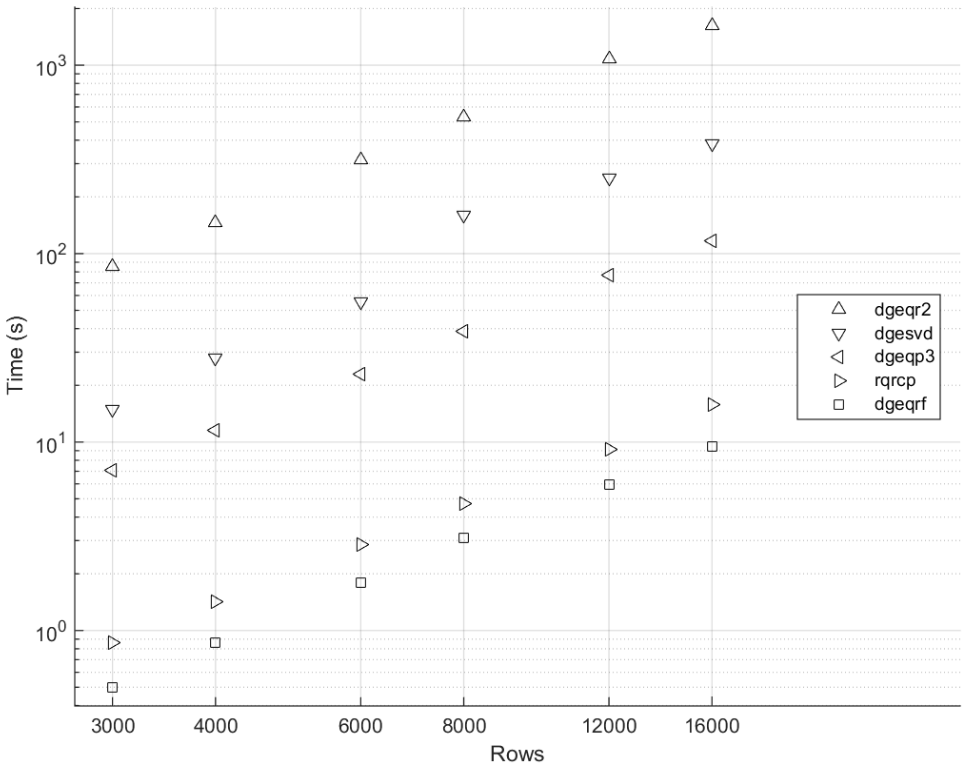


Figure 3.2: Processing time versus rows

m	3000	4000	6000	8000	12000	16000	Exponent
dgeqr2	85.73	146.04	314.06	531.42	1077.07	1627.80	1.78
dgesvd	14.91	27.96	55.81	159.37	252.94	381.55	1.99
dgeqp3	7.09	11.52	22.99	38.92	76.87	116.85	1.69
rqrcp	0.86	1.43	2.85	4.74	9.10	15.78	1.72
dgeqrf	0.50	0.86	1.78	3.09	5.92	9.52	1.76

3.8.2 Full decomposition column scaling

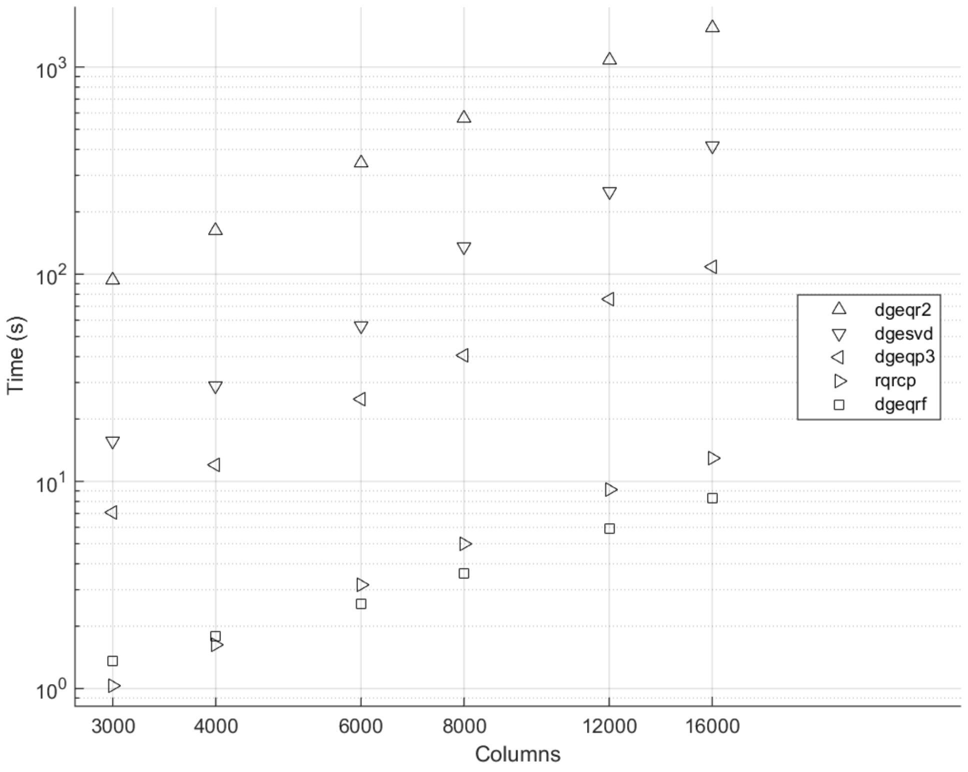


Figure 3.3: Processing time versus columns

n	**3000**	**4000**	**6000**	**8000**	**12000**	**16000**	**Exponent**
dgeqr2	94.05	162.83	343.23	567.02	1077.07	1553.57	1.69
dgesvd	15.73	28.79	56.47	135.46	250.66	417.47	1.98
dgeqp3	7.07	12.07	25.01	40.64	75.66	109.08	1.65
rqrcp	1.03	1.63	3.17	5.00	9.08	12.98	1.53
dgeqrf	1.35	1.79	2.57	3.61	5.94	8.26	1.09

