A block algorithm for computing antitriangular factorizations of symmetric matrices

Zvonimir Bujanović* Daniel Kressner†

November 13, 2014

Abstract

Any symmetric matrix can be reduced to antitriangular form in finitely many steps by orthogonal similarity transformations. This form reveals the inertia of the matrix and has found applications in, e.g., model predictive control and constraint preconditioning. Originally proposed by Mastronardi and Van Dooren, the existing algorithm for performing the reduction to antitriangular form is primarily based on Householder reflectors and Givens rotations. The poor memory access pattern of these operations implies that the performance of the algorithm is bound by the memory bandwidth. In this work, we develop a block algorithm that performs all operations almost entirely in terms of level 3 BLAS operations, which feature a more favorable memory access pattern and lead to better performance. These performance gains are confirmed by numerical experiments that cover a wide range of different inertia.

1 Introduction

Mastronardi and Van Dooren [10] recently proposed a novel antitriangular factorization for symmetric matrices. Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$, there is an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ such that

$$Q^T A Q = \begin{array}{c} \begin{matrix} n_0 & n_1 & n_2 & n_1 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Y \\ 0 & 0 & X & Z \\ 0 & Y^T & Z^T & W \end{pmatrix} \end{array} \begin{matrix} n_0 \\ n_1 \\ n_2 \\ n_1 \end{matrix}, \tag{1.1}$$

where the middle $n_2 \times n_2$ matrix X is either positive or negative definite. The $n_1 \times n_1$ matrix Y is nonsingular and lower antitriangular, that is, it is zero above the antidiagonal: $Y(i,j) = 0$ whenever $i + j \leq n_1$. The block sizes in (1.1) are directly related to the inertia of A, that is, the triplet of integers containing the number of its positive/negative/zero eigenvalues, respectively:

$$\text{Inertia}(A) = (n_+, n_-, n_0), \quad n_1 = \min\{n_+, n_-\}, \quad n_2 = \max\{n_+, n_-\} - n_1.$$

*University of Zagreb, Dept. of Mathematics, Croatia, zbujanov@math.hr. Part of this work was done while the author was a postdoctoral researcher at Max Planck Institute Magdeburg, Germany.

†ANCHP, MATHICSE, EPF Lausanne, Switzerland, daniel.kressner@epfl.ch

If A itself is definite then $n_0 = n_1 = 0$ and the factorization (1.1) becomes vacuous. The case primarily of interest is therefore when A is indefinite. The antitriangular factorization (1.1) has found applications in tracking dominant eigenspaces [9], indefinite constrained least-squares problems [12], model predictive control [13], and constraint preconditioning [15].

It is more common to find orthogonal similarity factorizations that aim at reducing A to tridiagonal or diagonal form. However, these factorizations have a number of disadvantages compared to (1.1).

It is well known that the data dependencies of the standard algorithm [6, §8.3.1] for tridiagonal reduction make it difficult to parallelize. Specifically, when reducing one column of A to tridiagonal form, the resulting orthogonal transformation needs to be applied from both sides to A before the next column can be processed. This two-sided nature of tridiagonal reduction ultimately imposes a limit on how much of the algorithm can be reformulated in terms of matrix-matrix multiplications or, more generally, level 3 BLAS. Although progress has been made to circumvent this drawback, see [1, 2, 7, 8] for recent work, the parallelization of two-sided reduction algorithms is still considered a major challenge. In contrast to the two-sided nature of tridiagonal reduction, the algorithm proposed in [10] for reducing A to antitriangular form has a strong one-sided flavor, similar to standard algorithms for the Cholesky and QR factorizations [6, §4.2.6 and §5.2.2, resp.]. In this paper, we will use this property to derive a novel block algorithm for computing antitriangular factorizations, with asymptotically 100% of its operations performed in terms of level 3 BLAS.

Another disadvantage of tridiagonal and diagonal factorizations is that they usually do not benefit from sparsity in A. Antitriangular factorizations can exploit certain types of sparsity [11]. Although this will not be discussed in detail, we expect that the ideas presented in this paper can be utilized in such sparsity-exploiting algorithms, similar to sparse QR factorizations [4].

Throughout this paper we use Matlab notation for submatrices: $A(i_1{:}j_1, i_2{:}j_2)$ denotes the submatrix of $A \in \mathbb{R}^{n\times n}$ consisting of all the entries of A in rows i_1 through j_1 and columns i_2 through j_2, with the conventions $A(i{:}j, :) = A(i{:}j, 1{:}n)$, $A(:, i{:}j) = A(1{:}n, i{:}j)$, and $i{:}i = i$.

2 Algorithms

Following [10], our proposed block algorithm proceeds by progressively computing the antitriangular factorizations (1.1) for the leading principal submatrices of A.

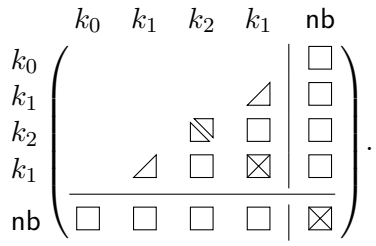
Suppose that the antitriangular factorization of $A(1{:}k, 1{:}k)$ has been computed for some $k < n$. We will symbolize the antitriangular form with

$$\begin{array}{c} \\ k_0 \\ k_1 \\ k_2 \\ k_1 \end{array} \begin{array}{c} \begin{array}{cccc} k_0 & k_1 & k_2 & k_1 \end{array} \\ \begin{pmatrix} & & & \\ & & & \diagup \\ & & \diagdown & \square \\ & \diagup & \square & \boxtimes \end{pmatrix} \end{array}. \tag{2.1}$$

The symbol $\diagdown$ indicates that the middle (positive or negative) definite block is always stored in terms of its Cholesky factorization $\varepsilon R^T R$ with R upper triangular and $\varepsilon \in \{+1, -1\}$. The symbol

$\boxtimes$ indicates a symmetric matrix. Note that only the upper triangular part of A needs to be stored and processed.

