SIAM J. Sci. Comput.
Vol. 35, No. 3, pp. A1325–A1349

STABLE AND EFFICIENT SPECTRAL DIVIDE AND CONQUER ALGORITHMS FOR THE SYMMETRIC EIGENVALUE DECOMPOSITION AND THE SVD[*]

YUJI NAKATSUKASA[†] AND NICHOLAS J. HIGHAM[†]

Abstract. Spectral divide and conquer algorithms solve the eigenvalue problem for all the eigenvalues and eigenvectors by recursively computing an invariant subspace for a subset of the spectrum and using it to decouple the problem into two smaller subproblems. A number of such algorithms have been developed over the last 40 years, often motivated by parallel computing and, most recently, with the aim of achieving minimal communication costs. However, none of the existing algorithms has been proved to be backward stable, and they all have a significantly higher arithmetic cost than the standard algorithms currently used. We present new spectral divide and conquer algorithms for the symmetric eigenvalue problem and the singular value decomposition that are backward stable, achieve lower bounds on communication costs recently derived by Ballard, Demmel, Holtz, and Schwartz, and have operation counts within a small constant factor of those for the standard algorithms. The new algorithms are built on the polar decomposition and exploit the recently developed QR-based dynamically weighted Halley algorithm of Nakatsukasa, Bai, and Gygi, which computes the polar decomposition using a cubically convergent iteration based on the building blocks of QR factorization and matrix multiplication. The algorithms have great potential for efficient, numerically stable computations in situations where the cost of communication dominates the cost of arithmetic.

Key words. symmetric eigenvalue problem, singular value decomposition, SVD, polar decomposition, QR factorization, spectral divide and conquer, dynamically weighted Halley iteration, subspace iteration, numerical stability, backward error analysis, communication-minimizing algorithms

AMS subject classifications. 15A23, 65F15, 65F30, 65G50

DOI. 10.1137/120876605

1. Introduction.

A recurring theme over the last 40 years or so has been the use of spectral projectors to solve an eigenproblem by recursively splitting the spectrum via divide and conquer. An early use of this idea is by Beavers and Denman [13], [21, sect. 7], who use the matrix sign function to split the spectra of shifted matrices about the imaginary axis. Lin and Zmijewski [38] develop a more numerically stable algorithm that employs orthogonal bases for the projectors and they implement it on a parallel computer. Similar ideas were developed independently by Auslander and Tsao [3] and, for generalized eigenvalue problems, by Bulgakov and Godunov [15], [23] and Malyshev [39], [40], [41] (see also [24]).

An important step in the practical development of spectral divide and conquer algorithms was the toolbox of Bai and Demmel [5], [6], which provides the building blocks for constructing algorithms via the Newton iteration for the matrix sign function. A parallel implementation based on these ideas is described in [8].

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section May 10, 2012; accepted for publication (in revised form) February 25, 2013; published electronically May 14, 2013.
http://www.siam.org/journals/sisc/35-3/87660.html

[†]School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK (yuji.nakatsukasa@manchester.ac.uk, http://www.ma.man.ac.uk/∼yuji, higham@ma.man.ac.uk, http://www.ma.man.ac.uk/∼higham). The work of the first author was supported by Engineering and Physical Sciences Research Council grant EP/I005293/1. The work of the second author was supported by Engineering and Physical Sciences Research Council grants EP/I006702/1 and EP/E050441/1 (CICADA: Centre for Interdisciplinary Computational and Dynamical Analysis) and European Research Council Advanced grant MATFUN (267526).

Bai, Demmel, and Gu [9] build on the work of Bulgakov, Godunov, and Malyshev to develop an inverse-free spectral divide and conquer algorithm based solely on rank-revealing QR factorization and matrix multiplication, applying to the generalized eigenvalue problem. Huss-Lederman et al. [35] develop a hybrid Newton and inverse-free iteration approach for the generalized eigenvalue problem, targeting parallel computers.

All the above contributions are aimed at general, nonsymmetric matrices. Spectral divide and conquer algorithms for symmetric matrices have been developed in the PRISM project [14], [36], and by Zhang, Zha, and Ying [50], [51].

Very recently, Demmel, Dumitriu, and Holtz [18] have shown how to exploit randomization to allow standard QR factorization to be used throughout the algorithm of Bai, Demmel, and Gu [9], while Ballard, Demmel, and Dumitriu [10] have developed this approach into algorithms that achieve, asymptotically, lower bounds on the costs of communication. The spectral divide and conquer algorithm of Bai and Demmel [7], which is based on the matrix sign function computed by the (scaled) Newton iteration, can also be implemented so as to attain communication lower bounds. With the exception of [51], the above papers develop and test algorithms for a single splitting step rather than for the recursive solution process that yields the eigensystem.

Unfortunately, except for the scaled Newton approach, these spectral divide and conquer algorithms generally require significantly more arithmetic than the standard algorithms based on reduction to condensed (tridiagonal or bidiagonal) form, and the scaled Newton approach has poorer stability than standard algorithms in our experiments. In addition, none of them has been proven to be backward stable in floating point arithmetic. For example, the authors in [10], [18] use probabilistic arguments combined with randomization to argue that the backward error is acceptably small with high probability, and suggest rerunning the process with a randomized starting matrix when the backward error is large. The analyses in [50], [51] assume exact arithmetic.

In this work we develop new spectral divide and conquer algorithms for the symmetric eigendecomposition and the singular value decomposition (SVD) that have a number of novel features. They

- are based on the polar decomposition, so they can exploit fast, backward stable algorithms for computing the polar decomposition;
- require just the building blocks of matrix multiplication and QR factorization (without pivoting);
- do not suffer from a high floating point operation count, slow convergence caused by eigenvalues near a splitting point, or numerical instability—one or more of which have affected all previous spectral divide and conquer algorithms;
- are backward stable under two mild conditions, both of which are easy to verify and almost always satisfied in practice;
- asymptotically minimize communication while at the same time have arithmetic costs within a small factor of those for the standard methods used in LAPACK (≈ 3 for the eigenproblem and ≈ 2 for the SVD).

The main tool underlying our algorithms is the QDWH (QR-based dynamically weighted Halley) algorithm of Nakatsukasa, Bai, and Gygi [45], which is a QR factorization-based algorithm for computing the polar decomposition. For the symmetric eigenproblem, the key fact is that the positive and negative invariant subspaces of a symmetric matrix can be efficiently computed via the orthogonal polar factor. This observation leads to our spectral divide and conquer algorithm QDWH-eig. For an $n \times n$ matrix A, the dominant cost of QDWH-eig is in performing six or fewer QR factorizations of $2n \times n$ matrices. QDWH-eig is much more efficient than the other spectral divide and conquer algorithms proposed in the literature and is the first to have proven backward stability.

Our SVD algorithm, QDWH-SVD, computes the polar decomposition $A = U_pH \in \mathbb{C}^{m\times n}$ using the QDWH algorithm and then uses QDWH-eig to compute the eigendecomposition $H = V\Sigma V^*$. The SVD of A is then $A = (U_pV)\Sigma V^* = U\Sigma V^*$. The essential cost for computing the SVD is in performing QR factorizations of no more than six $(m + n) \times n$ matrices and six $2n \times n$ matrices.

We prove the backward stability of QDWH-eig and QDWH-SVD under two conditions:

1. The polar decompositions computed by QDWH are backward stable.
2. The column space of a computed orthogonal projection matrix is obtained in a backward stable manner.

In [46] we prove that the first condition is satisfied when the QR factorizations are computed with column pivoting and row pivoting or sorting. In practice, even without pivoting QDWH almost always computes the polar decomposition in a backward stable manner, as demonstrated by the experiments in [45], [46]. The second condition holds in exact arithmetic for a subspace obtained by a single step of subspace iteration, and in practice it holds most of the time. Hence both conditions are usually satisfied in practice, and can be inexpensively verified if necessary. Note that backward stability of our algorithms implies that the eigen(singular) values are computed accurately in the absolute (but not necessarily relative) sense, as the eigenvalues of a symmetric matrix and singular values of any matrix are all well conditioned.

As proof of concept, we perform numerical experiments with our algorithms on a shared-memory machine with four processors, employing (via MATLAB function `qr`) the conventional LAPACK QR factorization algorithm that does not minimize communication. Even under such conditions, our algorithms are not too much slower than the conventional algorithms, and are sometimes faster than the traditional eigenvalue and SVD algorithms based on the QR algorithm. On massively parallel computing architectures we expect that the communication-optimality of our algorithms will improve the performance significantly. Furthermore, while both our algorithms and most conventional algorithms are proven to be backward stable and to yield numerically orthogonal computed eigen(singular)-vectors, experiments demonstrate that our algorithms usually give considerably smaller backward errors and vectors closer to orthogonality. Moreover, our algorithms tend to compute the eigenvalues and singular values more accurately, and determine the rank more reliably when applied to rank-deficient matrices.

The rest of the paper is organized as follows. In section 2 we summarize the QR factorization-based polar decomposition algorithm QDWH [45]. In section 3 we develop QDWH-eig, establish its backward stability, and compare it with other algorithms for symmetric eigenproblems. Section 4 describes our SVD algorithm, QDWH-SVD. Section 5 addresses practical implementation issues and techniques to enhance the performance of the algorithms. Numerical experiments are described in section 6 and conclusions are presented in section 7.

We summarize our notation. The singular values of an $m \times n$ rectangular matrix A with $m \geq n$ are $\sigma_1(A) \geq \cdots \geq \sigma_n(A)$ and we write $\sigma_{\max}(A) = \sigma_1(A)$ and $\sigma_{\min}(A) = \sigma_n(A)$. $\|A\|_2 = \sigma_{\max}(A)$ denotes the spectral norm and $\|A\|_F = (\sum_{i,j} |a_{ij}|^2)^{1/2}$ is the Frobenius norm. For a Hermitian matrix A, $\lambda_i(A)$ denotes the ith largest eigenvalue and $\text{eig}(A)$ is the set of all eigenvalues. $\kappa_2(A) = \sigma_{\max}(A)/\sigma_{\min}(A)$ is the condition number of A. A subscript p is used to avoid confusion between the unitary polar factor U_p and the matrix of left singular vectors U. Hence $A = U_pH = U\Sigma V^*$.

We are aiming for backward error bounds that are a modest multiple of the unit roundoff, u, but in our analysis we are not interested in the constants. Following [46], we denote by ϵ a matrix or scalar such that $\|\epsilon\| \le f(n)u$ for some modest function f depending only on n (such as a low-degree polynomial) and some fixed norm. Hence we will write $2\epsilon = \epsilon$, $n\epsilon = \epsilon$, and so on.

We develop algorithms for real matrices $A \in \mathbb{R}^{m\times n}$, but note without further comment that they extend straightforwardly to the complex case.

2. QDWH for the polar decomposition.

We begin by reviewing the QDWH algorithm for computing the polar decomposition [45], which is the foundation of the algorithms in the next two sections.

Any rectangular matrix $A \in \mathbb{R}^{m\times n}$ $(m \ge n)$ has a polar decomposition

$$A = U_pH, \tag{2.1}$$

where U_p has orthonormal columns and H is symmetric positive semidefinite [31, Thm. 8.1]. The decomposition (2.1) is unique if A has full column rank. QDWH computes the polar factor U_p as the limit of the sequence X_k defined by

$$X_{k+1} = X_k(a_kI + b_kX_k^*X_k)(I + c_kX_k^*X_k)^{-1}, \qquad X_0 = A/\alpha. \tag{2.2}$$

Here, $\alpha > 0$ is an estimate of $\|A\|_2$ such that $\alpha \gtrsim \|A\|_2$. Setting $a_k = 3$, $b_k = 1$, $c_k = 3$ gives the Halley iteration, which is the cubically convergent member of the family of principal Padé iterations [31, sect. 8.5]. In QDWH the parameters a_k, b_k, c_k are dynamically chosen to speed up the convergence. They are computed by $a_k = h(\ell_k)$, $b_k = (a_k - 1)^2/4$, $c_k = a_k + b_k - 1$, where $h(\ell) = \sqrt{1+\gamma} + \frac{1}{2}\left(8 - 4\gamma + 8(2-\ell^2)/(\ell^2\sqrt{1+\gamma})\right)^{1/2}$, $\gamma = \left(4(1-\ell^2)/\ell^4\right)^{1/3}$. Here, ℓ_k is a lower bound for the smallest singular value of X_k, which is computed from the recurrence $\ell_k = \ell_{k-1}(a_{k-1} + b_{k-1}\ell_{k-1}^2)/(1 + c_{k-1}\ell_{k-1}^2)$ for $k \ge 1$. Note that all the parameters are available for free (without any matrix computations) for all $k \ge 0$ once we have estimates $\alpha \gtrsim \|A\|_2$ and $\ell_0 \lesssim \sigma_{\min}(X_0)$, obtained for example via a condition number estimator.