3.8.3 Full decomposition order scaling

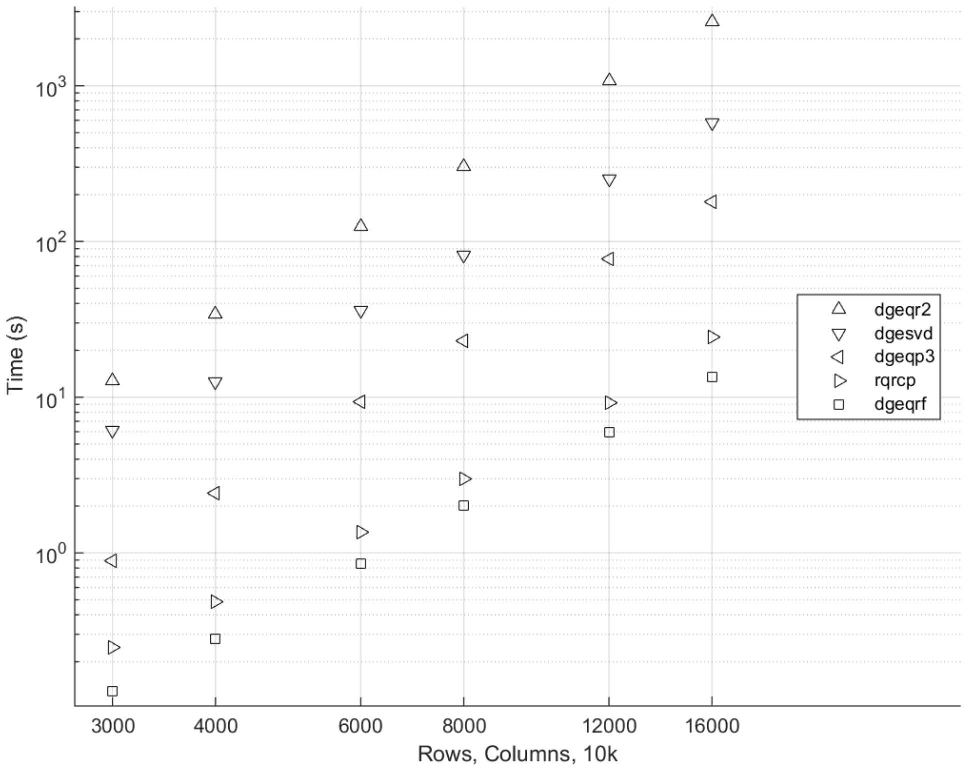


Figure 3.4: Processing time versus order

m,n	3000	4000	6000	8000	12000	16000	Exponent
dgeqr2	12.75	34.08	124.74	304.59	1075.03	2562.74	3.16
dgesvd	6.15	12.65	36.22	81.97	253.18	581.31	2.72
dgeqp3	0.89	2.42	9.32	23.11	76.91	179.13	3.16
rqrcp	0.25	0.48	1.37	2.99	9.28	24.44	2.73
dgeqrf	0.13	0.28	0.85	2.02	5.97	13.54	2.78

3.8.4 Full decomposition parallel core scaling

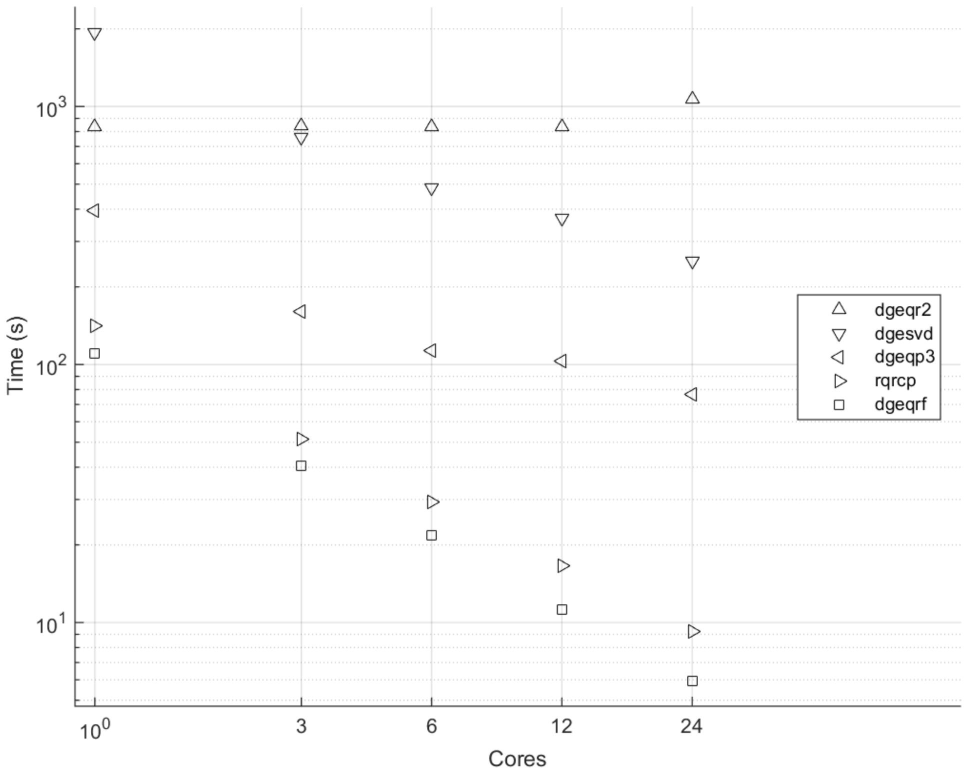


Figure 3.5: Processing time versus cores

cores	**1**	**3**	**6**	**12**	**24**	**Exponent**
dgeqr2	832.78	840.28	834.17	835.64	1069.37	0.06
dgesvd	1939.46	757.27	482.63	369.38	252.87	-0.63
dgeqp3	394.85	159.96	113.68	103.27	76.78	-0.49
rqrcp	141.80	51.54	29.26	16.56	9.24	-0.85
dgeqrf	110.64	40.56	21.82	11.24	5.92	-0.92