The kth step of the scalar algorithm presented in [10] starts with bordering (2.1) with the $(k+1)$st column and row of A. It restores the antitriangular form by a sequence of Householder reflectors and Givens rotations. Updating the orthogonal factor with the performed transformations then yields the antitriangular factorization of $A(1{:}k+1, 1{:}k+1)$. This procedure requires $\mathcal{O}(k^2)$ operations and access to $\mathcal{O}(k^2)$ memory. For larger k, its performance will therefore be bound by the memory bandwidth. To avoid this, the computational density of the kth step needs to be increased. Our algorithm approaches this task by bordering (2.1) with $\mathsf{nb} > 1$ columns and rows:



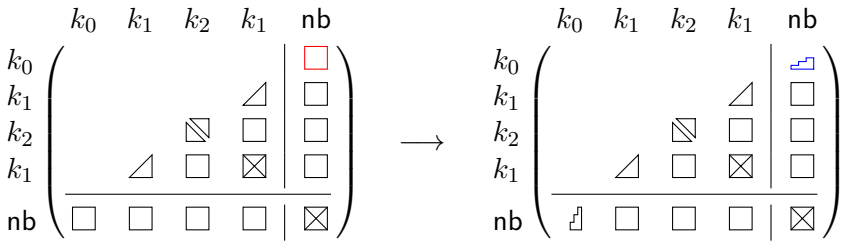
(2.2)

The rest of this section is concerned with the development of efficient blocked algorithms that restore the antitriangular form of this matrix, ultimately resulting in the antitriangular factorization of $A(1{:}k+\mathsf{nb}, 1{:}k+\mathsf{nb})$. Depending on the architecture, in particular the cache size, the value of nb will typically be chosen between 64 and 512. Our algorithms are designed to perform well for all possible values of k_0, k_1, k_2; no implicit assumption on their relation to nb is made.

2.1 Main algorithm

In each of the following figures describing the algorithm, we use the following convention: blocks marked in red on the left-hand side of the picture are being modified and used to compute an orthogonal transformation, which is then applied onto the rest of the matrix as well. The result is shown on the right-hand side, and only the blocks marked in blue are affected by the transformation. (Due to symmetry, only blocks on and above the main diagonal are stored and colored.)

Step 1. The first step of our algorithm consists of reducing the top-right block in (2.2) to anti-staircase form by applying Algorithm 1 (see Section 2.2.1 below):



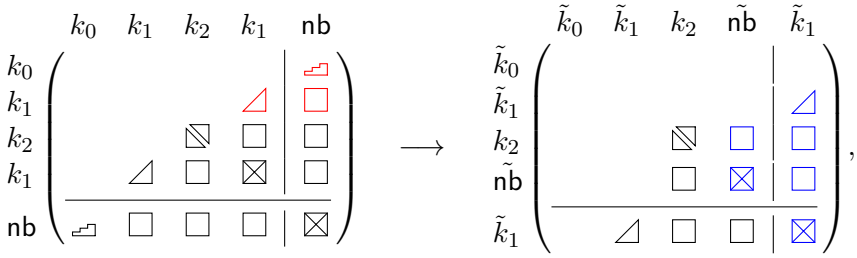
We say that a $k_0 \times \mathsf{nb}$ matrix is in anti-staircase form if there is $0 \le \ell \le \min\{k_0, \mathsf{nb}\}$ such that only the first $k_0 - \ell$ rows are zero and the remaining rows satisfy $j_{k_0} < j_{k_0-1} < \cdots < j_{k_0-\ell+1}$, where j_i is the index of the first nonzero entry in the ith row.

The $\ell \le \min\{k_0, \mathsf{nb}\}$ Householder reflectors returned by Algorithm 1 are applied to the rest of A and to Q by a blocked algorithm, using compact WY representations [6, §5.1.7]. For this purpose, our implementation uses a minor modification of the LAPACK routine `DORMQL` (which applies the reflectors computed by the QL factorization of a matrix).

Step 2. After Step 1, the first $k_0 - \ell$ rows of the matrix are zero. Letting the $(\ell + k_1) \times (k_1 + \mathsf{nb})$ matrix $\tilde{A}_1 = \begin{pmatrix} & \\ \blacktriangle & \blacksquare \end{pmatrix}$ contain the top-right nonzero blocks, we determine a factorization

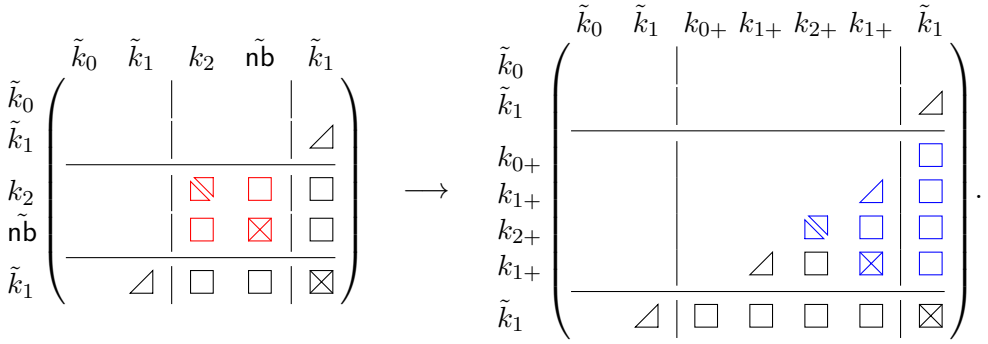
$$\tilde{A}_1 = A_1 Q_1, \tag{2.3}$$

such that A_1 is lower antitriangular and Q_1 is orthogonal. Note that $\tilde{A}_1$ has full row rank and hence all antidiagonal entries of A_1 are nonzero. To compute (2.3), we have implemented a blocked algorithm based on minor modifications of the LAPACK routine `DGELQF` (for computing the LQ factorization of a matrix) as well as the auxiliary routines for creating and applying compact WY representations. Similar to Step 1, the matrix Q_1 is represented in terms of Householder reflectors and applied to the rest of A and Q using a minor modification of the blocked LAPACK routine `DORMLQ`. The updated matrix takes the shape