With such parameters the iteration (2.2) is cubically convergent and needs at most six iterations for convergence to U_p with the tolerance $u = 2^{-53} \approx 1.1 \times 10^{-16}$ (the unit roundoff for IEEE double precision arithmetic) for any matrix A with $\kappa_2(A) \le u^{-1}$ [45].

Iteration (2.2) has a mathematically equivalent QR-based implementation, which is the practical QDWH iteration:

$$X_0 = A/\alpha, \tag{2.3a}$$

$$\begin{bmatrix} \sqrt{c_k}X_k \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \quad X_{k+1} = \frac{b_k}{c_k}X_k + \frac{1}{\sqrt{c_k}}\left(a_k - \frac{b_k}{c_k}\right)Q_1Q_2^*, \qquad k \ge 0. \tag{2.3b}$$

Note that the main costs of a QDWH iteration (2.3) are one QR factorization of an $(m+n)\times n$ matrix and a matrix multiplication, both of which can be done in a communication-optimal manner [12], [19].

Once the computed polar factor $\widehat{U}_p$ is obtained, we compute the symmetric polar factor $\widehat{H}$ by [31, sect. 8.8]

$$\widehat{H} = \frac{1}{2}(\widehat{U}_p^*A + (\widehat{U}_p^*A)^*). \tag{2.4}$$

Further details of the QWDH algorithm are given in [45].

We will say that the polar decomposition is computed in a backward stable manner if the computed polar factors $\widehat{U}_p, \widehat{H}$ satisfy

$$A = \widehat{U}_p \widehat{H} + \epsilon \|A\|_2, \qquad \widehat{U}_p^* \widehat{U}_p = I + \epsilon. \tag{2.5}$$

Note that the definitions of backward stability in [31, sect. 8.8], [46] require additionally that $\widehat{H}$ is close to a symmetric positive semidefinite matrix. Here we drop this requirement because we do not need it to establish backward stability of QDWH-eig and QDWH-SVD.

3. QDWH-eig: Symmetric eigendecomposition.

We begin by developing our spectral divide and conquer algorithm for the symmetric eigenvalue problem, which is based on spectral splitting achieved via the polar decomposition.

3.1. Invariant subspaces via polar decomposition.

Let $A \in \mathbb{R}^{n\times n}$ be symmetric. We describe how to compute an invariant subspace of A corresponding to the positive (or negative) eigenvalues using the polar decomposition. Assume that A is nonsingular; the singular case is discussed in section 5.4. The first step is to note the connection between the polar decomposition of a symmetric A and its eigendecomposition. Let $A = U_pH$ be the polar decomposition and let $A = V\Lambda V^*$, with $\Lambda = \mathrm{diag}(\Lambda_+, \Lambda_-)$, be an eigendecomposition, where the diagonal matrices Λ_+ and Λ_- contain the positive and negative eigenvalues, respectively. If there are k positive eigenvalues then

$$\begin{aligned} A &= V\,\mathrm{diag}(\Lambda_+, \Lambda_-)V^* \\ &= V\,\mathrm{diag}(I_k, -I_{n-k})V^* \cdot V\,\mathrm{diag}(\Lambda_+, |\Lambda_-|)V^* \\ &\equiv U_pH. \end{aligned} \tag{3.1}$$

An alternative way to understand (3.1) is to note that the polar decomposition $A = U_pH$ and the matrix sign decomposition [29] $A = (A(A^2)^{-1/2}) \cdot (A^2)^{1/2}$ are equivalent when A is symmetric. We find the polar decomposition interpretation more fruitful as we can then apply QDWH and can also derive an SVD algorithm in a unified fashion (see section 3.6).

Suppose we have computed U_p in (3.1) using the QDWH algorithm, and partition $V = [V_1,\ V_2]$ conformably with Λ. Note that

$$U_p + I = [V_1\ V_2] \begin{bmatrix} I_k & 0 \\ 0 & -I_{n-k} \end{bmatrix} [V_1\ V_2]^* + I = [V_1\ V_2] \begin{bmatrix} 2I_k & 0 \\ 0 & 0 \end{bmatrix} [V_1\ V_2]^* = 2V_1V_1^*,$$

so the symmetric matrix $C = \frac{1}{2}(U_p + I) = V_1V_1^*$ is an orthogonal projector onto $\mathrm{span}(V_1)$, which is the invariant subspace corresponding to the positive eigenvalues. Hence we can compute $\mathrm{span}(V_1)$ by computing an orthogonal basis for the column space of C. One way of doing this is to perform a single QR factorization with pivoting, as originally suggested in [38]. However, pivoting is expensive with regard to communication.

Instead, we use subspace iteration [47, Chap. 14] with $r = \lceil \|C\|_F^2 \rceil$ vectors (in exact arithmetic r is the precise rank of C, since the eigenvalues of C are either 0 or 1). Subspace iteration converges with the convergence factor $|\lambda_{r+1}|/|\lambda_k|$ for the kth eigenvalue, so $\lambda_r = 1$ and $\lambda_{r+1} = 0$ mean a *single* iteration of subspace iteration yields the desired subspace $\mathrm{span}(V_1)$. In practice, due to rounding errors more than one iteration is sometimes needed for subspace iteration with the computed $\widehat{C}$ to converge. If subspace iteration computes the invariant subspaces successfully then the computed matrices satisfy

$$\widehat{C}\widehat{V}_1 = \widehat{V}_1 + \epsilon \quad \text{and} \quad \widehat{C}\widehat{V}_2 = \epsilon. \tag{3.2}$$

Recall that ϵ denotes a matrix (or scalar) of order u whose values differ in different appearances. We provide more details of a practical implementation of subspace iteration in section 5.1.

We now have a matrix $\widehat{V} = [\widehat{V}_1\ \widehat{V}_2]$ such that $\widehat{V}^*A\widehat{V} = \left[\begin{smallmatrix} A_1 & E^* \\ E & A_2 \end{smallmatrix}\right]$. The submatrix E is the backward error of the spectral division, and it is acceptable if $\|E\|_F/\|A\|_F = \epsilon$.

3.2. Algorithm. The entire eigendecomposition can be computed recursively by applying the spectral division of the previous subsection on the submatrices $V_1^*AV_1$ and $V_2^*AV_2$. Algorithm 1 outlines the resulting divide and conquer algorithm (more details will be given in section 5).

ALGORITHM 1. QDWH-eig: compute an eigendecomposition $A = VDV^*$ of a symmetric matrix $A \in \mathbb{R}^{n\times n}$.

```
1 Choose $\sigma$, an estimate of the median of eig($A$).
2 Compute the orthogonal polar factor $U_p$ of $A - \sigma I$ by the QDWH algorithm.
3 Use subspace iteration to compute an orthogonal $V = [V_1\ V_2]$ ($V_1 \in \mathbb{R}^{n\times k}$) such
  that $\frac{1}{2}(U_p + I) = V_1V_1^*$.
4 Compute $A_1 = V_1^*AV_1 \in \mathbb{R}^{k\times k}$ and $A_2 = V_2^*AV_2 \in \mathbb{R}^{(n-k)\times(n-k)}$.
5 Repeat steps 1–4 with $A \leftarrow A_1$ and $A \leftarrow A_2$ until $A$ is diagonalized.
```

The first two steps shift A with the aim of making A_1 and A_2 of the same dimensions, in order to provide for an even division for the recursions; clearly, shifting by the median of the spectrum is the optimal choice. In step 2 we take advantage of the symmetry of the X_k and the triangularity of Q_2 when computing $Q_1Q_2^*$ in (2.3b) to reduce the cost. To smooth the exposition we defer the discussion of practical implementation issues such as the choice of shifts σ to section 5.

3.3. Backward stability proof. Our goal in this section is to prove that QDWH-eig is backward stable, which means that $\widehat{V}$, the computed product of all the V matrices in the recursion, satisfies

$$\widehat{V}^*\widehat{V} = I + \epsilon, \tag{3.3}$$

and $\widehat{V}^*A\widehat{V}$ has off-diagonal elements of order $\epsilon\|A\|_2$. The condition (3.3) is ensured because $\widehat{V}$ is a product of Q factors from Householder QR factorization, each of which is orthogonal to working accuracy [30, p. 360]. That $\widehat{V}^*A\widehat{V}$ is close to diagonal is shown in the proof of the next result.

THEOREM 3.1. *Suppose that* (2.5) (*with* $A \leftarrow A - \sigma I$) *and* (3.2) *are both satisfied throughout the execution of QDWH-eig. Then QDWH-eig is backward stable.*

Proof. It suffices to prove that a single invocation of steps 1–4 of Algorithm 1 computes an invariant subspace of A in a backward stable manner, that is, that $E = \widehat{V}_2^*A\widehat{V}_1$ satisfies $\|E\|_2 = \epsilon\|A\|_2$. From the condition (3.2) on subspace iteration and $\widehat{C} = \frac{1}{2}(\widehat{U}_p + I)$ we have

$$\frac{1}{2}(\widehat{U}_p + I)[\widehat{V}_1\ \widehat{V}_2] = [\widehat{V}_1\ 0] + \epsilon.$$

Right-multiplying by $2\widehat{V}^*$ and using (3.3) gives

$$\begin{aligned}\widehat{U}_p &= 2[\widehat{V}_1\ 0][\widehat{V}_1\ \widehat{V}_2]^* - I + \epsilon = [\widehat{V}_1\ \ -\widehat{V}_2][\widehat{V}_1\ \widehat{V}_2]^* + \epsilon\\ &= \widehat{V}\operatorname{diag}(I_k, -I_{n-k})\widehat{V}^* + \epsilon. \qquad (3.4)\end{aligned}$$

From (2.5) and (3.4) we have

$$A = \big(\widehat{V}\operatorname{diag}(I_k, -I_{n-k})\widehat{V}^* + \epsilon\big)\widehat{H} + \epsilon\|A\|_2 = \widehat{V}\operatorname{diag}(I_k, -I_{n-k})\widehat{V}^*\widehat{H} + \epsilon\|A\|_2, \qquad (3.5)$$

where we used $\|\widehat{H}\|_2 \approx \|A\|_2$, which follows from (2.5). Hence, using (3.3) again, we obtain

$$\widehat{V}^*A\widehat{V} = \operatorname{diag}(I_k, -I_{n-k})\widehat{V}^*\widehat{H}\widehat{V} + \epsilon\|A\|_2.$$

It therefore remains to prove that the off-diagonal blocks X_{21} and X_{12} of $\widehat{V}^*\widehat{H}\widehat{V} = \left[\begin{smallmatrix} X_{11} & X_{12}\\ X_{21} & X_{22}\end{smallmatrix}\right]$ can be expressed as $\epsilon\|A\|_2$. We obtain, using (3.5),

$$\begin{aligned}0 &= A - A^*\\ &= \big(\widehat{V}\operatorname{diag}(I_k, -I_{n-k})\widehat{V}^*\widehat{H} + \epsilon\|A\|_2\big) - \big(\widehat{V}\operatorname{diag}(I_k, -I_{n-k})\widehat{V}^*\widehat{H} + \epsilon\|A\|_2\big)^*\\ &= \widehat{V}\operatorname{diag}(I_k, -I_{n-k})\widehat{V}^*\widehat{H} - \widehat{H}\widehat{V}\operatorname{diag}(I_k, -I_{n-k})\widehat{V}^* + \epsilon\|A\|_2, \qquad (3.6)\end{aligned}$$

since $\widehat{H}^* = \widehat{H}$ by (2.4). Hence, multiplying (3.6) by $\operatorname{diag}(I_k, -I_{n-k})\widehat{V}^*$ on the left and $\widehat{V}$ on the right we get

$$\begin{aligned}\epsilon\|A\|_2 &= \widehat{V}^*\widehat{H}\widehat{V} - \operatorname{diag}(I_k, -I_{n-k})\widehat{V}^*\widehat{H}\widehat{V}\operatorname{diag}(I_k, -I_{n-k})\\ &= \begin{bmatrix} 0 & 2X_{12}\\ 2X_{21} & 0\end{bmatrix}.\end{aligned}$$

Therefore $\|X_{21}\|_2 = \epsilon\|A\|_2$ and $\|X_{12}\|_2 = \epsilon\|A\|_2$, as required. □

The conclusion is that if the polar decomposition algorithm QDWH is backward stable and subspace iteration is successful at every stage of the recursion then QDWH-eig is also backward stable. In [46] we showed that QDWH is backward stable when the QR factorizations are computed with column pivoting and either row pivoting or row sorting and in practice it is stable even without pivoting. As will be discussed in section 5.1, our implementation of subspace iteration succeeds with high probability, and has never failed in our experiments. The upshot is that QDWH-eig is backward stable in practice even without the use of pivoting in the QR factorizations.

