A Householder-based algorithm for Hessenberg-triangular reduction

Zvonimir Bujanović† Lars Karlsson‡ Daniel Kressner§

Abstract

The QZ algorithm for computing eigenvalues and eigenvectors of a matrix pencil $A - \lambda B$ requires that the matrices first be reduced to Hessenberg-triangular (HT) form. The current method of choice for HT reduction relies entirely on Givens rotations partially accumulated into small dense matrices which are subsequently applied using matrix multiplication routines. A non-vanishing fraction of the total flop count must nevertheless still be performed as sequences of overlapping Givens rotations alternatingly applied from the left and from the right. The many data dependencies associated with this computational pattern leads to inefficient use of the processor and makes it difficult to parallelize the algorithm in a scalable manner. In this paper, we therefore introduce a fundamentally different approach that relies entirely on (large) Householder reflectors partially accumulated into (compact) WY representations. Even though the new algorithm requires more floating point operations than the state of the art algorithm, extensive experiments on both real and synthetic data indicate that it is still competitive, even in a sequential setting. The new algorithm is conjectured to have better parallel scalability, an idea which is partially supported by early small-scale experiments using multi-threaded BLAS. The design and evaluation of a parallel formulation is future work.

1 Introduction

Given two matrices $A, B \in \mathbb{R}^{n \times n}$ the QZ algorithm proposed by Moler and Stewart [23] for computing eigenvalues and eigenvectors of the matrix pencil $A - \lambda B$ consists of three steps. First, a QR or an RQ factorization is performed to reduce $B$ to triangular form. Second, a Hessenberg-triangular (HT) reduction is performed, that is, orthogonal matrices $Q, Z \in \mathbb{R}^{n \times n}$ such that $H = Q^T A Z$ is in Hessenberg form (all entries below the sub-diagonal are zero) while $T = Q^T B Z$ remains in upper triangular form. Third, $H$ is iteratively (and approximately) reduced further to quasi-triangular form, which allows to easily determine the eigenvalues of $A - \lambda B$ and associated quantities.

During the last decade, significant progress has been made to speed up the third step, i.e., the iterative part of the QZ algorithm. Its convergence has been accelerated by extending aggressive early deflation from the QR [8] algorithm to the QZ algorithm [18]. Moreover, multi-shift techniques make sequential [18] as well as parallel [3] implementations perform well.

A consequence of the improvements in the iterative part, the initial HT reduction of the matrix pencil has become critical to the performance of the QZ algorithm. We mention in passing that this reduction also plays a role in aggressive early deflation and may thus become critical to the iterative part as well, at least in a parallel implementation [3] [12]. The original algorithm for HT reduction from [23] reduces $A$ to Hessenberg form (and maintains $B$ in triangular form) by performing $\Theta(n^2)$ Givens rotations. Even though progress has been made in [19] to accumulate these Givens rotations and apply them more efficiently using matrix multiplication, the need for propagating sequences of
rotations through the triangular matrix $B$ makes the sequential—but even more so the parallel—implementation of this algorithm very tricky.

A general idea in dense eigenvalue solvers to speed up the preliminary reduction step is to perform it in two (or more) stages. For a single symmetric matrix $A$, this idea amounts to reducing $A$ to banded form in the first stage and then further to tridiagonal form in the second stage. Usually called successive band reduction [6], this currently appears to be the method of choice for tridiagonal reduction; see, e.g., [4, 5, 13, 14]. However, this success story does not seem to carry over to the non-symmetric case, possibly because the second stage (reduction from block Hessenberg to Hessenberg form) is always an $\Omega(n^3)$ operation and hard to execute efficiently; see [20, 21] for some recent but limited progress. The situation is certainly not simpler when reducing a matrix pencil $A - \lambda B$ to HT form [19].