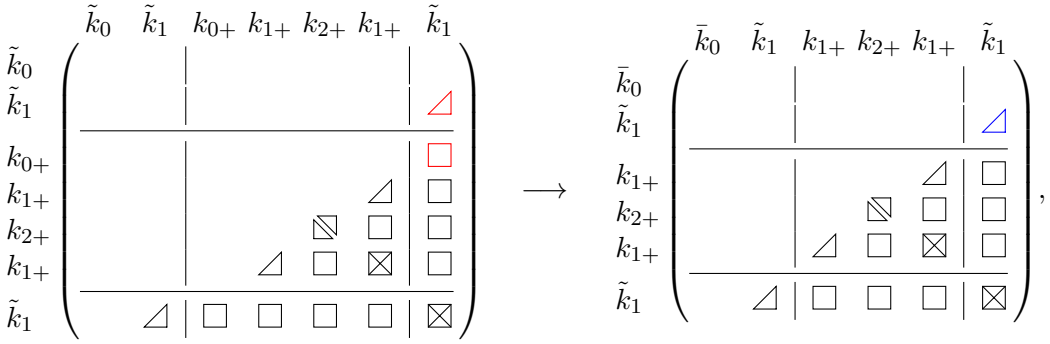
where we have repartitioned the block sizes as $\tilde{k}_0 = k_0 - \ell$, $\tilde{k}_1 = k_1 + \ell$, $\tilde{\mathsf{nb}} = \mathsf{nb} - \ell$. Also, note that the bottom-right $(k_1 + \mathsf{nb}) \times (k_1 + \mathsf{nb})$ block has to be updated by applying Q_1 both from the left and the right side.

Step 3. The antitriangular factorization of the symmetric $(k_2 + \tilde{\mathsf{nb}}) \times (k_2 + \tilde{\mathsf{nb}})$ middle block is computed:



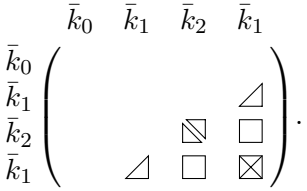
When $k_2 \leq \tilde{\mathsf{nb}}$, the scalar algorithm from [10] is used to perform this factorization. The resulting orthogonal transformation matrix Q_2 is applied to the rest of A and Q by matrix-matrix multiplication. Otherwise, when $k_2 > \tilde{\mathsf{nb}}$, the blocked algorithm described in Section 2.2.2 below is used.

Step 4. By a slight modification of the LAPACK routine `DGEQRF` (for computing the QR factorization), we reduce the blocks at positions (2,7) and (2,8) to lower antitriangular form:



where we let $\bar{k}_0 = \tilde{k}_0 + k_{0+}$.

Setting $\bar{k}_1 = \tilde{k}_1 + k_{1+}$ and $\bar{k}_2 = k_{2+}$, we finally obtain the antitriangular form of $A(1 : k + \mathsf{nb}, 1 : k + \mathsf{nb})$:



2.2 Ingredients of the main algorithm

In the following, we provide details on some of the steps of the main algorithm.

2.2.1 Reduction to anti-staircase form

Step 1 of the main algorithm requires the reduction of a rectangular matrix to anti-staircase form. Given an $m \times n$ matrix B, one step of the reduction proceeds as follows. Negligible leading columns of B are set to zero and skipped in the process. Let j_1 denote the index of the first column of B with 2-norm larger than a user-specified tolerance tol. Then a Householder reflector H_1 is computed such that the first $m-1$ elements of this column are annihilated. In effect, the matrix $H_1 B$ takes the form

$$\begin{array}{c} \\ m-1 \\ 1 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 - 1 & 1 & n - j_1 \end{array} \\ \begin{pmatrix} 0 & 0 & \tilde{B} \\ 0 & \tilde{b}_{m,j_1} & \star \end{pmatrix} \end{array}$$

with $\tilde{b}_{m,j_1} \neq 0$. The same procedure is applied again to $\tilde{B}$ and repeated until this block disappears, see Algorithm 1 for more details. This algorithm requires $\mathcal{O}(\ell mn)$ operations, where $\ell \leq \min\{m, n\}$.

Algorithm 1 Reduction to anti-staircase form

Input: Matrix $B \in \mathbb{R}^{m\times n}$, tolerance tol.

Output: Sequence of Householder reflectors $H_1, \ldots, H_\ell$ and updated matrix $B \leftarrow H_\ell H_{\ell-1} \cdots H_1 B$ in anti-staircase form.

```
ℓ ← 0
for j = 1, 2, ... n do
  if ‖B(1 : m − ℓ, j)‖₂ ≤ tol then
    B(1 : m − ℓ, j) ← 0
  else
    ℓ ← ℓ + 1
    Construct Householder reflector H_ℓ annihilating the first m − ℓ entries of B(:, j).
    B ← H_ℓ B
  end if
end for
```

2.2.2 Antitriangular factorization of symmetric matrix with factorized $(1,1)$ block

In the presence of a large middle block in Step 3 of the main algorithm, the use of a scalar algorithm for reducing this block would deteriorate performance. In the following, we therefore develop a blocked algorithm for computing the antitriangular factorization of a matrix having the form

$$M = \begin{array}{c} \\ k \\ \mathsf{nb} \end{array} \begin{array}{c} \begin{array}{cc} k & \mathsf{nb} \end{array} \\ \begin{pmatrix} \blacksquare & \square \\ \square & \boxtimes \end{pmatrix} \end{array} = \begin{pmatrix} \varepsilon R^T R & Z \\ Z^T & W \end{pmatrix},$$

where W is symmetric. It is assumed that $k > \mathsf{nb}$.