3.4. Computing partial eigenpairs. In some applications it is not the whole eigensystem that is required but rather a significant part of it, such as some of the smallest eigenpairs or the eigenpairs corresponding to eigenvalues smaller than a prescribed value τ [4]. In such cases QDWH-eig can easily be adapted to compute only the required part. For example, in the latter case we can first choose σ to be slightly larger than τ and work solely with A_2, as A_2 then contains the entire eigenspace of interest.

3.5. Communication and arithmetic costs. For linear algebra algorithms of $O(n^3)$ complexity (hence excluding Strassen-like algorithms [20], [48]), including matrix multiplication, LU, Cholesky, and QR factorization (without pivoting), asymptotic lower bounds on the communication cost are derived in [12] that can all be expressed as (#arithmetic_operations/$\sqrt{M}$) in the "big-Oh" sense, where M is the size of the fast memory (sequential case) or local memory (parallel case). Algorithms that attain this lower bound are said to minimize communication in the asymptotic sense. Considerable progress has been made in devising linear algebra algorithms that minimize communication, and for the above basic matrix operations implementations are known that asymptotically attain the lower bounds [10], [11], [12], [19].

Since QDWH-eig uses just the computational kernels of QR factorization and matrix multiplication, it asymptotically minimizes communication.

For the arithmetic cost, with advantage taken of the symmetry of X_k and the zero structure in the QR factorization (2.3) we find that QDWH-eig requires about $27n^3$ flops (assuming that each splitting point produces two subproblems of approximately the same size). Details are given in Appendix A.1.

3.6. Comparison with other methods. We now compare QDWH-eig with existing methods. The algorithms of Bai, Demmel, and Gu [9] and Demmel, Dumitriu, and Holtz [18] all have the same basic structure of first mapping the eigenvalues to 0 or ∞ by an implicit repeated squaring of the eigenvalues [9, sect. 4] and then computing an invariant subspace from the resulting matrix. We take for comparison the improved versions by Ballard, Demmel, and Dumitriu [10], which we refer to as IRS (implicit repeated squaring). We will denote the algorithms in [50], [51] by ZZY. In addition to these spectral divide and conquer algorithms, we also compare with standard algorithms based on tridiagonal reduction.

3.6.1. Numerical stability. To the best of our knowledge Theorem 3.1 is the first backward stability proof for a spectral divide and conquer algorithm for symmetric eigenproblems that makes no assumption on the splitting point, which is σ in QDWH-eig. This contrasts with the backward stability analysis for the IRS algorithm in [9, sect. 7], which proves that the backward error is bounded by a factor proportional to $\epsilon\|A\|_F/d^2$, where d is the smallest distance between an eigenvalue of A and the splitting points, which are ± 1 in IRS. Hence the backward error can potentially be large when d is small (indeed we observed instability with IRS in our experiments—see section 6) and, as discussed in [9], IRS's convergence is slow when d is small. Hence, in difficult cases for IRS in which there is an eigenvalue close to a splitting point, IRS is potentially unstable and slow to converge.

QDWH-eig has neither problem, even when A has an eigenvalue close to the splitting point σ, or equivalently when QDWH has to compute the polar decomposition of a nearly singular matrix $A - \sigma I$. This is because QDWH converges within six iterations for any $\kappa_2(A) \leq u^{-1}$, and the backward stability of QDWH is independent of $\kappa_2(A)$ [46].

Another algorithm that minimizes communication is the spectral divide and conquer algorithm proposed by Bai and Demmel [7] that uses the matrix sign function computed by the scaled Newton iteration. Scaling is recommended for both speed and stability, and since the sign function is identical to the unitary polar factor for symmetric matrices we can use the scaled Newton iteration for the polar decomposition [16], [28], [31, p. 205]. The resulting algorithm, which we call SN-eig, requires explicit matrix inversion as the major computational kernel and can be implemented in a communication-minimizing manner by using a communication-avoiding LU factorization [26], [37]. By the same argument as Theorem 3.1, the backward stability of SN-eig rests on that of the polar decomposition computed by the scaled Newton iteration, and it is shown in [46, sect. 6.1] that the computed polar decomposition is backward stable provided that the matrix inverses are computed in a mixed forward-backward stable manner. Unfortunately the standard inversion method based on LU factorization with partial pivoting does not guarantee this property (a bidiagonalization-based inversion does, but this is much too expensive and does not minimize communication). Indeed our numerical experiments (see section 6.1) suggest that the backward error of SN-eig is not as small as QDWH-eig.

For the spectral divide and conquer algorithms proposed in [50], [51], backward error analysis is not provided.

As is well known, standard algorithms for the symmetric eigenvalue problem are based on orthogonal transformation and hence are backward stable. However, our experiments demonstrate that QDWH-eig tends to give smaller backward errors.

3.6.2. Cost. Table 3.1 compares QDWH-eig, SN-eig, IRS, ZZY, and the standard algorithm that performs tridiagonalization followed by the symmetric tridiagonal QR algorithm [25] for computing the complete eigenvalue decomposition (eigenvalues and eigenvectors) of an $n \times n$ symmetric matrix. It summarizes whether the algorithm minimizes communication, the existence of a backward stability proof, the theoretical maximum iteration count (for computing the polar decomposition (or disk function in IRS [9])) needed for one recursion of spectral divide and conquer (not the iteration count of subspace iteration, which is always less than three), the typical arithmetic cost in flops, the maximum matrix dimension involved in the computations, and the memory requirement. For ZZY the flop count and maximum iteration are those of algorithm QUAD in [51]; the other algorithms in [50], [51] behave similarly.

The maximum number of iterations shows the case in which there is an eigenvalue within distance u of the splitting points, which is $-\log_2 u = 53$ iterations for IRS and ZZY. In general, IRS and ZZY require increasingly more iterations for ill conditioned problems. The arithmetic costs of QDWH-eig, SN-eig, IRS, and ZZY are obtained under the assumption that at each recursion stage the eigenvalues are at least 10^{-5} from the splitting points. Then QDWH-eig needs five iterations, SN-eig needs seven iterations, and IRS and ZZY need 22 iterations, and for each iteration these methods need $5n^3$ or $(3+\frac{1}{3})n^3$ flops (see the appendix), n^3 flops (taking advantage of symmetry in matrix inversions), $(13+\frac{1}{3})n^3$ flops, and $7n^3$ flops, respectively. This assumption is justified because (as in QDWH-eig) we can use a condition number estimator to obtain a lower bound for the smallest singular value, so we can change the shift σ if $\kappa_2(A-\sigma I) \gg 10^5$. In the "numerically worst case" $\kappa_2(A-\sigma I) \approx u^{-1}$, the arithmetic costs of QDWH-eig, SN-eig, IRS, and ZZY become $33n^3$, $29n^3$, $720n^3$, and $370n^3$, respectively.

Table 3.1 illustrates a general picture that IRS and ZZY always need significantly more arithmetic than QDWH-eig, which is reflected in our numerical experiments in section 6. The large difference between the cost of QDWH-eig and IRS is due to the difference in the required number of iterations and the fact that QDWH-eig employs the "thin" QR factorization whereas IRS requires the full QR factorization.

The spectral divide and conquer algorithms QDWH-eig, SN-eig, IRS, and ZZY can all be implemented in a communication-minimizing manner (for ZZY we need to replace QR with pivoting by subspace iteration or the randomized algorithm suggested in [10]). We note that by requiring a smaller number of iterations than IRS and ZZY, QDWH-eig will have a lower communication cost than them, although in the asymptotic argument this effect is absorbed as a constant.

To summarize, the cost of QDWH-eig is smaller than that of IRS and ZZY by a large constant factor, in both arithmetic and communication. Note, however, that IRS is applicable in more general settings, namely the generalized eigenproblem and

Table 3.1
Comparison of algorithms for symmetric eigendecomposition.

	QDWH-eig	SN-eig	IRS [10]	ZZY [50], [51]	Standard
Minimizes communication?	$\surd$	$\surd$	$\surd$	$\surd$	$\times$
Backward stability proof?	$\surd$	$(\surd)$	conditional	none	$\surd$
Max. # iterations	6	9	53	53	
Arithmetic cost	$27n^3$	$(15+\frac{2}{9})n^3$	$\approx 300n^3$	$\approx 160n^3$	$9n^3$
Matrix dimension	$2n \times n$	$n \times n$	$2n \times n^a$	$2n \times n$	$n \times n$
Memory	$4n^2$	$4n^2$	$6n^2$	$6n^2$	$3n^2$

^{a}The IRS algorithm requires $2n \times 2n$ orthogonal QR factors Q, but only the last n columns of Q are explicitly required.

the nonsymmetric eigenproblem. The costs of QDWH-eig and SN-eig are comparable, but we shall see through numerical experiments that the practical stability of SN-eig is not as good as QDWH-eig.

Compared with the standard tridiagonalization-based algorithm, QDWH-eig can be implemented in a communication-minimizing manner, whereas it is not known how to perform tridiagonalization with minimal communication; an implementation that minimizes the bandwidth cost is given in [12, sect. 6], but it does not minimize the latency cost. The arithmetic cost of QDWH-eig is higher than the standard algorithm by a factor at most 3.

4. QDWH-SVD. Higham and Papadimitriou [32], [33] propose a framework for computing the SVD via the polar decomposition and the eigendecomposition: the polar decomposition $A = U_pH$ and the symmetric eigendecomposition $H = V\Sigma V^*$ are computed and the SVD is obtained from $A = (U_pV)\Sigma V^*$. With parallel computing in mind, they suggest using a method based on Padé iteration for the polar decomposition and any standard method for the symmetric eigendecomposition.

Our SVD algorithm QDWH-SVD, summarized in Algorithm 2, follows this path but replaces both steps with QDWH-based algorithms: it computes the polar decomposition by the QDWH algorithm and the symmetric eigendecomposition by QDWH-eig. Without loss of generality we assume $m \geq n$ because when $m < n$ we can compute the SVD $A^* = U\Sigma V^*$, in terms of which the SVD of A is just $V\Sigma U^*$.

Algorithm 2. QDWH-SVD: compute the SVD $A = U\Sigma V^*$ of $A \in \mathbb{R}^{m\times n}$ with $m \geq n$.

```
1  Compute the polar decomposition $A = U_pH$ via QDWH.
2  Compute the symmetric eigendecomposition $H = V\Sigma V^*$ via QDWH-eig.
3  Form $U = U_pV$.
```

It is well known that in the standard SVD methods of bidiagonalization followed by the QR algorithm or dqds the flop count is minimized by performing an initial QR factorization when $m \gg n$ and then computing the SVD of the R factor [2, sect. 2.4.6], and the threshold is around $m \approx 2n$ [17]. We find (see Appendix A.2) that to minimize the flop count of QDWH-SVD we should compute an initial QR factorization when $m > 1.15n$.