3.8.5 Partial decomposition norm comparisons

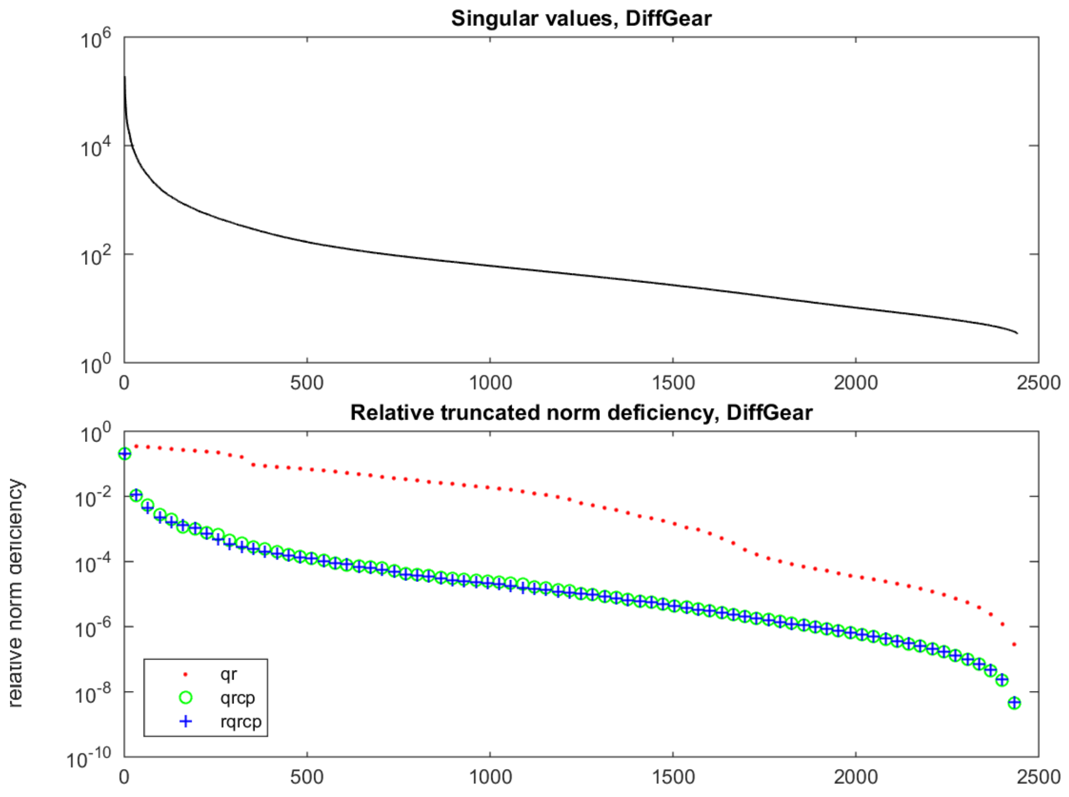


Figure 3.6: Truncated norm comparison, Differential Gear

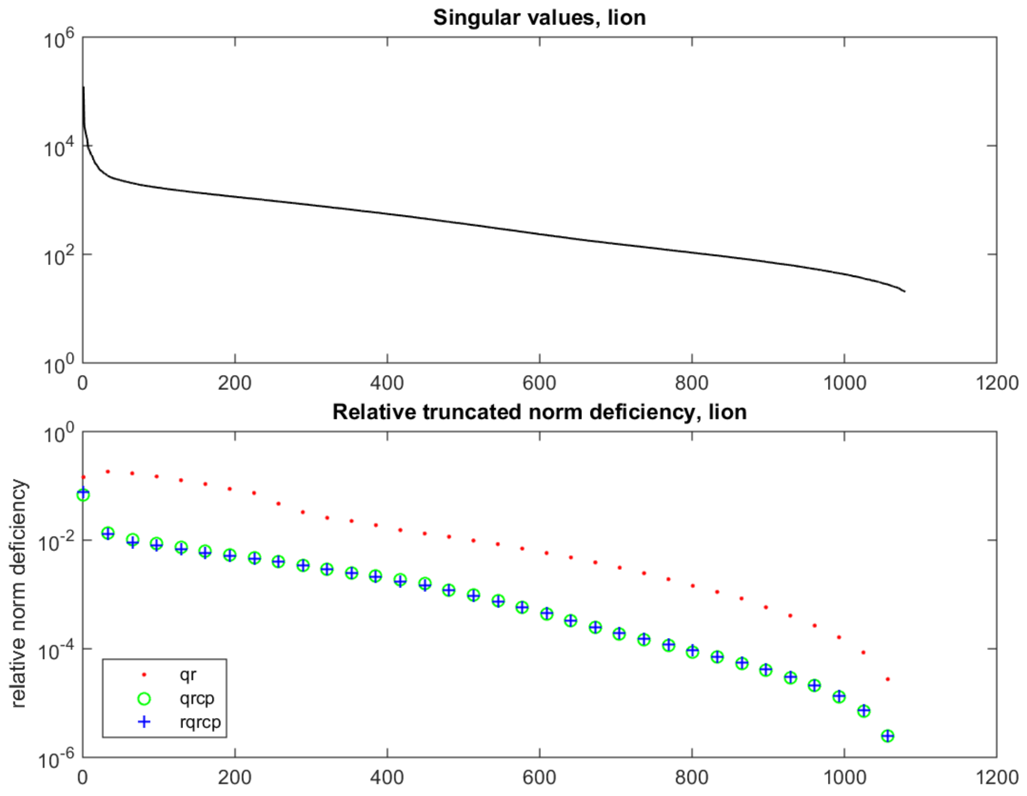


Figure 3.7: Truncated norm comparison, Lion

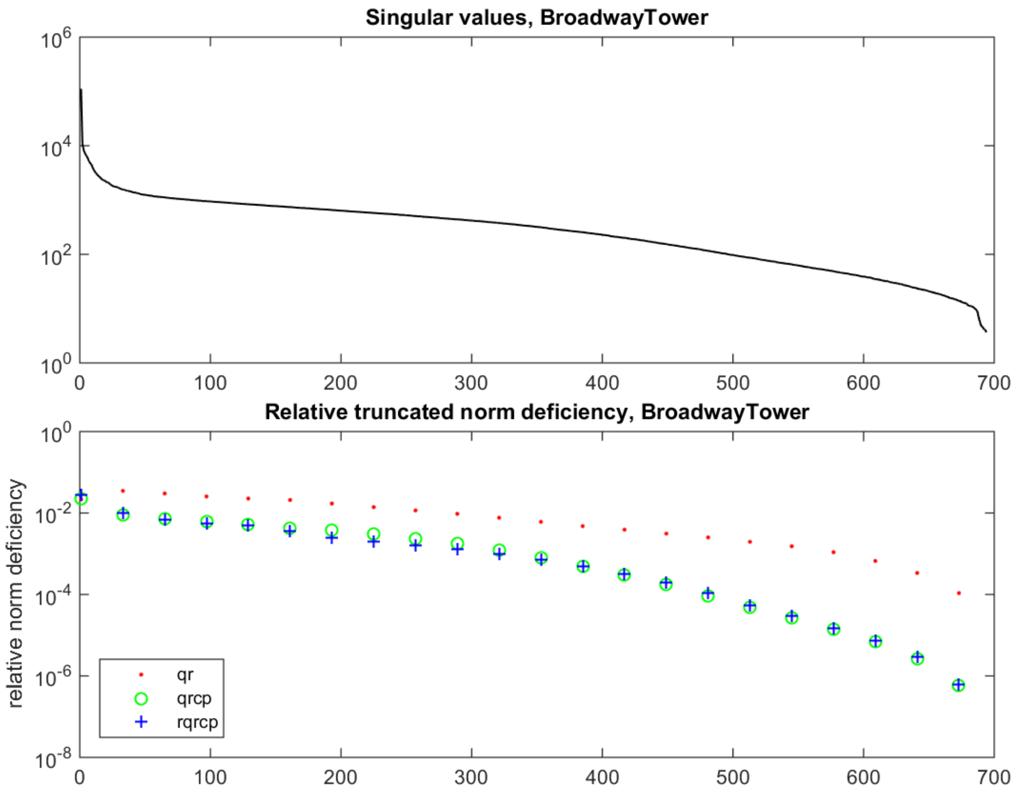


Figure 3.8: Truncated norm comparison, Broadway Tower

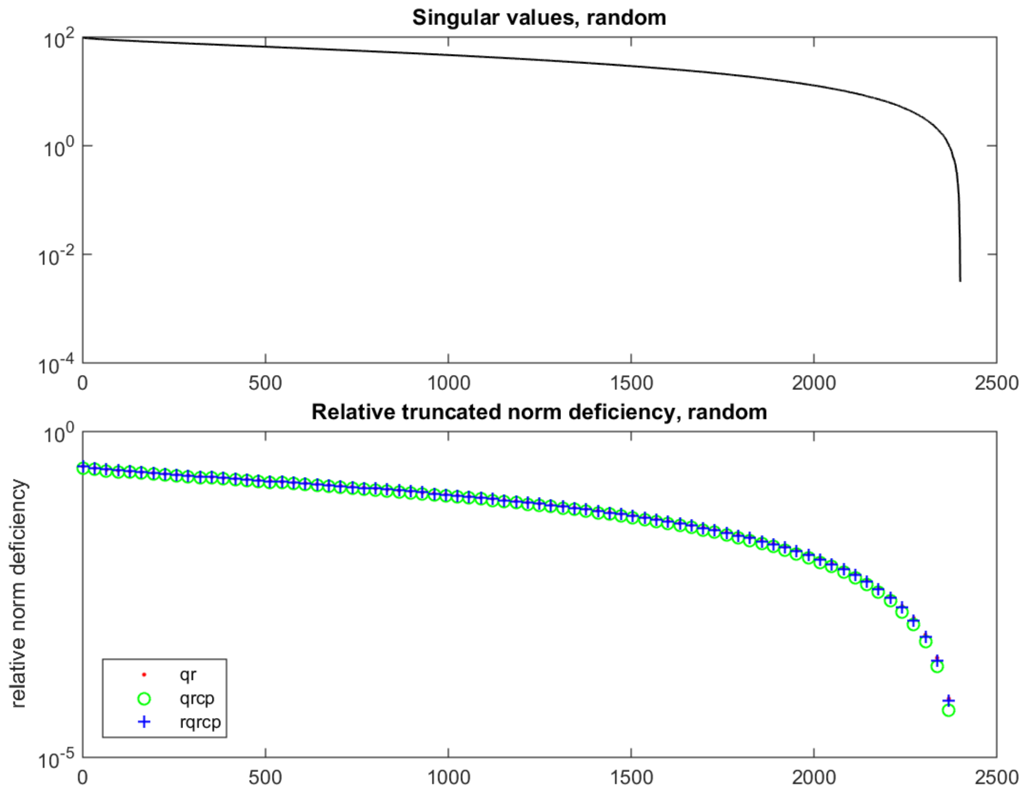


Figure 3.9: Truncated norm comparison, random matrix