Step 3.1. Using Algorithm 2, orthogonal matrices $\tilde{Q}$ and $\tilde{U}$ are determined such that $\tilde{Q}^T Z$ is lower triangular and $\tilde{U}^T R \tilde{Q}$ is upper triangular.
Applied to M, this yields the shape

$$\begin{array}{c} \\ k \\ \mathsf{nb} \end{array} \begin{array}{c} \begin{array}{cc} k & \mathsf{nb} \end{array} \\ \begin{pmatrix} \blacksquare & \square \\ \square & \boxtimes \end{pmatrix} \end{array} \quad \longrightarrow \quad \begin{array}{c} \\ k \\ \mathsf{nb} \end{array} \begin{array}{c} \begin{array}{cc} k & \mathsf{nb} \end{array} \\ \begin{pmatrix} \blacksquare & \blacktriangleright \\ \blacktriangleleft & \boxtimes \end{pmatrix} \end{array}.$$

In our implementation, the matrix $\tilde{U}$ is actually not formed, as it is not needed. Also, the matrix $\tilde{Q}$ is not formed; instead, the compact WY representation of each individual factor Q_i is immediately applied to the rest of A and Q during the execution of Algorithm 2. In effect, the total number of operations needed for performing the updates of A and Q remains $\mathcal{O}(k \cdot \mathsf{nb} \cdot n)$.

Algorithm 2 QL factorization with simultaneous preservation of Cholesky factorization

Input: Upper triangular matrix $R \in \mathbb{R}^{k\times k}$ and matrix $Z \in \mathbb{R}^{k\times \mathsf{nb}}$ such that $k < \mathsf{nb}$.

Output: Orthogonal matrices $Q, U \in \mathbb{R}^{k\times k}$, such that $Z \leftarrow Q^T Z$ is lower triangular and $R \leftarrow U^T RQ$ is upper triangular.

$Q \leftarrow I_k$, $U \leftarrow I_k$, $i \leftarrow 1$

while $i \le k - \mathsf{nb}$ **do**

$j = \min\{k, i + 2\cdot \mathsf{nb} - 1\}$

Compute orthogonal matrix Q_i such that $Z(i:j,:) \leftarrow Q_i^T Z(i:j,:)$ is lower triangular, using the QL factorization (LAPACK routine `DGEQLF`).

Update $R(1:j, i:j) \leftarrow R(1:j, i:j)Q_i$ and $Q(:, i:j) \leftarrow Q(:, i:j)Q_i$ using the compact WY representation of Q_i (LAPACK routine `DORMQL`).

Compute orthogonal matrix U_i such that $R(i:j, i:i+\mathsf{nb}-1) \leftarrow U_i^T R(i:j, i:i+\mathsf{nb}-1)$ is upper triangular, using the QR factorization (LAPACK routine `DGEQRF`).

Update $R(i:j, i+\mathsf{nb}:k) \leftarrow U_i^T R(i:j, i+\mathsf{nb}:k)$ and $U(:, i:j) \leftarrow U(:, i:j)U_i$ using the compact WY representation of U_i (LAPACK routine `DORMQR`).

$i \leftarrow i + \mathsf{nb}$

end while

Step 3.2. After Step 3.1 has been performed, we partition the updated Cholesky factor as

$$R = \begin{array}{c} \\ \tilde{k} \\ \mathsf{nb} \end{array} \begin{array}{c} \begin{array}{cc} \tilde{k} & \mathsf{nb} \end{array} \\ \begin{pmatrix} \diagdown & \square \\ & \diagdown \end{pmatrix} \end{array} = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

with $\tilde{k} = k - \mathsf{nb}$. The relation

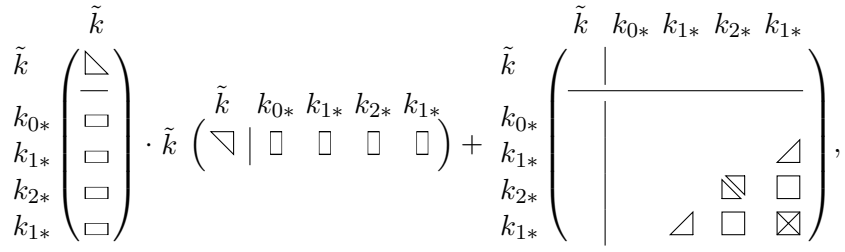
$$R^T R = \begin{pmatrix} R_{11}^T \\ R_{12}^T \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & R_{22}^T R_{22} \end{pmatrix}$$

then allows us to decompose M schematically as

$$M = \begin{array}{c} \\ k \\ \mathsf{nb} \end{array} \begin{array}{c} \begin{array}{cc} k & \mathsf{nb} \end{array} \\ \begin{pmatrix} \diagdown & \diagdown \\ \diagdown & \boxtimes \end{pmatrix} \end{array} = \begin{array}{c} \\ \tilde{k} \\ \mathsf{nb} \\ \mathsf{nb} \end{array} \begin{array}{c} \begin{array}{c} \tilde{k} \end{array} \\ \begin{pmatrix} \diagdown \\ \hline \square \\ \\ \end{pmatrix} \end{array} \cdot \tilde{k} \begin{array}{c} \begin{array}{ccc} \tilde{k} & \mathsf{nb} & \mathsf{nb} \end{array} \\ \left(\begin{array}{c|cc} \diagdown & \square & \\ & & \end{array}\right) \end{array} + \begin{array}{c} \\ \tilde{k} \\ \mathsf{nb} \\ \mathsf{nb} \end{array} \begin{array}{c} \begin{array}{ccc} \tilde{k} & \mathsf{nb} & \mathsf{nb} \end{array} \\ \left(\begin{array}{c|cc} & & \\ \hline & \boxtimes & \diagdown \\ & \diagdown & \boxtimes \end{array}\right) \end{array}. \tag{2.4}$$

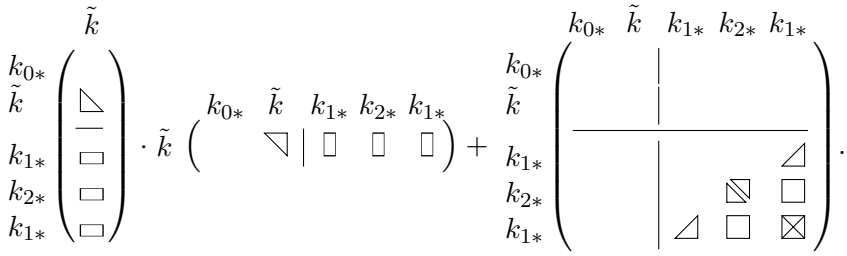
Step 3.3. Let the symmetric $2\cdot\mathsf{nb} \times 2\cdot\mathsf{nb}$ matrix $M_{22} = \begin{pmatrix} \varepsilon R_{22}^T R_{22} & Z_2 \\ Z_2^T & W \end{pmatrix}$ denote the bottom right block in the second term of (2.4). Since R_{22} is invertible, the number of eigenvalues of M_{22} with sign ε is not smaller than the total number of eigenvalues that are zero or have sign $-\varepsilon$; this is a simple consequence of the Cauchy interlacing theorem, see, e.g., [14]. Consequently, if there is a middle block X in the antitriangular factorization of M_{22} then it has sign ε. Computing this

factorization with the scalar algorithm and applying the resulting orthogonal transformation to both terms of (2.4) (and to the rest of A and Q) yields the shape



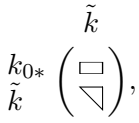
where $k_{0*} + k_{1*} + k_{2*} + k_{1*} = 2 \cdot \mathsf{nb}$.