4.1. Backward stability. Higham and Papadimitriou [32] show that the SVD computed via their framework is backward stable provided that both the polar decomposition and the eigendecomposition are computed in a backward stable manner.

TABLE 4.1
Comparison of algorithms for the SVD.

	QDWH-SVD	SN-SVD	IRS for SVD [10]	Standard
Minimize communication?	✓	✓	✓	×
Backward stability proof	✓	(✓)	conditional	✓
Max. # iterations	6	9	53	
Arithmetic cost ($m = n$)	$35n^3$–$52n^3$	$21n^3$–$35n^3$	$\approx 2400n^3$	$22n^3$
Matrix dimension	$(m+n) \times n$	$n \times n$	$2(m+n) \times (m+n)$	$n \times n$
Memory ($m = n$)	$5n^2$	$5n^2$	$24n^2$	$4n^2$

Combining this result with Theorem 3.1 we obtain a backward stability result for QWDH-SVD.

THEOREM 4.1. *Suppose that in QDWH-SVD the polar decomposition computed by QDWH in step* 1 *satisfies* (2.5) *and that the conditions of Theorem 3.1 are satisfied for step* 2. *Then QDWH-SVD is backward stable, that is,* $\|A - \widehat{U}\widehat{\Sigma}\widehat{V}^*\|_F = \epsilon\|A\|_F$, $\|\widehat{U}^*\widehat{U} - I\|_F = \epsilon$, *and* $\|\widehat{V}^*\widehat{V} - I\|_F = \epsilon$.

Combining the theorem with the analysis in [46] we conclude that QDWH-SVD is backward stable if column pivoting and either row pivoting or row sorting are used when computing the QR factorizations, and in practice is backward stable even without the use of pivoting.

4.2. Communication and arithmetic costs. QDWH-SVD minimizes communication costs in the asymptotic sense because, just like QDWH-eig, it uses only QR factorizations (with pivoting not needed in practice) and matrix multiplications.

The arithmetic cost is essentially the sum of the costs of QDWH and QDWH-eig and for $n \times n$ A ranges from $35n^3$ flops to $52n^3$ flops depending on $\kappa_2(A)$. Thus the cost of computing the SVD of a general square matrix is less than twice that of computing an eigendecomposition of a symmetric matrix of the same size by QDWH-eig.

4.3. Comparison with other methods. As described in [10], IRS for the symmetric eigenproblem can be used to compute the SVD by computing the eigendecomposition of $\left[\begin{smallmatrix} 0 & A^* \\ A & 0 \end{smallmatrix}\right]$ (we simply call this algorithm IRS). This algorithm still minimizes communication in the asymptotic sense. However, since this way of computing the SVD costs about eight times as much as an eigendecomposition, the QDWH-SVD approach of first computing the polar decomposition is generally much more efficient.

Table 4.1 compares four algorithms for computing the whole SVD (singular values and singular vectors): QDWH-SVD, SN-SVD (obtained by computing the polar decomposition in QDWH-SVD by the scaled Newton iteration instead of QDWH, and hence applicable only to square nonsingular matrices), IRS, and the standard algorithm that performs bidiagonalization followed by bidiagonal QR. QDWH-SVD and SN-SVD require more than 50 times fewer flops than the communication-minimizing algorithm IRS (the costs per iteration are $5n^3$ or $(3 + \frac{1}{3})n^3$, $2n^3$ or n^3, and $107n^3$, respectively). The difference between the QDWH-based approach and IRS is even more pronounced than for the eigendecomposition, because for the SVD, IRS iterates with an $(m + n) \times (m + n)$ matrix, which soon becomes dense as the iteration proceeds. QDWH-SVD has an arithmetic cost at most a factor 2.4 larger than the bidiagonalization method, and no communication minimizing implementation of the latter is available.

5. Practical considerations. We now consider detailed implementation issues for QDWH-eig and QDWH-SVD.

5.1. Implementing subspace iteration. Algorithm 3 gives pseudocode for subspace iteration for computing the column space of $C = \frac{1}{2}(U_p + I)$.

ALGORITHM 3. Subspace iteration: given symmetric $A, C = \frac{1}{2}(U_p + I) \in \mathbb{R}^{n\times n}$, where $C = V_1V_1^*$ is the orthogonal projection onto an invariant subspace of A and $k = \|C\|_F^2$, compute $V_1 \in \mathbb{R}^{n\times k}$, $V_2 \in \mathbb{R}^{n\times(n-k)}$ such that $[V_1\ V_2]$ is orthogonal.

```
1  Choose initial matrix $X \in \mathbb{R}^{n\times k}$.
2  Compute QR factorization $X = [V_1\ V_2]\left[\begin{smallmatrix} R \\ 0 \end{smallmatrix}\right]$.
3  Form $E = V_2^* A V_1$.
4  If $\|E\|_F/\|A\|_F = \epsilon$, quit, end
5  $X := CV_1$, go to step 2.
```

Since a good initial matrix X is one whose columns lie nearly in the column space of C, a practical and efficient choice is the set of $k + \widetilde{k}$ columns of C of largest norms, where $k = \lceil \|C\|_F^2 \rceil$ and $\widetilde{k}$ is a small constant used as a safeguard; in our experiments we used $\widetilde{k} = 3$. This choice generally works well and saves one matrix multiplication compared with generating a random X and forming CX before taking the QR factorization; in fact we have $CX = X$ with our choice of X because C is an orthogonal projection.

A reliable approach is to run subspace iteration until (3.2) is satisfied, which guarantees $\|E\|_F = \epsilon\|A\|_F$ (see Theorem 3.1). However, verifying (3.2) requires an extra matrix multiplication and, as noted in section 3.1, typically subspace iteration converges in just one step, so we find it more efficient to immediately form E in $[V_1\ V_2]^* A [V_1\ V_2] = \left[\begin{smallmatrix} A_1 & E^* \\ E & A_2 \end{smallmatrix}\right]$ and check the actual backward error, which we deem acceptable if $\|E\|_F \le 10u\|A\|_F$.

In finite precision arithmetic the eigenvalues are not exactly 0 or 1, so subspace iteration may not yield $\|E\|_F = \epsilon\|A\|_F$ in just one iteration. In all our experiments two subspace iterations were enough to either successfully yield an invariant subspace to working accuracy, occasionally achieving smaller $\|E\|_F$ than with one iteration, or reach stagnation, in which case further subspace iteration does not further reduce $\|E\|_F$. The latter case, although exceptionally rare, can happen in two unlikely events:

(i) The initial matrix X is nearly orthogonal to a subspace of C.

(ii) C has eigenvalues far from both 0 and 1.

Case (i) can be remedied by rerunning subspace iteration using a different initial matrix X, an effective choice of which is a randomized matrix $X = CW_1$, where W_1 is the first k columns of a random Haar distributed orthogonal matrix $W = [W_1\ W_2]$. As shown in [18, sect. 5], with high probability the QR factorization $CW = [X\ CW_2] = [V_1\ V_2]R$ is rank-revealing, in which case V_1 represents the column space of both C and X. Hence by computing the QR factorization of X we can obtain $[\widehat{V}_1\ \widehat{V}_2]$ such that (3.2) is satisfied. Therefore subspace iteration terminates successfully with high probability.

Case (ii) indicates that the $\widehat{U}_p$ computed by QDWH failed to be unitary to working accuracy. This can happen if the smallest singular value of X_0 is severely overestimated, that is, $\ell_0 \gg \sigma_{\min}(X_0)$ (corresponding to failure of a condition estimator—see section 5.8). If (ii) happens, then we can either run QDWH again to compute the unitary polar factor of $\widehat{U}_p$ (which is identical to, but cheaper to compute than, that of $A - \sigma I$, as the singular values of $\widehat{U}_p$ are much closer to 1) or choose a different σ and rerun steps 1–4 of QDWH-eig. We have never had to resort to these remedies in our experiments.

5.2. Choice of splitting point σ in QDWH-eig. The ideal splitting point σ is the median of eig(A), so that one recursion of QDWH-eig results in matrices A_1 and A_2 both of order $\approx n/2$. To estimate the median in our experiments we used the median of diag(A). Since this quantity lies in $[\lambda_n(A), \lambda_1(A)]$ it produces matrices A_1 and A_2 both with dimension at least one. The authors of IRS [10] suggest choosing σ to be around the center of an interval containing the eigenvalues (obtained, e.g., via Gershgorin's theorem). We prefer the median of diag(A) because when A is close to diagonal form it ensures $\dim(A_1) \approx \dim(A_2)$. Another strategy of complexity $O(n^2)$ is to compute the eigenvalues of the tridiagonal part of A and then take their median. All these choices work reasonably well, but it is possible to contrive examples in which the size of A_1 or A_2 is much smaller than $n/2$. It is an open question to develop a better splitting strategy.

We note that, as mentioned in [10, sect. 3.4], spectral divide and conquer type algorithms such as QDWH-eig for symmetric eigendecompositions deal effectively with multiple (or clusters of) eigenvalues. This is because if a diagonal block A_j has all its eigenvalues lying in $[\lambda_0 - \epsilon, \lambda_0 + \epsilon]$, then A_j must be nearly diagonal: $A_j = \lambda_0 I + \epsilon$. Upon detecting such a submatrix A_j, in QDWH-eig we stop the spectral divide and conquer recursion on A_j and return the value of the (nearly) multiple eigenvalue λ_0 and its multiplicity, along with $V_j = I$, the (numerical) matrix of eigenvectors of A_j.

5.3. When to terminate the recursion. In a practical implementation of QDWH-eig we can stop the recursion (step 5) once the diagonal blocks are small enough so that any standard eigensolver can be invoked to finish the diagonalization efficiently. However, for large ($n \geq 2000$) matrices A the typical runtime spent during this final stage of QDWH-eig is negligibly small, and we found no noticeable difference in speed if we carry on the recursion in step 5 of QDWH-eig to the scalar level. Our experiments in section 6 ran the recursion until the end.

5.4. When σ is equal to an eigenvalue. Even when σ is equal to an eigenvalue of A in QDWH-eig, the QDWH iteration for computing the polar decomposition of $A-\sigma I$ does not break down (unlike the scaled Newton iteration used in SN-eig, which requires the matrix to be nonsingular) but rather computes the partial isometry U_p in the canonical polar decomposition of A [31, p. 194]. In terms of the QDWH-eig execution this causes little problem, because in this case the matrix $\frac{1}{2}(U_p + I)$ has eigenvalues 1, 0, or 0.5. Subspace iteration has no difficulty finding an invariant subspace V_1 corresponding to the eigenvalues 1 and 0.5. V_1 is then an invariant subspace corresponding to the nonnegative (including zero) eigenvalues of $A - \sigma I$.

In practice, this situation rarely arises, because rounding errors usually cause the zero singular values to be perturbed to a small positive value, and QDWH eventually maps them to 1, in which case the singularity of $A - \sigma I$ is treated unexceptionally by QDWH.

5.5. QDWH-SVD for rank-deficient matrices. When A has rank $r < n$, QDWH with the choice $(u \ll)\ell_0 \leq \sigma_r(X_0)$ computes $A = U_p H$ where U_p is a partial isometry, which does not have orthonormal columns. However, H is still the unique Hermitian polar factor of A, and it has the same number of zero singular values as A. If the computed eigendecomposition is $H = [V_1\ V_2]\,\mathrm{diag}(\Sigma_r, 0_{n-r})[V_1\ V_2]^*$ then we obtain $A = (U_p V_1)\Sigma_r V_1^*$, where $U_p V_1$ can be shown to have orthonormal columns, and this is hence the "economy" SVD. If a full SVD with $U \in \mathbb{R}^{m\times n}, V \in \mathbb{R}^{n\times n}$ is desired we can compute a full QR factorization $U_p[V_1\ V_2] = QR$, where $Q \in \mathbb{R}^{m\times n}$ and $r_{ii} \geq 0$ for all i, and then set $U := Q$, $V := [V_1\ V_2]$.