For the reduction of a single non-symmetric matrix to Hessenberg form, the classical Householder-based algorithm [10, 24] remains the method of choice. This is despite the fact that not all of its operations can be blocked, that is, a non-vanishing fraction of level 2 BLAS remains (approximately 20% in the form of one matrix–vector multiplication involving the unreduced part per column). Extending the use of (long) Householder reflectors (instead of Givens rotations) to HT reduction of a matrix pencil gives rise to a number of issues, which are difficult but not impossible to address. The aim of this paper is to describe how to satisfactorily address all of these issues. We do so by combining an unconventional use of Householder reflectors with blocked updates of RQ decompositions. We see the resulting Householder-based algorithm for HT reduction as a first step towards an algorithm that is more suitable for parallelization. We provide some evidence in this direction, but the parallelization itself is out of scope and is deferred to future work.

The rest of this paper is organized as follows. In Section 2, we recall the notions of (opposite) Householder reflectors and (compact) WY representations and their stability properties. The new algorithm is described in Section 3 and numerical experiments are presented in Section 4. The paper ends with conclusions and future work in Section 5.

2 Preliminaries

We recall the concepts of Householder reflectors, the little-known concept of opposite Householder reflectors, iterative refinement, and regular as well as compact WY representations. These concepts are the main building blocks of the new algorithm.

2.1 Householder reflectors

We recall that an $n \times n$ Householder reflector takes the form

$$H = I - \beta vv^T, \quad \beta = \frac{2}{v^Tv}, \quad v \in \mathbb{R}^n,$$

where $I$ denotes the $(n \times n)$ identity matrix. Given a vector $x \in \mathbb{R}^n$, one can always choose $v$ such that $Hx = \pm \|x\|_2 e_1$ with the first unit vector $e_1$; see [11, Sec. 5.1.2] for details.

Householder reflectors are orthogonal (and symmetric) and they represent one of the most common means to zero out entries in a matrix in a numerically stable fashion. For example, by choosing $x$ to be the first column of an $n \times n$ matrix $A$, the application of $H$ from the left to $A$ reduces the first column of $A$, that is, the trailing $n-1$ entries in the first column of $HA$ are zero.

2.2 Opposite Householder reflectors

What is less commonly known, and was possibly first noted in [26], is that Householder reflectors can be used in the opposite way, that is, a reflector can be applied from the right to reduce a column of a matrix. To see this, let $B \in \mathbb{R}^{n \times n}$ be invertible and choose $x = B^{-1}e_1$. Then the corresponding Householder reflector $H$ that reduces $x$ satisfies

$$(HB^{-1})e_1 = \pm \|B^{-1}e_1\|_2 e_1 \quad \Rightarrow \quad (BH)e_1 = \pm \frac{1}{\|B^{-1}e_1\|_2} e_1.$$
In other words, a reflector that reduces the first column of $B^{-1}$ from the left (as in $HB^{-1}$) also reduces the first column of $B$ from the right (as in $BH$). As shown in [13] Sec. 2.2, this method of reducing columns of $B$ is numerically stable provided that a backward stable method is used for solving the linear system $Bx = e_1$. More specifically, suppose that the computed solution $\hat{x}$ satisfies

$$(B + \Delta)\hat{x} = e_1, \quad \|\Delta\|_2 \leq \text{tol}$$

(1)

for some tolerance $\text{tol}$ that is small relative to the norm of $B$. Then the standard procedure for constructing and applying Householder reflectors [11] Sec. 5.1.3 produces a computed matrix $BH$ such that the trailing $n - 1$ entries of its first column have a 2-norm bounded by

$$\text{tol} + c_H \|B\|_2,$$

(2)

with $c_H \approx 12\epsilon$ and the unit round-off $\epsilon$. Hence, if a stable solver has been used and, in turn, $\text{tol}$ is not much larger than $\|B\|_2$, it is numerically safe to set these $n - 1$ entries to zero.