Step 3.4. By computing an RQ factorization of its first $\tilde{k} + k_{0*}$ columns, the factor of the first term can be reduced without affecting the second term:



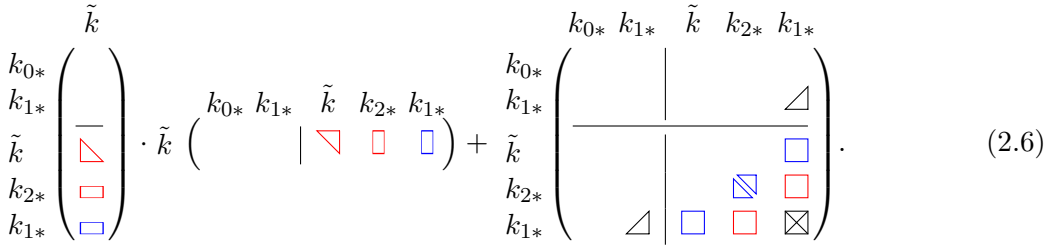
(2.5)

Remark 2.1 *Especially when $\tilde{k} \gg k_{0*}$, it is important that the RQ factorization in Step 3.4 exploits the upper triangular structure of the leading $\tilde{k} \times \tilde{k}$ block to reduce the computational complexity from $\mathcal{O}((\tilde{k} + k_{0*})\tilde{k}^2)$ to $\mathcal{O}(k_{0*}\tilde{k}^2)$. For the closely related problem of computing the QR factorization of a matrix of the form*



the corresponding LAPACK routine `DGEQRF` *exploits most of the triangular structure automatically. This is achieved by exploiting trailing zero entries in the application of (blocked) Householder reflectors with the auxiliary routine* `DLARFB`*. Unfortunately, we have observed that the currently implemented mechanism does not function for the situation at hand; neither the vanilla Netlib implementation nor the Intel MKL implementation of the LAPACK routine* `DGERQF` *for RQ factorizations seem to make use of the structure arising in Step 3.4. To overcome this drawback and obtain a complexity of $\mathcal{O}(k_{0*}\tilde{k}^2)$, we have applied a simple patch to* `DLARFB` *such that it also exploits leading zero entries in the application of Householder reflectors. Note that this patch is needed to accelerate* `DGEQLF` *in Step 3.1 as well.*

Step 3.5. Step 3.4 is repeated for the first $\tilde{k}+k_{1*}$ nonzero columns of the factor in the first term, leading to



(2.6)

Note that, in contrast to Step 3.4, the application of the corresponding transformation affects the second term: the zero-pattern of the blocks $(2,5)$ and $(3,5)$ from (2.5) is now destroyed. To continue the algorithm, we have to enforce the lower antitriangularity of the newly introduced $(2,5)$ block in (2.6); this is achieved by using the same modification of `DGEQRF` as described in Step 4 above.

The following lemma implies that the newly introduced antitriangular block in the second term of (2.6) is invertible.

Lemma 2.2 *Consider partitioned matrices*

$$\begin{array}{c} \\ m \\ \end{array}\!\!\begin{array}{c} \begin{array}{cc} m & n \end{array} \\ \begin{pmatrix} A_1 & A_2 \end{pmatrix} \end{array}\!\! = A, \qquad \begin{array}{c} \\ n \\ \end{array}\!\!\begin{array}{c} \begin{array}{cc} m & n \end{array} \\ \begin{pmatrix} 0 & B_2 \end{pmatrix} \end{array}\!\! = B,$$

with A_1 and B_2 nonsingular. Let $A = RQ$ be an RQ factorization and partition

$$\begin{array}{c} \\ n \\ \end{array}\!\!\begin{array}{c} \begin{array}{cc} n & m \end{array} \\ \begin{pmatrix} \tilde{B}_1 & \tilde{B}_2 \end{pmatrix} \end{array}\!\! = BQ^T.$$

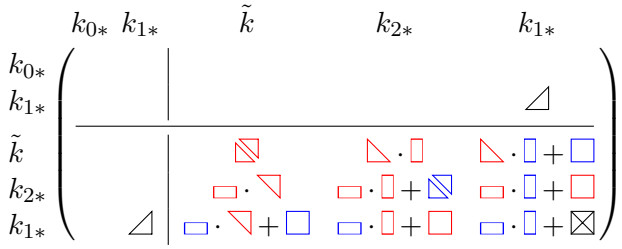
Then $\tilde{B}_1$ is invertible.