In practice we may not know in advance that the matrix is rank-deficient or have a good estimate of the smallest positive singular value $\sigma_r(X_0)$. In such cases we can run QDWH-SVD as usual by estimating $\ell_0 \approx \sigma_{\min}(X_0)$; the condition number estimator gives $\kappa_2(A) \approx u^{-1}$ so we let $\ell_0 \approx u$. This corresponds to mapping the zero singular values (perturbed to $O(u)$ by rounding errors) to 1. This approach is slower than the one above that uses $u \ll \ell_0 \leq \sigma_r(X_0)$ because QDWH requires more iterations.

When A is (nearly) rank-deficient, the computed singular values $\widehat{\sigma}_i$ from QDWH-SVD are not guaranteed to be nonnegative. However, since the algorithm is backward stable the $\widehat{\sigma}_i$ are the singular values of $A + \Delta A$ with $\|\Delta A\|_2 = \epsilon\|A\|_2$. Hence any negative $\widehat{\sigma}_i$ are necessarily of order $\epsilon\|A\|_2$ and can safely be set to zero or replaced by their absolute values, as suggested in [32].

5.6. Faster QDWH iterations. The QDWH iteration (2.3) is mathematically equivalent to (2.2), which after some rewriting can be implemented according to

$$Z_k = I + c_k X_k^* X_k, \quad W_k = \mathrm{chol}(Z_k), \tag{5.1a}$$

$$X_{k+1} = \frac{b_k}{c_k} X_k + \left(a_k - \frac{b_k}{c_k}\right)(X_k W_k^{-1}) W_k^{-*}, \tag{5.1b}$$

where $\mathrm{chol}(Z_k)$ denotes the Cholesky factor of Z_k. Note that (5.1) involves $m \times n$ matrices, in contrast to (2.3), which contains an $(m+n)\times n$ matrix. The arithmetic cost of (5.1) is mn^2 flops for forming the symmetric positive definite matrix Z_k, $n^3/3$ flops for computing its Cholesky factorization, and $2mn^2$ flops for two multiple right-hand side triangular substitutions. Therefore this implementation requires $3mn^2 + n^3/3$ flops, which is cheaper than evaluating (2.3) by Householder QR factorization and exploiting the structure of $\left[\begin{smallmatrix}\sqrt{c_k}X_k\\ I\end{smallmatrix}\right]$, which needs $5mn^2$ flops (see Appendix A.1). Furthermore, the Cholesky decomposition and triangular substitution both have a known arithmetic and communication-minimizing implementation [11], [12]. Thus (5.1) is expected to be faster than (2.3).

The numerical stability of (5.1), however, is compromised if Z_k is ill conditioned, as standard error analysis shows that X_{k+1} has errors of order $\kappa_2(Z)\epsilon$ [30]. The implementation (2.3) is also subject to errors, arising from a QR factorization of the matrix $\left[\begin{smallmatrix}\sqrt{c_k}X_k\\ I\end{smallmatrix}\right]$, but this matrix has 2-norm condition number equal to the square root of that of Z_k, and in any case numerical stability of the polar decomposition computed by (2.3) is independent of $\kappa_2(Z_k)$, as shown in [46].

To investigate (5.1) further, we note that $\kappa_2(Z) \leq 1 + c_k\|X_k\|_2^2$ and $\|X_k\|_2 \leq 1$ (provided that $\alpha \geq \|A\|_2$), and moreover b_k/c_k and $a_k - b_k/c_k$ are both of order 1 for all k [46]. Hence X_{k+1} is computed from (5.1) with forward error bounded by $c_k\epsilon$. Fortunately, c_k converges to 3 and the convergence is fast enough so that $c_k \leq 100$ for $k \geq 2$ for any practical choice $\ell_0 > 10^{-16}$. In our experiments we switch from (2.3) to (5.1) once c_k is smaller than 100, which improves the speed and also in practice slightly improves the stability. In particular, if $\ell_0 > 10^{-5}$ then we have $c_k \leq 100$ for $k \geq 1$ (for a given k, c_k is a decreasing function of ℓ_0), so we need just one iteration of (2.3).

We note that we can, alternatively, directly evaluate (2.2). The resulting algorithm DWH is, unfortunately, numerically unstable, as observed in [45].

5.7. Newton–Schulz postprocessing for improved orthogonality. In practice the computed orthogonal factors $\widehat{U}$ and $\widehat{V}$ are not exactly orthogonal. As mentioned in [46, sect. 6.3], an effective way to reduce the distance to orthogonality is to run one step of the Newton–Schulz iteration, so that we compute $\widehat{U} \leftarrow \frac{1}{2}\widehat{U}(3I - \widehat{U}^*\widehat{U})$, $\widehat{V} \leftarrow \frac{1}{2}\widehat{V}(3I - \widehat{V}^*\widehat{V})$. Experiments suggest that this postprocessing tends to improve the distance to orthogonality considerably (by about a factor 4 for large dimensions) and also slightly improves the backward errors of the eigendecomposition or SVD.

5.8. Estimating α and ℓ. When running the QDWH iterations (2.3), we obtain the estimates $\alpha \gtrsim \|A\|_2$ and $\ell \lesssim \sigma_{\min}(X_0)$ using the MATLAB assignments

$$\alpha = \texttt{normest}(A), \quad \ell_0 = 0.9/\texttt{condest}(X_0), \tag{5.2}$$

where the factor 0.9 attempts to make it more likely that $\ell_0 \le \sigma_{\min}(X_0)$. The MATLAB function `normest` estimates the 2-norm of its matrix argument using the power method, while `condest` estimates the 1-norm condition number using the algorithm of [34], producing a lower bound.

The QDWH iteration maps all the singular values of X_0 in $[\ell_0, 1]$ to 1. If there exists a singular value $\sigma_i(X_0)$ with $\sigma_i(X_0) < \ell_0$, one can show that QDWH maps it within $k \le 6$ iterations to a value $\sigma_i(X_k) \in [\sigma_i(X_0)/\ell_0, 1]$, which is generally much closer to 1 than $\sigma_i(X_0)$. Then p (a few) additional QDWH iterations, now equivalent to Halley's iteration, are sufficient to yield $|\sigma_i(X_{k+p}) - 1| \le u$, as can be seen from the convergence $x_p \to 1$ of the scalar iteration $x_{p+1} = x_p(3 + x_p^2)/(1 + 3x_p^2)$, $x_0 = \sigma_i(X_k)$. A similar comment applies to the estimation of α. Hence wrong estimates of α and ℓ_0 can cause QDWH to require one or two additional iterations (more if $\ell_0 \gg \sigma_{\min}(X_0)$ or $\alpha \ll \|A\|_2$), but not instability or misconvergence, so rough estimates that are accurate to a factor of (say) 5 are completely acceptable.

Note that we can use $\ell_0 = \|X_0\|_1/(n^{1/2}\texttt{condest}(X_0))$ if we wish to guarantee $\ell_0 \lesssim \sigma_{\min}(X_0)$. To see this, note that

$$\texttt{condest}(X_0) \approx \|X_0\|_1\|X_0^{-1}\|_1 \ge \|X_0\|_1 n^{-1/2}\|X_0^{-1}\|_2 = \|X_0\|_1 n^{-1/2}\sigma_n^{-1},$$

which implies $\sigma_n \gtrsim \|X_0\|_1/(n^{1/2}\texttt{condest}(X_0))$. We can also safely use $\alpha = \|A\|_F$ to guarantee $\alpha \ge \|A\|_2$. We used the estimates (5.2) because they have proved reliable in practice.

Algorithms SN-eig and SN-SVD are obtained simply by replacing QDWH with the scaled Newton iteration with the scaling of [16], with extremal singular values estimated as just described.

6. Numerical experiments. We now present some numerical experiments. Their purpose is threefold.

- To compare the new algorithms with existing spectral divide and conquer algorithms.
- To compare the numerical stability of the new algorithms with that of existing algorithms for the symmetric eigendecomposition and the SVD.
- To check that when programmed in MATLAB on a multicore machine our algorithms run with speed within a small constant factor of that of the best current algorithms, as our analysis suggests should be the case.

We do not attempt to test a communication-minimizing implementation of the algorithms on a highly parallel machine; this will be the subject of future work.

All the experiments were carried out in MATLAB version R2012a on a machine with an Intel Core i7 3.40GHz processor with four cores, eight threads, and 16GB RAM, using IEEE double precision arithmetic. In implementing QDWH-eig we used the multithreaded built-in MATLAB functions: the QR factorizations are computed using `qr`, which calls LAPACK routines DGEQRF and DORGQR (which do not minimize communication), and for the Cholesky factorization we used MATLAB function `chol`, which calls DPBTRF. Similarly, for SN-eig we used MATLAB function `inv` for computing explicit matrix inverses.

6.1. Symmetric eigendecomposition.

6.1.1. Spectral divide and conquer algorithms. We first compare spectral divide and conquer type algorithms for computing an invariant subspace corresponding to the positive/negative eigenvalues of a symmetric matrix.

The algorithms we compare are QDWH-eig (with no pivoting in the QR factorizations), SN-eig, IRS [10], and ZZY (the algorithm QUAD in [51]—the behavior of other algorithms proposed in [50], [51] was similar). We also test QDWH-eig with QR factorizations computed with row sorting and column pivoting, denoted "QDWH-eig(p)."

We compare how the methods behave for problems of varying difficulties. Our experiments were carried out as follows. With $n = 100$ we generated $n \times n$ symmetric matrices $A = V\Lambda V^* \in \mathbb{R}^{n\times n}$, where we used the MATLAB function `gallery('qmult',...)` to obtain a random Haar distributed orthogonal matrix V, after seeding the random number generator via `rng(10)`. Λ is a diagonal matrix whose diagonal entries form a geometric series $1, r, r^2, \ldots$, with ratio $r = -\kappa^{-1/(n-1)}$, where $\kappa = \kappa_2(A)$ is the prescribed condition number. The smallest eigenvalue of A is κ^{-1}.

Here we consider computing an invariant subspace corresponding to the positive eigenvalues of A. To do this, we can apply QDWH-eig and SN-eig to A with shift $\sigma = 0$. IRS and ZZY compute invariant subspaces corresponding to eigenvalues inside and outside the interval $(-1, 1)$, so we apply them to the shifted matrix $A - I$, whose invariant subspace corresponding to the eigenvalues in $(-1, 1)$ matches the desired one. Spectral divide and conquer algorithms generally face difficulty when the matrix has an eigenvalue close to the splitting points (0 for QDWH and SN, and ± 1 for IRS and ZZY), so in our setting $\kappa_2(A)$ is an appropriate measure of the problem's difficulty. By focusing on a single invariant subspace calculation (instead of the whole eigendecomposition) we can directly study the effect of varying the difficulty of the splitting problem.

We generated 100 matrices for each of $\kappa = 10^2, 10^8, 10^{15}$, and report the maximum and minimum values of the iteration counts (for mapping the eigenvalues numerically to ± 1 or $(0, \infty)$), shown as "iter" in Table 6.1, along with the backward error $\|E\|_F/\|A\|_F$ where $[\widehat{V}_1\ \widehat{V}_2]^* A[\widehat{V}_1\ \widehat{V}_2] = \left[\begin{smallmatrix} A_1 & E^* \\ E & A_2 \end{smallmatrix}\right]$, shown as "berr," where $\widehat{V}_1$ is the computed $n \times k$ matrix with orthonormal columns. We obtained $k = 50 = \frac{n}{2}$ in all cases and verified that all the eigenvalues of $\widehat{V}_1^* A\widehat{V}_1$ are larger than $-u$ ("numerically nonnegative"), indicating that the computed $\widehat{V}_1$ indeed approximates the positive eigenspace. Also, the orthogonality measure $\|\widehat{V}_1^*\widehat{V}_1 - I_k\|_F/\sqrt{n}$ was of order u for all the methods.