3.8.6 Truncated decomposition row scaling

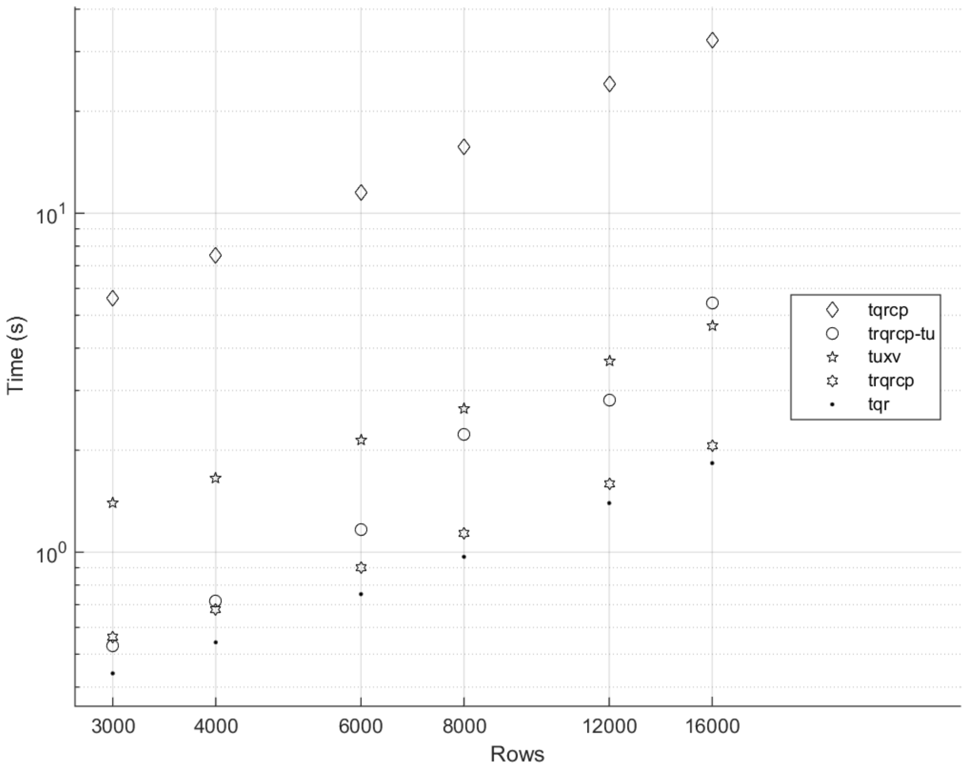


Figure 3.10: Processing time versus rows

m	3000	4000	6000	8000	12000	16000	Exponent
tqrcp	5.61	7.52	11.48	15.74	24.11	32.42	1.05
trqrcp-tu	0.53	0.72	1.16	2.23	2.81	5.43	1.36
tuxv	1.40	1.66	2.14	2.65	3.67	4.65	0.72
trqrcp	0.56	0.68	0.90	1.14	1.59	2.06	0.78
tqr	0.44	0.54	0.75	0.97	1.40	1.83	0.86