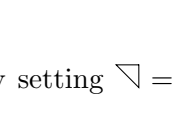
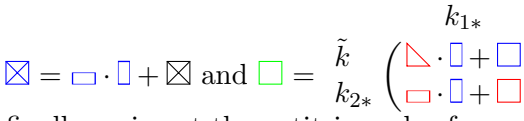
Proof. Partition

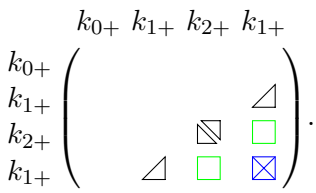
$$\begin{array}{c} \\ n \\ m \end{array}\!\!\begin{array}{c} \begin{array}{cc} m & n \end{array} \\ \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix} \end{array}\!\! = Q, \qquad \begin{array}{c} \\ m \\ \end{array}\!\!\begin{array}{c} \begin{array}{cc} n & m \end{array} \\ \begin{pmatrix} 0 & R_2 \end{pmatrix} \end{array}\!\! = R.$$

Then R_2 is invertible and $A = RQ$ implies that $Q_{21} = R_2^{-1}A_1$ is also invertible. By the CS decomposition of Q [6, Thm. 2.6.3], Q_{12} is invertible as well, which in turn yields the invertibility of $\tilde{B}_1 = B_2 Q_{12}^T$. □

Step 3.6 After the completion of Steps 3.3–3.5 both terms are merged again, yielding



the color-coding is inherited from (2.6). The Cholesky factor of the middle $(\tilde{k}+k_{2*})\times(\tilde{k}+k_{2*})$ block is readily available by setting  . All other blocks are formed explicitly:  Setting $k_{0+}=k_{0*}$, $k_{1+}=k_{1*}$ and $k_{2+}=\tilde{k}+k_{2*}$, we finally arrive at the antitriangular form



By construction, the middle block is definite with sign ε. Also, Lemma 2.2 implies that the lower antitriangular blocks are invertible.

2.3 Computational complexity

The precise overall complexity of the described algorithm for computing the antitriangular factorization of the bordered matrix in (2.2) and updating the rest of A and Q depends in a nontrivial way on the inertia of the involved matrices. However, the asymptotic complexity can be easily seen to be $\mathcal{O}(\mathsf{nb}\cdot n^2)$, leading to a total asymptotic complexity of $\mathcal{O}(n^3)$. All reductions that involve matrices with a size potentially proportional to n are based on blocked algorithms which have asymptotically 100% of their operations performed by level 3 BLAS. The overhead involved in the blocking becomes negligible as n increases.

3 Numerical experiments

We implemented the algorithm from [10] as well as the blocked algorithm described in this paper in Fortran 90.[1] The following computational environment was used:

- 2x Intel(R) Xeon(R) CPU E5620 @ 2.40GHz (8 cores in total);
- 24 GB RAM, each processor is equipped with 12 MB of cache memory;
- Intel Composer XE 2013 + MKL 11.1.1.

Unless specified otherwise, we used a tolerance of $\|A\|_F\cdot u$, where $u\approx 10^{-16}$ denotes the unit roundoff, in order to detect negligible columns in the reduction to anti-staircase form, and in order to declare an approximate eigenvalue as zero in the scalar algorithm from [10].

[1] The software is available from `http://anchp.epfl.ch/antitriangular`.

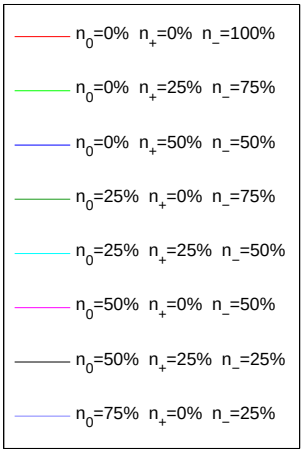


Figure 1: Percentages of zero (n_0), positive (n_+), and negative (n_-) eigenvalues of test matrices used in the numerical experiments.

To test for a wide variety of inertia, we generated a batch of test matrices A in the following way: Given n_0, n_+, and n_-, a diagonal matrix

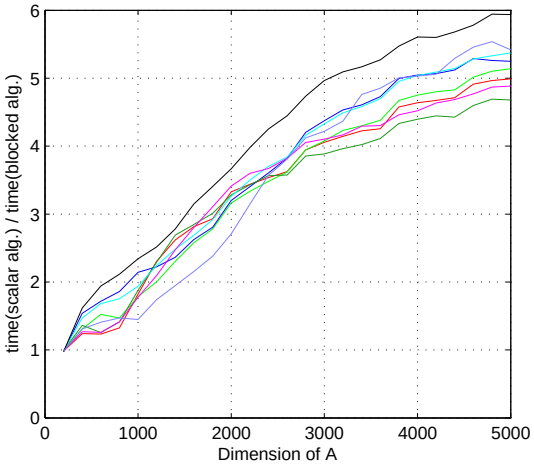
$$\Lambda = \begin{bmatrix} \Lambda_0 & & \\ & \Lambda_+ & \\ & & \Lambda_- \end{bmatrix} \in \mathbb{R}^{n\times n}, \quad n = n_0 + n_+ + n_-, \tag{3.1}$$

is created so that $\Lambda_0 \in \mathbb{R}^{n_0\times n_0}$ is zero, while $\Lambda_+ \in \mathbb{R}^{n_+\times n_+}, \Lambda_- \in \mathbb{R}^{n_-\times n_-}$ are diagonal matrices with diagonal entries chosen uniformly at random in the intervals $(0,1)$ and $(-1,0)$, respectively. Then a random orthogonal similarity transformation is applied to Λ in order to yield A. Figure 1 lists the different configurations of the inertia we used in the numerical experiments.

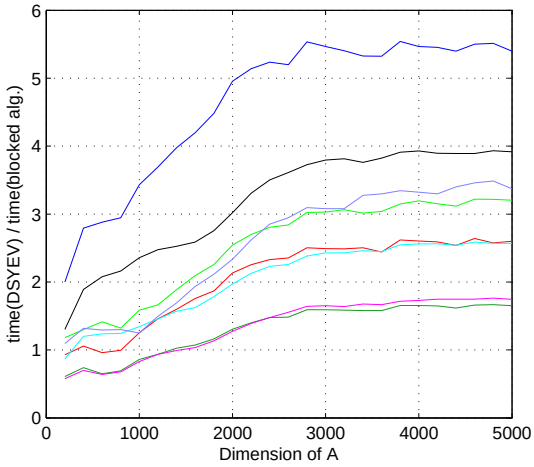
For each tested matrix A, the accuracy of the *computed* antitriangular factorization QTQ^T was verified by computing the backward error $\|A - QTQ^T\|_F$ and the orthogonality test $\|I - Q^TQ\|_F$. For all matrices we tested, these quantities have been observed to be of order $\mathcal{O}(\|A\|_F \cdot u)$ and $\mathcal{O}(\sqrt{n}\cdot u)$, respectively.