We see from Table 6.1 that QDWH-eig converged within six iterations in every case, whereas ZZY and IRS needed many more iterations, especially in the difficult cases where $\kappa_2(A)$ is large. Furthermore, QDWH-eig performed in a backward stable way throughout (recall that Theorem 3.1 combined with [46] proves backward stability of QDWH-eig when pivoting is employed). SN-eig required about 50% more iterations than QDWH-eig and gave slightly larger backward errors. IRS lost backward stability when $\kappa_2(A)$ was large, which reflects the backward error bound given in [9]. ZZY

Table 6.1
Results for computing an invariant subspace.

$\kappa_2(A)$		10^2		10^8		10^{15}	
		min	max	min	max	min	max
iter	QDWH-eig(p)	4	5	5	6	6	6
	QDWH-eig	4	5	5	6	6	6
	SN-eig	7	8	8	9	9	10
	ZZY	12	12	32	32	55	55
	IRS	11	11	31	31	53	54
berr	QDWH-eig(p)	4.4e-16	5.2e-16	4.4e-16	5.3e-16	4.7e-16	6.3e-16
	QDWH-eig	4.3e-16	5.1e-16	4.5e-16	5.3e-16	4.9e-16	6.2e-16
	SN-eig	1.4e-15	2.0e-15	1.3e-15	1.8e-15	1.2e-15	2.1e-15
	ZZY	1.3e-15	1.6e-15	1.8e-15	2.4e-15	2.2e-15	3.1e-15
	IRS	2.0e-15	1.8e-12	4.1e-13	8.3e-11	1.2e-9	6.5e-6

performed in a backward stable manner, but no error analysis is available for it.

The experiment demonstrates that QDWH-eig is clearly superior to IRS and ZZY for computing an invariant subspace, and hence for the whole eigendecomposition via a spectral divide and conquer process. Pivoting does not seem to be necessary in practice for QDWH-eig, so below we focus on QDWH-eig without pivoting (and SN-eig).

6.1.2. Comparison with conventional algorithms for the full symmetric eigendecomposition. We now compare QDWH-eig with several algorithms used in practice for computing the full eigendecomposition of a symmetric matrix, all of which begin by reducing the matrix to tridiagonal form.

- The divide and conquer algorithm [27] (not to be confused with the spectral divide and conquer algorithms that include QDWH-eig, IRS, and ZZY), denoted "D-C" (LAPACK routine DSPEVD, called via the NAG MATLAB Toolbox (Mark 23) function `f08fc.m` [44]).
- The built-in MATLAB function `eig`.
- The QR algorithm [47, Chap. 8], denoted "QR" (LAPACK routine DSYEV, called via `f08fa.m`).
- The solver based on multiple relatively robust representations [22], denoted "MR^3" (LAPACK routine DSYEVR, called via `f08fd.m`).

As we shall see (and as is known to experts [43]), the performance of these algorithms can differ significantly.

We generated symmetric matrices using the same $A = V\Lambda V^*$ prescription as in section 6.1.1 but with the eigenvalues uniformly distributed in $[0, 1]$.

Table 6.2 shows the backward error $\|\widehat{V}\widehat{\Lambda}\widehat{V}^* - A\|_F/\|A\|_F$, where $\widehat{\Lambda}$ and $\widehat{V}$ are the matrices of computed eigenvalues and eigenvectors, respectively. "QDWH-eig" includes Newton–Schulz postprocessing, while "QDWH-eig, no NS" is QDWH-eig without the Newton–Schulz postprocessing. The backward stability is acceptable for all the methods with the exception of SN-eig. It is worth noting, however, that the backward errors of QDWH-eig are smaller than those of the other methods, by more than a factor 3 for large n.

Table 6.3 shows the normalized measure of orthogonality of the computed $\widehat{V}$ (we note that $\|X^TX - I\|_F$ is within a factor 2 of the distance to orthogonality $\min\{\, \|X - Q\|_F : Q^TQ = I\,\}$ if $\|X\|_2 \approx 1$ [31, Lem. 8.17]). While all the methods yielded values $\|\widehat{V}^*\widehat{V} - I\|_F/\sqrt{n}$ that are moderate multiples of u, we see that those of QDWH-eig are much smaller than those for the other algorithms. The Newton–Schulz

Table 6.2
Backward error $\|A - \widehat{V}\widehat{\Lambda}\widehat{V}^*\|_F/\|A\|_F$.

n	2000	4000	6000	8000	10000
QDWH-eig	2.1e-15	2.4e-15	2.6e-15	2.9e-15	3.0e-15
QDWH-eig, no NS	3.6e-15	4.3e-15	4.8e-15	5.2e-15	5.5e-15
SN-eig	1.3e-13	4.1e-13	8.2e-13	1.3e-12	2.0e-12
D-C	5.2e-15	7.1e-15	8.7e-15	9.8e-15	1.1e-14
`eig`	1.3e-14	1.9e-14	2.3e-14	2.6e-14	2.9e-14
QR	1.5e-14	2.8e-14	4.0e-14	2.9e-14	5.4e-14
MR^3	4.2e-15	5.7e-15	7.1e-15	8.0e-15	9.0e-15

Table 6.3
Orthogonality of $\widehat{V}$: $\|\widehat{V}^*\widehat{V} - I\|_F/\sqrt{n}$.

n	2000	4000	6000	8000	10000
QDWH-eig	7.7e-16	8.0e-16	8.2e-16	8.4e-16	8.5e-16
QDWH-eig, no NS	2.4e-15	2.8e-15	3.0e-15	3.3e-15	3.4e-15
SN-eig	2.5e-15	2.9e-15	3.2e-15	3.4e-15	3.6e-15
D-C	4.5e-15	6.3e-15	7.6e-15	8.6e-15	9.8e-15
`eig`	1.1e-14	1.5e-14	1.9e-14	2.1e-14	2.4e-14
QR	1.1e-14	2.3e-14	3.6e-14	2.1e-14	5.1e-14
MR^3	2.6e-15	3.4e-15	4.2e-15	4.7e-15	5.3e-15

postprocessing usually improves the orthogonality considerably and also slightly reduces the backward error.

This behavior was not peculiar to this class of matrices, and was observed in all our experiments. These results suggest that QDWH-eig might have better stability than the other algorithms, which merits further investigation.

On our machine with four cores, D-C (and `eig`, which had similar speed) was the fastest algorithm in all cases, on average being faster than QR, QDWH-eig, SN-eig, and MR^3 by factors about 2, 3.5, 5, and 6, respectively.

6.2. SVD algorithms. We now compare algorithms for computing the SVD. We generate matrices by using the MATLAB function `gallery('randsvd',...)` to form $A = U\Sigma V^*$, where U and V are random Haar distributed orthogonal matrices and Σ is a diagonal matrix of singular values that form an arithmetic sequence.

6.2.1. Nonsingular case. We compare[1] QDWH-SVD, SN-SVD, D-C (LAPACK routine DGESDD, called via the NAG MATLAB Toolbox function `f08kd.m`), the MATLAB `svd` function, and the QR algorithm (LAPACK routine DGESVD, called via `f08ke.m`).

Varying matrix sizes. We set $\kappa_2(A) = 1.5$ and varied the matrix size n. Tables 6.4 and 6.5 show the backward errors and orthogonality measures.

The backward errors of all the algorithms, except arguably SN-SVD, were acceptably small, and for large n that of QDWH-SVD was smaller than the rest by a factor 3 or more. The orthogonality of the computed $\widehat{U}$ and $\widehat{V}$ was also always acceptable, with QDWH-SVD again yielding the smallest values of the orthogonality measure.

As for the symmetric eigendecomposition, D-C (whose speed resembles that of `svd`) was the fastest, being faster than QDWH-SVD, SN-SVD, and QR by factors about 2, 3, and 3.5 respectively. The reason SN-SVD was slower than QDWH-SVD

[1]MR^3 for the bidiagonal SVD [49] was not available in LAPACK at the time of writing, but its performance is expected to resemble that for the symmetric eigenproblem.

Table 6.4
Backward error $\|A - \widehat{U}\widehat{\Sigma}\widehat{V}^*\|/\|A\|_F$; $\kappa_2(A) = 1.5$.

n	2000	4000	6000	8000	10000
QDWH-SVD	2.1e-15	2.4e-15	2.5e-15	2.7e-15	2.8e-15
SN-SVD	6.4e-14	1.4e-13	2.4e-13	3.6e-13	5.2e-13
D-C	5.7e-15	7.6e-15	9.3e-15	1.0e-14	1.2e-14
`svd`	5.9e-15	7.7e-15	9.3e-15	1.1e-14	1.2e-14
QR	1.5e-14	2.2e-14	2.6e-14	3.1e-14	3.4e-14

Table 6.5
Orthogonality of computed $\widehat{U}, \widehat{V}$: $\max(\|\widehat{U}^*\widehat{U} - I\|_F/\sqrt{n}, \|\widehat{V}^*\widehat{V} - I\|_F/\sqrt{n})$; $\kappa_2(A) = 1.5$.

n	2000	4000	6000	8000	10000
QDWH-SVD	7.7e-16	8.0e-16	8.2e-16	8.4e-16	8.5e-16
SN-SVD	3.9e-14	7.2e-14	1.1e-13	1.4e-13	1.7e-13
D-C	5.4e-15	7.0e-15	8.5e-15	9.7e-15	1.1e-14
`svd`	5.3e-15	7.1e-15	8.7e-15	1.0e-14	1.1e-14
QR	1.2e-14	1.8e-14	2.1e-14	2.5e-14	2.8e-14

despite the lower arithmetic cost in Table 4.1 appears to be that the MATLAB function `inv` does not take full advantage of the symmetry of the matrix.

Varying condition numbers. Here we fixed the matrix size $n = 10000$ and varied the condition number $\kappa_2(A)$. Tables 6.6 and 6.7 show the results. The same remarks as above apply, with QDWH-SVD giving the smallest backward errors and orthogonality measure.

We note that QDWH-SVD had speed comparable with QR, which is still a widely used algorithm (QDWH-SVD was up to two times faster for $\kappa_2(A) \approx 1$, of similar speed for $\kappa_2(A) \approx 10^5$, and up to two times slower for $\kappa_2(A) \approx u^{-1}$).

6.2.2. Rectangular rank-deficient matrices. An important usage of the SVD is to determine the rank of a matrix. Here we compare algorithms for computing the SVD of a rank-deficient matrix. We let $\Sigma = \left[\begin{smallmatrix}\Sigma_0 \\ 0_{m\times n}\end{smallmatrix}\right]$ where $\Sigma_0 = \mathrm{diag}(\sigma_i)$ so that $(1 =)\sigma_1, \dots, \sigma_r$ forms an arithmetic sequence and $\sigma_{r+1} = \cdots = \sigma_n = 0$. We take $m = 550$, $n = 500$, $\sigma_1/\sigma_r = 10$, and $r = 450$, so that $A = U\Sigma V^*$ has rank 450 and 50 zero singular values.

We used ten such matrices with random Haar distributed orthogonal U and V. We ran QDWH-SVD with $\alpha = 1$ and $\ell_0 = 1/10$, and postprocessed by performing a full QR factorization to obtain an orthogonal $\widehat{U}$ as discussed in section 5.5 (the alternative approach of setting $\ell_0 = u$ is slower but gave nearly identical results). Table 6.8 shows the minimum and maximum values of $\widehat{\sigma}_{r+1}$, which measures how accurately the zero singular values were computed, since $\sigma_{r+1} = \cdots = \sigma_n = 0$. It also shows the absolute error of the computed singular values $\max_i |\sigma_i - \widehat{\sigma}_i|$ for $r + 1 \le i \le n$ (the "zero" singular values, which satisfy $\sigma_i = O(u)$) and all i (we "obtained" the exact values σ_i using the MATLAB Symbolic Math Toolbox variable precision arithmetic function `vpa` with 32 digit precision), and the backward error. The orthogonality measure behaved as in the previous experiments.