3.8.7 Truncated decomposition column scaling

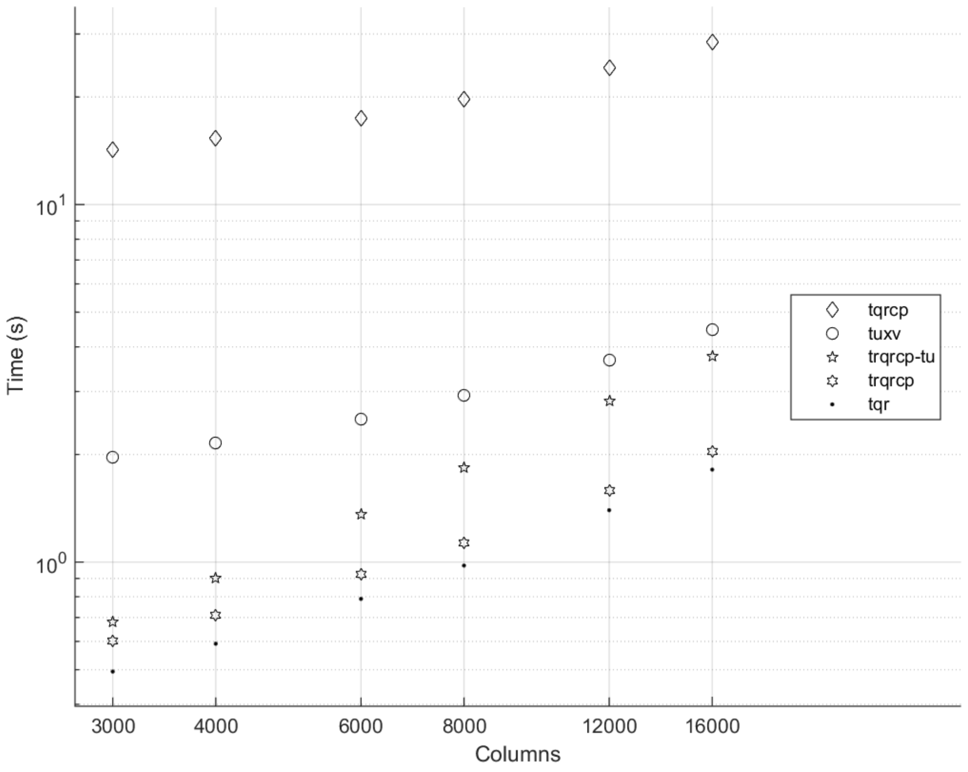


Figure 3.11: Processing time versus columns

n	3000	4000	6000	8000	12000	16000	Exponent
tqrcp	14.25	15.31	17.48	19.66	24.10	28.51	0.41
tuxv	1.96	2.15	2.52	2.93	3.68	4.46	0.49
trqrcp-tu	0.68	0.90	1.36	1.83	2.82	3.77	1.03
trqrcp	0.60	0.71	0.92	1.13	1.58	2.04	0.73
tqr	0.49	0.59	0.79	0.98	1.40	1.81	0.78

3.8.8 Truncation rank scaling

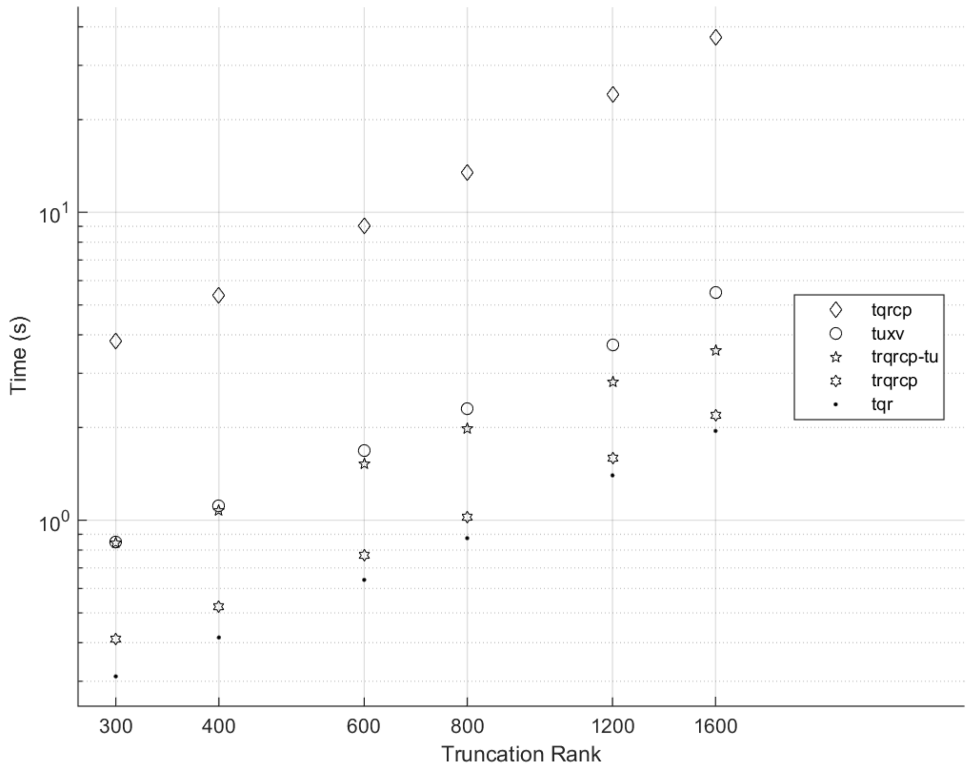


Figure 3.12: Processing time versus truncated rank

k	**300**	**400**	**600**	**800**	**1200**	**1600**	**Exponent**
tqrcp	3.81	5.38	9.06	13.44	24.08	37.05	1.36
tuxv	0.85	1.11	1.68	2.31	3.70	5.49	1.11
trqrcp-tu	0.84	1.07	1.52	1.98	2.80	3.55	0.87
trqrcp	0.41	0.53	0.77	1.02	1.59	2.19	1.00
tqr	0.31	0.42	0.64	0.87	1.40	1.95	1.10