Serial performance. In our first set of experiments, we turned multithreading off and linked with the sequential version of MKL BLAS, so that only one of the eight cores was utilized. The obtained performance results are shown in Figure 2. Several observations can be made. Figure 2a reveals that the blocked algorithm is – depending on the inertia – between 4.5 and 6 times faster than the scalar algorithm for sufficiently large matrices. The blocked algorithm is also advantageous for matrices of fairly modest size; some speedup is obtained already for $n = 300$.

As discussed in the introduction, the antitriangular factorization is expected to be cheaper compared to the full spectral decomposition of a symmetric matrix. To verify whether this claim is actually reflected in the execution times, we compared our implementation with the LAPACK



(a) Ratios of execution times between scalar and blocked antitriangular algorithm.



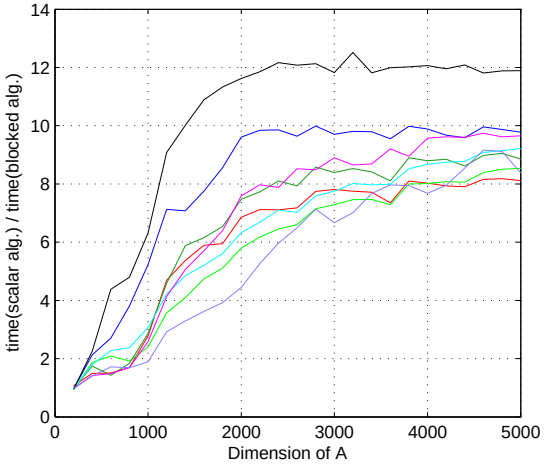
(b) Ratios of execution times between the LAPACK routine `DSYEV` and the blocked antitriangular algorithm.

Figure 2: Performance of blocked antitriangular algorithm with sequential BLAS. The different colors refer to different configuration of the inertia; see Figure 1.

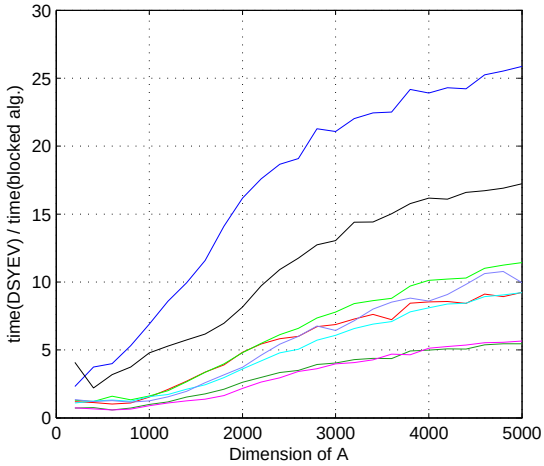
routine `DSYEV` for computing the spectral decomposition. As shown by Figure 2b, the *blocked* algorithm for the antitriangular factorization is faster, by a large margin, for sufficiently large matrices. Interestingly, the differences between the inertia are quite pronounced. The better speedups (relative to the `DSYEV` routine) are obtained for matrices with a small value of $n_2 = \max\{n_+, n_-\} - \min\{n_+, n_-\}$, that is, for matrices that have a small block X in their antitriangular factorization (1.1). A comparison between Figures 2a and 2b reveals that the *scalar* algorithm for the antitriangular factorization is actually often slower compared to `DSYEV`.

Performance on 8 cores. Multithreaded implementation of level 3 BLAS routines can exploit current computer architectures with multicore processors, thus making the blocked algorithm even more efficient. Figure 3a shows that linking with the multithreaded MKL BLAS library increases the speedup of the blocked algorithm relative to the scalar algorithm up to a factor of 12 on our machine. This effect is even more emphasized when comparing to the LAPACK implementation of `DSYEV`; see Figure 3b. Note, however, that neither our current implementation of the block algorithm nor LAPACK's `DSYEV` are particularly tuned to perform well on several cores. This becomes visible in Figure 3c, which shows a comparison with the MKL implementation of `DSYEV`. This implementation appears to be tailored to several cores and, consequently, the speedups attained by our block algorithm drop significantly.

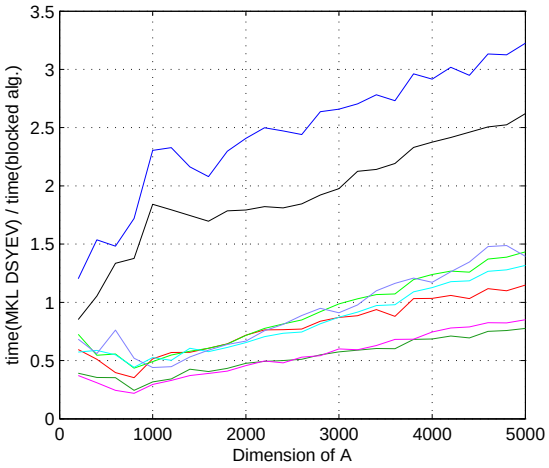
Recovery of inertia. It is important to remark that neither the scalar algorithm from [10] nor our block algorithm claim to be capable of reliably detecting the inertia in the presence of roundoff error. To attain such a reliability, one would probably need to combine these algorithms with



(a) Ratios of execution times between scalar and blocked antitriangular algorithm.