The computed $\widehat{\sigma}_{r+1}$ was notably smaller with QDWH-SVD than with other algorithms (note that SN-SVD is not applicable here as the matrix is not square), and in general the $O(u)$ singular values were computed more accurately by QDWH-SVD. The errors in the other singular values were also smaller, although the difference is less apparent. This behavior is not peculiar to the choice of parameters n, r, and

TABLE 6.6
Backward error $\|A - \widehat{U}\widehat{\Sigma}\widehat{V}^*\|_F / \|A\|_F$; $n = 10000$.

$\kappa_2(A)$	1.1	1.5	10	10^5	10^{10}	10^{15}
QDWH-SVD	2.8e-15	2.9e-15	3.1e-15	3.3e-15	3.2e-15	3.4e-15
SN-SVD	2.6e-13	5.2e-13	1.7e-12	2.0e-12	2.0e-12	2.0e-12
D-C	1.2e-14	1.2e-14	1.1e-14	1.1e-14	1.1e-14	1.0e-14
`svd`	1.2e-14	1.2e-14	1.1e-14	1.1e-14	1.1e-14	1.1e-14
QR	3.3e-14	3.4e-14	3.2e-14	2.9e-14	2.9e-14	2.9e-14

TABLE 6.7
Orthogonality of computed $\widehat{U}, \widehat{V}$: $\max(\|\widehat{U}^*\widehat{U} - I\|_F/\sqrt{n}, \|\widehat{V}^*\widehat{V} - I\|_F/\sqrt{n})$; $n = 10000$.

$\kappa_2(A)$	1.1	1.5	10	10^5	10^{10}	10^{15}
QDWH-SVD	8.5e-16	8.5e-16	8.5e-16	8.5e-16	8.5e-16	8.5e-16
SN-SVD	1.7e-13	1.7e-13	1.7e-13	1.7e-13	1.7e-13	1.7e-13
D-C	1.1e-14	1.1e-14	1.1e-14	1.1e-14	1.1e-14	1.1e-14
`svd`	1.1e-14	1.1e-14	1.1e-14	1.1e-14	1.1e-14	1.1e-14
QR	2.8e-14	2.8e-14	2.3e-14	2.0e-14	2.0e-14	2.0e-14

TABLE 6.8
Largest "zero" singular value $\widehat{\sigma}_{r+1}$, two measures of error of computed singular values, and backward errors for matrices $A \in \mathbb{R}^{550\times 500}$ of rank $r = 450$.

	$\widehat{\sigma}_{r+1}$		$\max_{r+1\le i\le n} \lvert\sigma_i - \widehat{\sigma}_i\rvert$		$\max_{1\le i\le n} \lvert\sigma_i - \widehat{\sigma}_i\rvert$		Backward error	
	min	max	min	max	min	max	min	max
QDWH-SVD	9.5e-17	1.2e-16	6.1e-17	7.8e-17	2.0e-16	2.5e-16	2.1e-15	2.1e-15
D-C	8.1e-16	9.6e-16	6.6e-16	6.7e-16	6.6e-16	6.7e-16	3.1e-15	3.2e-15
`svd`	6.4e-16	6.9e-16	6.4e-16	7.8e-16	6.4e-16	7.8e-16	3.2e-15	3.3e-15
QR	8.4e-16	8.5e-16	4.6e-16	5.2e-16	4.6e-16	5.2e-16	7.4e-15	7.7e-15

σ_1/σ_r; other values gave similar results. The experiment suggests that QDWH-SVD is perhaps more capable of computing small (or all) singular values accurately and is hence a more reliable means of determining the rank of a matrix.

6.3. Summary of numerical experiments. The results of our experiments can be summarized as follows.

- The QDWH-based framework is generally superior to known spectral divide and conquer algorithms in terms of both speed and stability.
- QDWH-eig and QDWH-SVD both have excellent stability properties: the backward errors and the orthogonality measures of the computed eigenvectors and singular vectors are generally considerably smaller than those of the alternative algorithms. Over our complete set of experiments, including those not shown here, they also tend to compute the eigen/singular values more accurately.
- QDWH-SVD is well suited for the task of rank determination.
- On our shared-memory machine with four cores, D-C (divide and conquer) is faster than our algorithms, both for the eigendecomposition (by about a factor 3.5) and the SVD (a factor 2). For well-conditioned matrices QDWH-SVD is faster than the QR algorithm. We note that our implementations of QDWH-eig and QDWH-SVD use the built-in MATLAB functions `qr` and `chol`, so in particular the computation is not optimized of the QR factorization $\left[\begin{smallmatrix}\sqrt{c}X\\ I\end{smallmatrix}\right] = \left[\begin{smallmatrix}Q_1\\ Q_2\end{smallmatrix}\right]R$, the full-triangular matrix product $Q_1Q_2^*$ in (2.3b), or the symmetric

matrix $Z = I + cX^*X$ in (5.1a). An implementation that exploits such a structure should yield better performance.

We repeat that our experiments are all in MATLAB on a multicore platform, and the performance evaluation with a parallel implementation requires further study.

7. Conclusions and discussion.

We have developed new spectral divide and conquer algorithms for the symmetric eigendecomposition and the SVD that are backward stable and much more efficient (by a factor at least 10) than previously known algorithms of this type.

The rate of convergence of our QDWH-based methods is cubic and to our knowledge they are one of the first practical matrix iterations with this property. We note that the symmetric tridiagonal QR algorithm also converges cubically [47, Chap. 8] in the generic case, but the convergence is in terms of a scalar (the bottom off-diagonal element), as opposed to the whole matrix as in QDWH-based methods.

While our experiments suggest that our algorithms are not yet competitive in speed with divide and conquer on the four-core machine used for the tests, our algorithms have improved experimental accuracy. Moreover, our codes are pure M-files and contain some inefficiencies (see section 6.3) and we used the MATLAB function `qr` for computing the QR factorization, which does not minimize communication. Recent studies in implementing communication-optimal QR factorization [19] suggest that a communication-avoiding implementation of QR can run significantly faster than standard QR. Our algorithms are QR-based and so can take direct advantage of such progress.

A large part of the runtime of divide and conquer is consumed in the reduction stage. This reduction is a bottleneck in parallel computing, as it is not known how to do it in a way that minimizes communication. In particular, an implementation that minimizes bandwidth cost is presented in [10], but it does not minimize latency. Hence we expect that on massively parallel computing architectures in which communication cost dominates arithmetic cost our QDWH-based algorithms may be preferred. This will be investigated in future work.

Finally, we note that since QDWH-based algorithms use only basic matrix operations (matrix multiplication, QR factorization, and Cholesky factorization), our algorithms make it easier to do high-precision computations in MATLAB than conventional algorithms for symmetric eigendecompositions and the SVD. Indeed, using the Advanpix Multiprecision Computing Toolbox for MATLAB [1] we can easily run QDWH-eig and QDWH-SVD with arbitrary precision to obtain results to correspondingly high accuracy (this is not currently possible using the variable precision vpa arithmetic of the MATLAB Symbolic Math Toolbox, since the `chol` and `qr` functions do not support vpa arithmetic).

Appendix A. Flop counts.

A.1. Flop counts of QDWH for computing the polar decomposition.

As discussed in [31, Prob. 8.26] and [45], the flop count of one QDWH iteration (2.3), when all the matrices are treated as dense, is $6mn^2 + 8n^3/3$. However, some of the matrices in (2.3) have sparsity structure. Specifically, because of the trailing identity block in the QR factorization, $Q_2 = R^{-1}$ is upper triangular, and the Householder reflectors involved in the QR factorization always have $m + 1$ nonzero elements. By taking these into account we obtain a reduced flop count of QDWH as follows.

Table A.1
QDWH iteration counts for varying $\kappa_2(A)$.

$\kappa_2(A)$	1.1	1.5	10	10^3	10^5	10^{10}	10^{16}
# of (2.3)	0	0	0	1	1	2	2
# of (5.1)	2	3	4	3	4	3	4

First the Householder reflectors are applied to form $\prod_{k=1}^{n-1} H_k A = R$. Since applying a Householder reflector with $m+1$ nonzero elements to k vectors involves $4(m+1)k$ flops, forming R requires $\sum_{k=1}^{n-1} 4(n-k)(m+1) = 2mn^2$ flops (we ignore the $O(m^2)$ terms).

Next the Householder reflectors are accumulated to form $\left[\begin{smallmatrix} Q_1 \\ Q_2 \end{smallmatrix}\right] = \prod_{k=1}^{n-1} H_{n-k}\left[\begin{smallmatrix} I_n \\ 0 \end{smallmatrix}\right]$. We can see that computing in the order $H_{n-1}(H_{n-2}(\cdots(H_1\left[\begin{smallmatrix} I_n \\ 0 \end{smallmatrix}\right])$ requires the fewest flops: $\sum_{k=1}^{n-1} 4mk = 2mn^2$. Finally, forming $Q_1Q_2^*$ needs mn^2 flops, as Q_2 is upper triangular. Thus the total arithmetic cost is $5mn^2$ flops.

If A is symmetric (as in QDWH-eig) X_k is also symmetric for all $k \geq 0$, so we can save $\frac{1}{2}n^3$ flops in (2.3) by using the fact that $Q_1Q_2^*$ is also symmetric. The same applies to the Cholesky-based implementation (5.1).

As discussed in section 5.6, for an efficient implementation of QDWH it is preferable to switch to the Cholesky-based iteration (5.1) once we have $c_k \leq 100$, which indicates $\kappa_2(Z) \lesssim 100$. The number of the QR-based iterations (2.3) needed until $c_k \leq 100$ depends on $\ell_0 \lesssim \kappa_2(A)^{-1}$, and hence so does the overall cost for computing U_p. Table A.1 shows for varying $\kappa_2(A)$ (assuming $\ell_0 = 1/\kappa_2(A)$) the flop counts along with the number of iterations of (5.1) and (2.3). The middle row is obtained by computing the first k for which $c_k \leq 100$, and the last row is the first k for which $|\ell_k - 1| \leq 10^{-16}$ minus the middle row. The overall flop count is given by $5mn^2\times$(middle row) plus $(3mn^2 + n^3/3)\times$(last row).

We note that efficient techniques for computing a QR factorization (or a null space) of a matrix of the form $\left[\begin{smallmatrix} A \\ B \end{smallmatrix}\right]$ are given in [42], and similar techniques may be effective for further reducing the flop count of QDWH.

A.2. Flop counts for QDWH-eig and QDWH-SVD. We assume that during the run of QDWH-eig, σ is always taken so that A_1 and A_2 are both approximately of dimension $\frac{n}{2}$. Since one spectral division results in two submatrices of size $\approx n/2$ and the arithmetic cost scales cubically with the matrix size, the overall arithmetic cost is approximately $\sum_{i=0}^{\infty}(2\cdot 2^{-3})^i\beta = \frac{4}{3}\beta$ flops, where β is the number of flops needed for one run of spectral divide and conquer for an $n\times n$ matrix. We next evaluate a typical value of β.

For computing the polar decomposition $A - \sigma I = U_pH$, for practical dense matrices of size sufficiently smaller than 10^5, an arbitrary splitting point σ on the interval $[-\|A\|_2, \|A\|_2]$ yields $(\kappa_2(A-\sigma I))^{-1} \gtrsim)\ \ell_0 > 10^{-5}$ with high probability (if the estimate ℓ_0 is too small we can choose a different σ), so by Table A.1 we find that computing U_p typically needs $(5 - \frac{1}{2})n^3 + 4\cdot(3 + \frac{1}{3} - \frac{1}{2})n^3$ flops or fewer, where the $-\frac{1}{2}n^3$ terms are the savings that we get by exploiting symmetry.