3.8.9 Truncated decomposition parallel core scaling

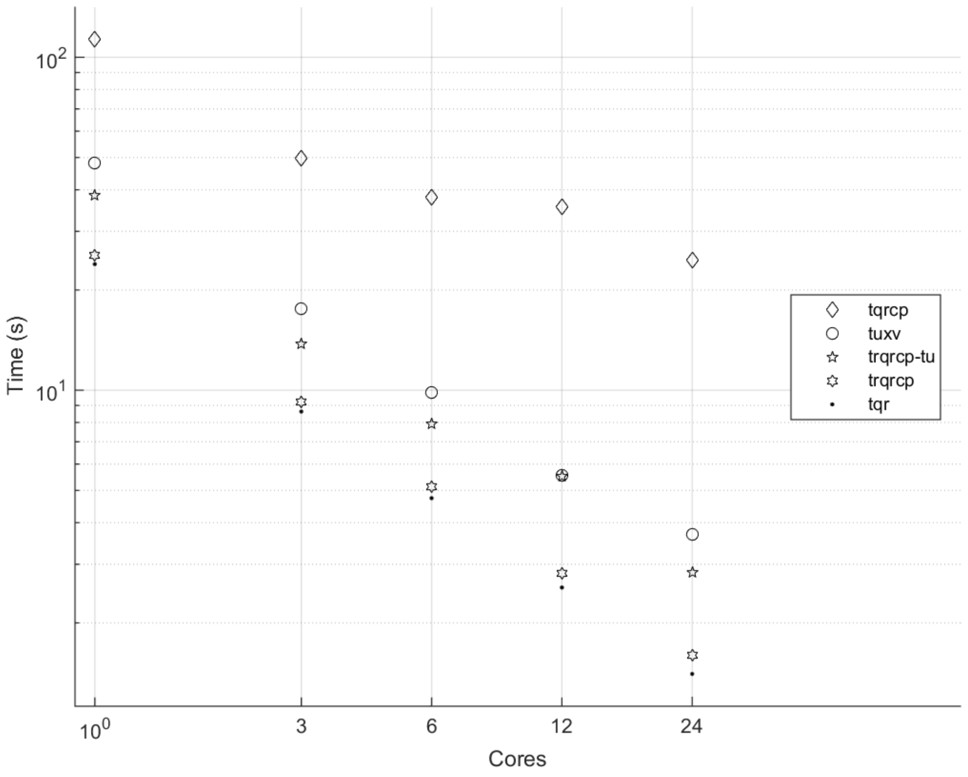
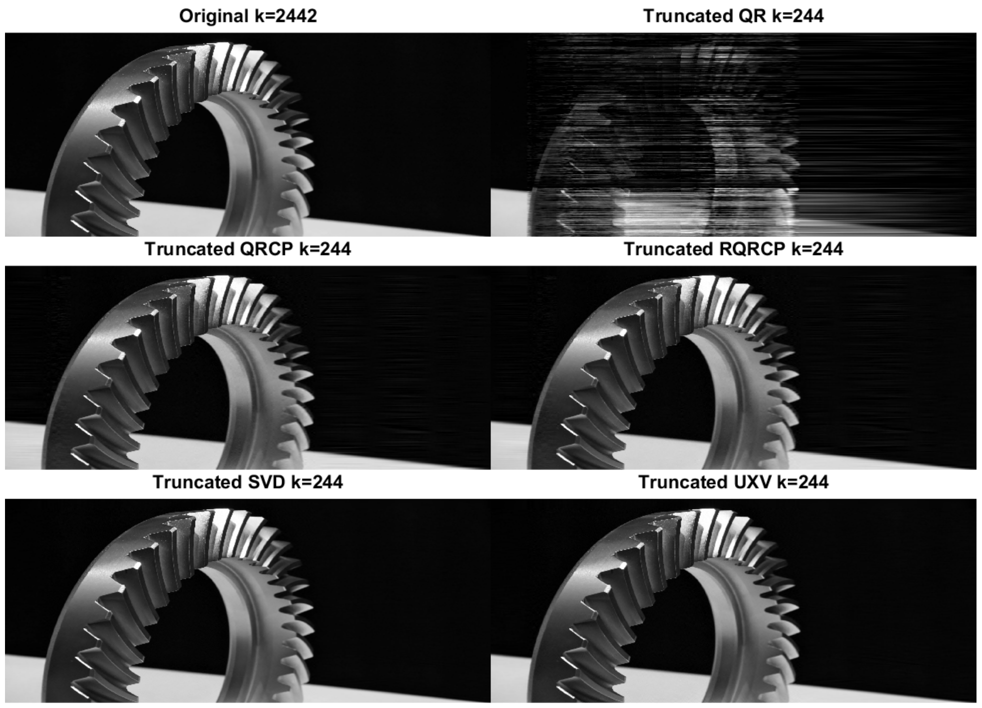


Figure 3.13: Processing time versus cores

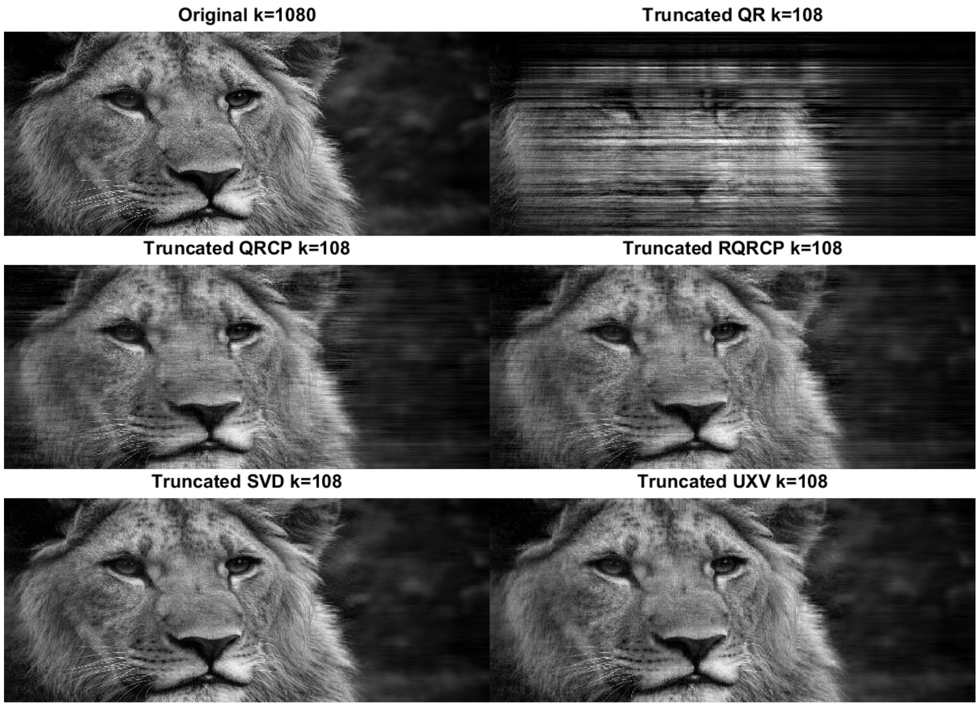
cores	1	3	6	12	24	Exponent
tqrcp	113.06	49.70	37.88	35.56	24.65	-0.45
tuxv	48.05	17.61	9.86	5.56	3.69	-0.82
trqrcp-tu	38.48	13.73	7.94	5.53	2.84	-0.80
trqrcp	25.41	9.22	5.12	2.81	1.60	-0.87
tqr	23.91	8.63	4.74	2.56	1.41	-0.89