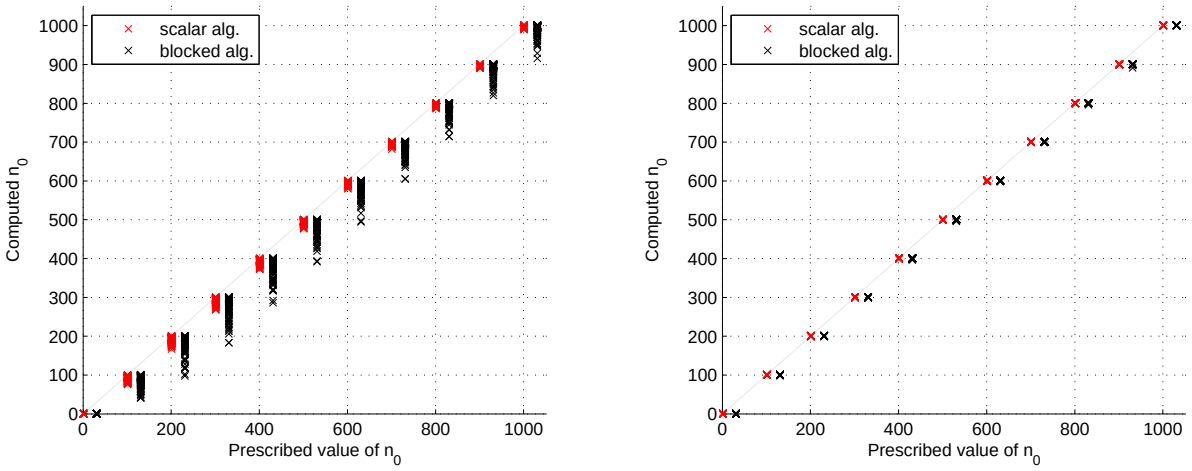


(b) Ratios of execution times between the LAPACK routine `DSYEV` and the blocked antitriangular algorithm.



(c) Ratios of execution times between the MKL routine `DSYEV` and the blocked antitriangular algorithm.

Figure 3: Performance of blocked antitriangular algorithm with multithreaded BLAS. The different colors refer to different configuration of the inertia; see Figure 1.



(a) Tolerance $\|A\|_F \cdot u$.

(b) Tolerance $100 \cdot \|A\|_F \cdot u$.

Figure 4: Computed versus prescribed values of n_0 for all test matrices, using two different tolerances. For each test matrix, $n_0 \in \{0, 100, 200, \ldots, 1000\}$ is prescribed and the corresponding computed value of n_0 is displayed by a cross marks: a red mark for the scalar algorithm and a black mark for the block algorithm. Note that, for readability, the black marks have been shifted to the right.

pivoting techniques, similar to existing pivoting techniques for the QR factorization [3, 5]. Still, it might be of interest to verify to which extent these algorithms can recover the inertia of our test matrices. For this purpose, we compared the computed values of n_0, n_+, n_- with the ones used for generating the test matrices (3.1). The value of n_0, the number of detected zero eigenvalues, is of particular interest, as this is critically influenced by deciding on the negligibility of certain entries during the algorithm. With the default tolerance, we observed that the computed n_0 is often (slightly) lower than the prescribed value. Some of the zero eigenvalues are falsely detected as non-zero eigenvalues. As shown in Figure 4a, this effect is particularly notable for the blocked algorithm. Setting a higher tolerance helps remedy this issue for the test matrices: Figure 4b shows the result of multiplying the tolerance with 100. Clearly, the computed n_0 now approximate the prescribed value of n_0 much better.

4 Conclusions

Our newly proposed block algorithm for reducing a symmetric matrix to antitriangular form has been found to significantly outperform the scalar algorithm. One critical issue, which has been revealed in our numerical experiments, is the safe detection of zero eigenvalues. To avoid this issue, which affects both the scalar and the block algorithms, pivoting strategies similar to the ones used in QR decompositions need to be employed.

References

[1] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, and P.R. Willems. Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. *Parallel Computing*, 37(12):783–794, 2011.

[2] P. Bientinesi, F. D. Igual, D. Kressner, M. Petschow, and E. S. Quintana-Ortí. Condensed forms for the symmetric eigenvalue problem on multi-threaded architectures. *Concurrency and Computation: Practice and Experience*, 23(7):694–707, 2011.

[3] C. H. Bischof and G. Quintana-Ortí. Computing rank-revealing QR factorizations of dense matrices. *ACM Trans. Math. Software*, 24(2):226–253, 1998.

[4] T. A. Davis. Algorithm 915, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization. *ACM Trans. Math. Software*, 38(1):Art. 8, 22, 2011.

[5] Z. Drmač and Z. Bujanović. On the failure of rank-revealing QR factorization software—a case study. *ACM Trans. Math. Software*, 35(2):Art. 12, 28, 2009.

[6] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, third edition, 1996.

[7] A. Haidar, H. Ltaief, and J. Dongarra. Parallel reduction to condensed forms for symmetric eigenvalue problems using aggregated fine-grained and memory-aware kernels. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 8:1–8:11, New York, NY, USA, 2011. ACM.

[8] A. Haidar, R. Solcà, M. Gates, S. Tomov, T. Schulthess, and J. Dongarra. Leading edge hybrid multi-GPU algorithms for generalized eigenproblems in electronic structure calculations. In J. Kunkel, T. Ludwig, and H. Meuer, editors, *Supercomputing*, volume 7905 of *Lecture Notes in Computer Science*, pages 67–80. Springer Berlin Heidelberg, 2013.

[9] N. Mastronardi and P. Van Dooren. Recursive approximation of the dominant eigenspace of an indefinite matrix. *J. Comput. Appl. Math.*, 236(16):4090–4104, 2012.

[10] N. Mastronardi and P. Van Dooren. The antitriangular factorization of symmetric matrices. *SIAM J. Matrix Anal. Appl.*, 34(1):173–196, 2013.

[11] N. Mastronardi and P. Van Dooren. On solving KKT linear systems with antitriangular matrices, 2013. TUM-IAS Workshop on Novel Numerical Methods, Munich Germany. Presentation available from `http://www.tum-ias.de/fileadmin/material_ias/pdf/NoNuMe/Presentations/Paul_van_Dooren.pdf`.

[12] N. Mastronardi and P. Van Dooren. An algorithm for solving the indefinite least squares problem with equality constraints. *BIT*, 54(1):201–218, 2014.

[13] N. Mastronardi, P. Van Dooren, and R. Vandebril. On solving KKT linear systems arising in model predictive control via recursive antitriangular factorization, 2014. Presentation at Householder Symposium XIX, June 8-13, Spa, Belgium.

[14] B. N. Parlett. *The Symmetric Eigenvalue Problem*. Society for Industrial and Applied Mathematics, 1998.

[15] J. Pestana and A. J. Wathen. The antitriangular factorization of saddle point matrices. *SIAM J. Matrix Anal. Appl.*, 35(2):339–353, 2014.