Then subspace iteration follows, which in most cases needs just one iteration. This involves computing the full QR factorization $X \approx [\widehat{V}_1\ \widehat{V}_2]\left[\begin{smallmatrix} R \\ 0 \end{smallmatrix}\right]$ and forming the symmetric matrix $[\widehat{V}_1\ \widehat{V}_2]^*A[\widehat{V}_1\ \widehat{V}_2]$. We need $\sum_{k=1}^{n/2} 4(n-k)(n-k) = \frac{7}{6}n^3$ flops to obtain the $\frac{n}{2}$ Householder reflectors, and another $\frac{3}{2}n^3 + \frac{3}{4}n^3 = \frac{9}{4}n^3$ flops for applying them to A on both sides [25, p. 213]. The final step of one recursion of QDWH-eig is to perform updates $\widehat{V}_1 := \widehat{V}_1\widehat{V}_{21}$ and $\widehat{V}_2 := \widehat{V}_2\widehat{V}_{22}$, where $\widehat{V}_{21}$ and $\widehat{V}_{22}$ are splitting subspaces of A_1 and A_2. Each of these needs $\frac{1}{2}n^3$ flops. Hence the total flop count of one recursion of QDWH-eig is

$$\begin{aligned} \beta &= \Big(\Big(5-\frac{1}{2}\Big) + 4\cdot\Big(3+\frac{1}{3}-\frac{1}{2}\Big) + \frac{7}{6}+\frac{9}{4}+1\Big)n^3 \\ &= \Big(20+\frac{1}{4}\Big)n^3, \end{aligned} \tag{A.1}$$

so the total arithmetic cost of QDWH-eig is $\frac{4}{3}\beta = 27n^3$. If we perform the Newton–Schulz postprocessing, an additional $3n^3$ flops is needed.

For SN-eig, the cost for the polar decomposition is n^3 per iteration if symmetry is taken advantage of in the inversions [31, p. 336], so in the typical case $\kappa_2(A-\sigma I) \le 10^5$ in which seven iterations is needed, the overall cost is $\frac{4}{3}(7+\frac{7}{6}+\frac{9}{4}+1)n^3 = (15+\frac{2}{9})n^3$.

For QDWH-SVD, the additional arithmetic cost is in computing an $m \times n$ polar decomposition $A = U_pH$. The flop count for computing U_p can be determined from Table A.1 as $5mn^2\times$(middle row) plus $(3mn^2+n^3/3)\times$(last row), and computing H by (2.4) requires $2mn^2$ flops. We conclude that the total flop count for QDWH-SVD ranges from $8mn^2 + (27+2/3)n^3$ (for $\kappa_2(A) \approx 1$) to $24mn^2 + (28+1/3)n^3$ (for $\kappa_2(A) \gg 1$). When $m = n$, these range roughly between $35n^3$ and $52n^3$ flops.

For SN-SVD applied to a square matrix, the polar decomposition needs $2n^3$ flops per iteration for inversion of a nonsymmetric matrix, so in total the cost ranges from $(4+2+15+\frac{2}{9})n^3 = (21+\frac{2}{9})n^3$ to $(18+2+15+\frac{2}{9})n^3 = (35+\frac{2}{9})n^3$.

Recall that when $m > n$ we can perform an initial QR factorization $A = QR$ and then work on the $n \times n$ matrix R. Since this can be done for an extra $4mn^2 - \frac{4}{3}n^3$ flops [25, p. 232], to minimize the arithmetic cost this process is recommended whenever $24mn^2 + (28+1/3)n^3 > 4mn^2 - \frac{4}{3}n^3 + (52+\frac{1}{3})n^3$, which is $m > 1.13n$.

Finally, when only the eigen/singular values are required, the accumulation of the orthogonal factors is not necessary, and this leads to a reduced arithmetic cost for QDWH-eig and for QDWH-SVD. However, in this case it is more likely that the standard algorithms based on reduction to simpler forms will outperform them, requiring just $\frac{4}{3}n^3$ flops for computing eigenvalues and $4mn^2 - \frac{4}{3}n^3$ flops for the singular values.

Acknowledgments. We are grateful to Roland Freund, Françoise Tisseur, and Alex Townsend for their valuable comments on the work, and Grey Ballard for discussions about communication-avoiding algorithms.

REFERENCES

[1] *Multiprecision Computing Toolbox for MATLAB*, Advanpix LLC, Tokyo, available online at http://www.advanpix.com.

[2] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, and D. C. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999.

[3] L. Auslander and A. Tsao, *On parallelizable eigensolvers*, Adv. in Appl. Math., 13 (1992), pp. 253–261.

[4] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, 2000.

[5] Z. Bai and J. W. Demmel, *Design of a parallel nonsymmetric eigenroutine toolbox, Part* I, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Volume I, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds., SIAM, Philadelphia, 1993, pp. 391–398. Also available as Research report 92-09, Department of Mathematics, University of Kentucky, Lexington, KY, 1992.

[6] Z. Bai and J. W. Demmel, *Design of a Parallel Nonsymmetric Eigenroutine Toolbox, Part* II, Research Report 95-11, Department of Mathematics, University of Kentucky, Lexington, KY, 1995.

[7] Z. Bai and J. W. Demmel, *Using the matrix sign function to compute invariant subspaces*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 205–225.

[8] Z. Bai, J. W. Demmel, J. J. Dongarra, A. Petitet, H. Robinson, and K. Stanley, *The spectral decomposition of nonsymmetric matrices on distributed memory parallel computers*, SIAM J. Sci. Comput., 18 (1997), pp. 1446–1461.

[9] Z. Bai, J. W. Demmel, and M. Gu, *Inverse free parallel spectral divide and conquer algorithms for nonsymmetric eigenproblems*, Numer. Math., 76 (1997), pp. 279–308.

[10] G. Ballard, J. Demmel, and I. Dumitriu, *Minimizing Communication for Eigenproblems and the Singular Value Decomposition*, Technical report 237, LAPACK Working Note, 2010. UCB/EECS-2010-136.

[11] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, *Communication-optimal parallel and sequential Cholesky decomposition*, SIAM J. Sci. Comput., 32 (2010), pp. 3495–3523.

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

[13] A. N. Beavers, Jr., and E. D. Denman, *A computational method for eigenvalues and eigenvectors of a matrix with real eigenvalues*, Numer. Math., 21 (1973), pp. 389–396.

[14] C. Bischof, S. Huss-Lederman, X. Sun, and A. Tsao, *The PRISM project: Infrastructure and algorithms for parallel eigensolvers*, in Proceedings of the Scalable Parallel Libraries Conference, IEEE, Washington, DC, 1993, pp. 123–131.

[15] A. Ya. Bulgakov and S. K. Godunov, *Circular dichotomy of the spectrum of a matrix*, Siberian Math. J., 29 (1988), pp. 734–744.

[16] R. Byers and H. Xu, *A new scaling for Newton's iteration for the polar decomposition and its backward stability*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 822–843.

[17] T. F. Chan, *An improved algorithm for computing the singular value decomposition*, ACM Trans. Math. Software, 8 (1982), pp. 72–83.

[18] J. Demmel, I. Dumitriu, and O. Holtz, *Fast linear algebra is stable*, Numer. Math., 108 (2007), pp. 59–91.

[19] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A206–A239.

[20] J. W. Demmel and N. J. Higham, *Stability of block algorithms with fast level-*3 *BLAS*, ACM Trans. Math. Software, 18 (1992), pp. 274–291.

[21] E. D. Denman and A. N. Beavers, Jr., *The matrix sign function and computations in systems*, Appl. Math. Comput., 2 (1976), pp. 63–94.

[22] I. S. Dhillon and B. N. Parlett, *Orthogonal eigenvectors and relative gaps*, SIAM J. Matrix Anal. Appl., 25 (2004), pp. 858–899.

[23] S. K. Godunov, *Problem of the dichotomy of the spectrum of a matrix*, Siberian Math. J., 27 (1986), pp. 649–660.

[24] S. K. Godunov, *Modern Aspects of Linear Algebra*, Transl. Math. Monogr. 175, AMS, Providence, RI, 1998.

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

[26] L. Grigori, J. W. Demmel, and H. Xiang, *CALU: A communication optimal LU factorization algorithm*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1317–1350.

[27] M. Gu and S. C. Eisenstat, *A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 172–191.

[28] N. J. Higham, *Computing the polar decomposition—with applications*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1160–1174.

[29] N. J. Higham, *The matrix sign decomposition and its relation to the polar decomposition*, Linear Algebra Appl., 212/213 (1994), pp. 3–20.

[30] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.

[31] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, Philadelphia, 2008.

[32] N. J. Higham and P. Papadimitriou, *Parallel Singular Value Decomposition via the Polar Decomposition*, Numerical Analysis Report 239, Manchester Centre for Computational Mathematics, Manchester, England, 1993.

[33] N. J. Higham and P. Papadimitriou, *A new parallel algorithm for computing the singular value decomposition*, in Proceedings of the Fifth SIAM Conference on Applied Linear Algebra, J. G. Lewis, ed., SIAM, Philadelphia, 1994, pp. 80–84.

[34] N. J. Higham and F. Tisseur, *A block algorithm for matrix* 1*-norm estimation, with an application to* 1*-norm pseudospectra*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1185–1201.

[35] S. Huss-Lederman, E. S. Quintana-Ortí, X. Sun, and Y.-J. Y. Wu, *Parallel spectral division using the matrix sign function for the generalized eigenproblem*, Int. J. High Speed Computing, 11 (2000), pp. 1–14.

[36] S. Huss-Lederman, A. Tsao, and T. Turnbull, *A parallelizable eigensolver for real diagonalizable matrices with real eigenvalues*, SIAM J. Sci. Comput., 18 (1997), pp. 869–885.

[37] 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, 2012.

[38] C.-C. Lin and E. Zmijewski, *A Parallel Algorithm for Computing the Eigenvalues of an Unsymmetric Matrix on an SIMD Mesh of Processors*, Report TRCS 91-15, Department of Computer Science, University of California, Santa Barbara, 1991.

[39] A. N. Malyshev, *Guaranteed accuracy in spectral problems of linear algebra*, I, Siberian Advances in Mathematics, 2 (1992), pp. 144–197.

[40] A. N. Malyshev, *Guaranteed accuracy in spectral problems of linear algebra*, II, Siberian Advances in Mathematics, 2 (1992), pp. 153–204.

[41] A. N. Malyshev, *Parallel algorithms for solving some spectral problems of linear algebra*, Linear Algebra Appl., 188 (1993), pp. 489–520.

[42] M. Marques, E. Quintana-Orti, and G. Quintana-Orti, *Specialized spectral division algorithms for generalized eigenproblems via the inverse-free iteration*, in Applied Parallel Computing. State of the Art in Scientific Computing, 8th International Workshop, PARA 2006, B. Kågström, E. Elmroth, J. Dongarra, and J. Waśniewski, eds., Lecture Notes in Comput. Sci. 4699, Springer-Verlag, Berlin, 2007, pp. 157–166.

[43] O. A. Marques, C. Vömel, J. W. Demmel, and B. N. Parlett, *Algorithm* 880: *A testing infrastructure for symmetric tridiagonal eigensolvers*, ACM Trans. Math. Software, 35 (2008), pp. 1–13.

[44] *NAG Toolbox for MATLAB*, NAG Ltd., Oxford, available online at http://www.nag.co.uk.

[45] Y. Nakatsukasa, Z. Bai, and F. Gygi, *Optimizing Halley's iteration for computing the matrix polar decomposition*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2700–2720.

[46] Y. Nakatsukasa and N. J. Higham, *Backward stability of iterations for computing the polar decomposition*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 460–479.

[47] B. N. Parlett, *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, 1998.

[48] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[49] P. Willems, *On MR^3-type Algorithms for the Tridiagonal Symmetric Eigenproblem and the Bidiagonal SVD*, Ph.D. thesis, University of Wuppertal, Wuppertal, Germany, 2010.

[50] H. Zha and Z. Zhang, *A cubically convergent parallelizable method for the Hermitian eigenvalue problem*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 468–486.

[51] Z. Zhang, H. Zha, and W. Ying, *Fast parallelizable methods for computing invariant subspaces of Hermitian matrices*, J. Comput. Math., 25 (2007), pp. 583–594.