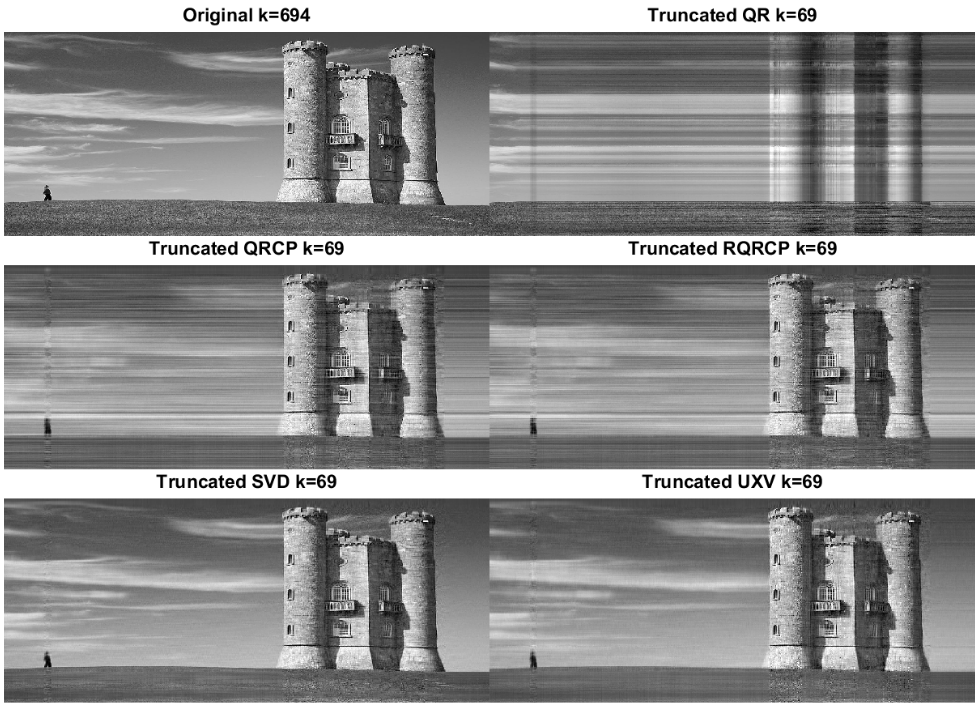
3.8.10 Approximation quality



Relative truncation error of images in Frobenius norm:

Image	Diff_Gear	Lion	Tower
tqr	63.91%	52.46%	26.08%
tqrcp	4.34%	18.82%	15.41%
trqrcp	4.34%	18.56%	15.40%
tuxv	2.59%	15.46%	11.16%
tsvd	2.20%	13.87%	10.08%


Original k=1080
Truncated QR k=108
Truncated QRCP k=108
Truncated RQRCP k=108
Truncated SVD k=108
Truncated UXV k=108


Original k=694
Truncated QR k=69
Truncated QRCP k=69
Truncated RQRCP k=69
Truncated SVD k=69
Truncated UXV k=69

Bibliography

[1] R. I. Arriaga and S. Vempala. An algorithmic theory of learning: Robust concepts and random projection. *Mach. Learn.*, 63(2):161–182, 2006.

[2] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Mat. Anal. Appl.*, 32(3):866–901, 2011.

[3] M. Benzi and M. Tuma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Scientific Computing*, 19(3):968–994, 1998.

[4] L. Bergamaschi, G. Pini, and F. Sartoretto. Parallel preconditioning of a sparse eigensolver. *Parallel Computing*, 27(7):963–976, 2001.

[5] S. Birk and A. Frommer. A CG method for multiple right hand sides and multiple shifts in lattice QCD calculations. In *PoS*, 2012.

[6] C. Bischof. A pipelined block qr decomposition algorithm. In *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing*, pages 3–7, Philadelphia, PA, USA, 1989. Society for Industrial and Applied Mathematics.

[7] C. H. Bischof. A parallel QR factorization algorithm with controlled local pivoting. 12(1):36–57, Jan. 1991.

[8] T. F. Chan. Rank revealing QR factorizations. *Linear Algebra Appl.*, 88–89:67–82, 1987.

[9] T. F. Chan and P. C. Hansen. Some applications of the rank revealing QR factorization. *SIAM J. Sci. Stat. Comput.*, 13(3):727–741, May 1992.

[10] T.-Y. Chen. *Preconditioning Sparse Matrices for Computing Eigenvalues and Solving Linear Systems of Equations.* PhD thesis, University of California at Berkeley, 2001.

[11] T.-Y. Chen and J. W. Demmel. Balancing sparse matrices for computing eigenvalues. *Linear algebra and its applications*, 309, 2000.

[12] E. Chow and Y. Saad. Experimental study of ilu preconditioners for indefinite matrices. *J. COMPUT. APPL. MATH*, 86:387–414, 1997.

[13] S. Dasgupta and A. Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, 2003.

[14] J. Demmel. On floating point errors in Cholesky. LAPACK Working Note 14, Oct. 1989. UT-CS-89-87, October 1989.

[15] J. Demmel, L. Grigori, M. Gu, and H. Xiang. Communication avoiding rank revealing QR factorization with column pivoting. Technical Report UCB/EECS-2013-46, EECS Department, University of California, Berkeley, May 2013.

[16] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential qr and lu factorizations. *SIAM J. Scientific Computing*, 34(1), 2012.

[17] J. W. Demmel, J. Dongarra, B. Parlett, W. Kahan, M. Gu, D. Bindel, Y. Hida, X. S. Li, O. A. Marques, E. J. Riedy, C. Vmel, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, J. Langou, and S. Tomov. Prospectus for the next lapack and scalapack libraries. In *PARA'06: State-of-the-Art in Scientific and Parallel Computing*, Ume, Sweden, June 2006. High Performance Computing Center North (HPC2N) and the Department of Computing Science, Ume University, Springer.

[18] J. A. Duersch and M. Gu. Randomized strong rank-revealing qr factorization. UC Berkeley, Spring 2014, Math 273 final project presentation, May 2014.

[19] J. v. d. Eshof and G. L. G. Sleijpen. Accurate conjugate gradient methods for shifted systems. 2003.

[20] M. Ferronato, E. Chow, and K. K. Phoon. Preconditioning techniques for sparse linear systems. *J. Applied Mathematics*, 2012:518165:1–518165:3, 2012.

[21] M. Ferronato, C. Janna, and G. Pini. Efficient parallel solution to large-size sparse eigenproblems with block fsai preconditioning. *Numerical Lin. Alg. with Applic.*, 19(5):797–815, 2012.

[22] D. R. Fokkema, G. L. G. Sleijpen, and H. A. V. der Vorst. Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils. *J. Sci. Comput.*, 20:94–125, 1998.

[23] L. V. Foster and X. Liu. Comparison of rank revealing algorithms applied to matrices with well defined numerical rank. Technical report, 2012.

[24] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, 3rd edition, 1996.

[25] G. H. Golub and C. F. van Loan. *Matrix Computations*. JHU Press, 4th edition, 2013.

[26] L. Grigori. personal communication.

[27] R. G. Grimes, J. G. Lewis, and H. D. Simon. A shifted block lanczos algorithm for solving sparse symmetric generalized eigenproblems, 1994.

[28] M. Gu and S. C. Eisenstat. Efficient algorithms for computing a strong rank-revealing *QR* factorization. *SIAM J. Sci. Comput.*, 17(4):848–869, July 1996.

[29] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[30] S. C. Hawkins and K. C. 0002. An implicit wavelet sparse approximate inverse preconditioner. *SIAM J. Scientific Computing*, 27(2):667–686, 2005.

[31] G. Hechme. Convergence analysis of the jacobi-davidson method applied to a generalized eigenproblem. *C. R. Acad. Sci. Paris, Ser. I*, 345 (5):293–296, 2007.

[32] U. Hetmaniuk and R. Lehoucq. Basis selection in LOBPCG. *J. Comput. Phys.*, 218:324–332, 2006.

[33] U. Hetmaniuk and R. B. Lehoucq. Basis selection in lobpcg. *J. Comput. Physics*, 218(1):324–332, 2006.

[34] M. E. Hochstenbach and Y. Notay. The jacobi–davidson method, Dec. 2004.

[35] M. E. Hochstenbach and G. L. G. Sleijpen. Harmonic and refined rayleigh-ritz for the polynomial eigenvalue problem. *Numerical Lin. Alg. with Applic.*, 15(1):35–54, 2008.

[36] D. A. Huckaby and T. F. Chan. On the convergence of stewart's qlp algorithm for approximating the svd. *Numerical Algorithms*, 32(2-4):287–316, 2003.

[37] Z. Jia and C. Li. Harmonic and refined harmonic shift-invert residual arnoldi and jacobi-davidson methods for interior eigenvalue problems. *J. Computational Applied Mathematics*, 282:83–97, 2015.

[38] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society, 1984.

[39] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[40] A. Khabou, J. Demmel, L. Grigori, and M. Gu. Lu factorization with panel rank revealing pivoting and its communication avoiding version. Technical Report UCB/EECS-2012-15, EECS Department, University of California, Berkeley, Jan 2012.

[41] A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM J. Sci. Comput.*, 23(2):517–541, 2001.

[42] A. V. Knyazev, M. E. Argentati, I. Lashuk, and E. E. Ovtchinnikov. Block locally optimal preconditioned eigenvalue xolvers (BLOPEX) in hypre and PETSc. *SIAM J. Sci. Comput.*, 29(5):2224–2239, 2007.

[43] R. Li and Y. Saad. Divide and conquer low-rank preconditioners for symmetric matrices. *SIAM J. Scientific Computing*, 35(4), 2013.

[44] X. S. Li and J. Demmel. Superlu_dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw.*, 29(2):110–140, 2003.

[45] E. Liberty, F. Woolfe, P. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167, 2007.

[46] L. Lin. personal communication.

[47] L. Lin, C. Yang, J. C. Meza, J. Lu, L. Ying, and W. E. Selinv - an algorithm for selected inversion of a sparse symmetric matrix. *ACM Trans. Math. Softw.*, 37(4):40, 2011.

[48] M. W. Mahoney. Randomized algorithms for matrices and data. *CoRR*, abs/1104.5557, 2011.

[49] P. G. Martinsson. Blocked rank-revealing qr factorizations: How randomized sampling can be used to avoid single-vector pivoting. May 2015.

[50] P.-G. Martinsson, V. Rokhlin, and M. Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30(1):47 – 68, 2011.

[51] R. B. Morgan. Computing interior eigenvalues of large matrices. *Linear Algebra and its Applications*, 154–156:289–309, 1991.

[52] R. B. Morgan and D. S. Scott. Preconditioning the lanczos algorithm for sparse symmetric eigenvalue problems. *SIAM J. Scientific Computing*, 14(3):585–593, 1993.

[53] E. D. Napoli, E. Polizzi, and Y. Saad. Efficient estimation of eigenvalue counts in an interval. *CoRR*, abs/1308.4275, 2013.

[54] K. Neymeyr. A geometric theory for preconditioned inverse iteration. I. Extrema of the Rayleigh quotient. *Linear Algebra Appl.*, 322(1-3):61–85, 2001.

[55] K. Neymeyr and M. Zhou. Iterative minimization of the Rayleigh quotient by the block steepest descent iteration. Technical report, Universität Rostock, 2013.

[56] P. Quillen and Q. Ye. A block inverse-free preconditioned krylov subspace method for symmetric generalized eigenvalue problems. *J. Computational Applied Mathematics*, 233(5):1298–1313, 2010.

[57] G. Quintana-Ortí, X. Sun, and C. H. Bischof. A blas-3 version of the qr factorization with column pivoting. *SIAM J. Sci. Comput.*, 19(5):1486–1494, Sept. 1998.

[58] M. Rewienski, A. Lamecki, and M. Mrozowski. An extended basis inexact shift-invert lanczos for the efficient solution of large-scale generalized eigenproblems. *Computer Physics Communications*, 184(9):2127–2135, 2013.

[59] Y. Saad. *Iterative Methods for sparse linear systems.* SIAM, 2nd edition, 2003.

[60] T. Sarlós. Improved approximation algorithms for large matrices via random projections. In *FOCS*, pages 143–152. IEEE Computer Society, 2006.

[61] O. Schenk and K. Grtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Future Generation Comp. Syst.*, 20(3):475–487, 2004.

[62] G. L. G. Sleijpen and H. A. V. der Vorst. A jacobi-davidson iteration method for linear eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 17:401–425, 1996.

[63] G. L. G. Sleijpen and H. A. van der Vorst. A jacobi-davidson iteration method for linear eigenvalue problems. *SIAM Review*, 42(2):267–293, 2000.

[64] A. Stathopoulos and K. Orginos. Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics. *SIAM J. Sci. Comput.*, 32(1):439–462, 2010.

[65] A. Stathopoulos and K. Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM J. Sci. Comput.*, 23(6):2162–2182, 2002.

[66] G. W. Stewart. The qlp approximation to the singular value decomposition. *SIAM J. Scientific Computing*, 20(4):1336–1348, 1999.

[67] E. Vecharynski and A. V. Knyazev. Absolute value preconditioning for symmetric indefinite linear systems. *SIAM J. Scientific Computing*, 35(2), 2013.

[68] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335 – 366, 2008.