**Remark 2.1** In [18], it was shown that the case of a singular matrix $B$ can be addressed as well, by using an RQ decomposition of $B$. We favor a simpler and more versatile approach. To define the Householder reflector for a singular matrix $B$, we replace it by a non-singular matrix $\tilde{B} = B + \Delta$ with a perturbation $\Delta$ of norm $O(\|u\|_2)$. By (2), the Householder reflector based on the solution of $\tilde{B}x = e_1$ effects a transformation of $B$ such that the trailing $n - 1$ entries of its first column have norm $\text{tol} + \|\Delta\|_2 + c_H \|B\|_2$. Assuming that $\tilde{B}x = e_1$ is solved in a stable way, it is again safe to set these entries to zero.

### 2.3 Iterative refinement

The algorithm we are about to introduce operates in a setting for which the solver for $Bx = e_1$ is not always guaranteed to be stable. We will therefore use iterative refinement (see, e.g., [16] Ch. 12) to refine a computed solution $\hat{x}$:

1. Compute the residual $r = e_1 - B\hat{x}$.
2. Test convergence: Stop if $\|r\|_2/\|\hat{x}\|_2 \leq \text{tol}$.
3. Solve correction equation $Bc = r$ (with unstable method).
4. Update $\hat{x} \leftarrow \hat{x} + c$ and repeat from Step 1.

By setting $\Delta = r\hat{x}^T/\|\hat{x}\|_2^2$, one observes that (1) is satisfied upon successful completion of iterative refinement. In view of (2), we use the tolerance $\text{tol} = 2\epsilon\|B\|_F$ in our implementation.

The addition of iterative refinement to the algorithm improves its speed but is not a necessary ingredient. The algorithm has a robust fall-back mechanism that always ensures stability at the expense of slightly degraded performance. What is necessary, however, is to compute the residual to determine if the computed solution is sufficiently accurate.

### 2.4 Regular and compact WY representations

Let $I - \beta_i v_i v_i^T$ for $i = 1, 2, \ldots, k$ be Householder reflectors with $\beta_i \in \mathbb{R}$ and $v_i \in \mathbb{R}^n$. Setting

$$V = [v_1, \ldots, v_k] \in \mathbb{R}^{n \times k},$$

there is an upper triangular matrix $T \in \mathbb{R}^{k \times k}$ such that

$$\prod_{i=1}^k (I - \beta_i v_i v_i^T) = I - VTV^T.$$  

(3)

This so-called *compact WY representation* [25] allows for applying Householder reflectors in terms of matrix–matrix products (level 3 BLAS). The LAPACK routines DLARFT and DLARFB can be used to construct and apply compact WY representation, respectively.

In the case that all Householder reflectors have length $O(k)$ the factor $T$ in (3) constitutes a non-negligible contribution to the overall cost of applying the representation. In these cases, we instead use a *regular WY representation* [7] Method 2], which takes the form $I - VW^T$ with $W = VT^T$. 


3 Algorithm

Throughout this section, which is devoted to the description of the new algorithm, we assume that $B$ has already been reduced to triangular form, e.g., by an RQ decomposition. For simplicity, we will also assume that $B$ is non-singular (see Remark 2.1 for how to eliminate this assumption).

3.1 Overview

We first introduce the basic idea of the algorithm before going through most of the details.

The algorithm proceeds as follows. The first column of $A$ is reduced below the first sub-diagonal by a conventional reflector from the left. When this reflector is applied from the left to $B$, every column except the first fills in:

$$ (A, B) \leftarrow \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ \circ & x & x & x & x \\ \circ & \circ & x & x & x \\ \circ & \circ & \circ & x & x \\ \circ & \circ & \circ & \circ & x \end{pmatrix}, \begin{pmatrix} x & x & x & x \end{pmatrix}.$$

The second column of $B$ is reduced below the diagonal by an opposite reflector from the right, as described in Section 2.2. Note that the computation of this reflector requires the (stable) solution of a linear system involving the matrix $B$. When the reflector is applied from the right to $A$, its first column is preserved:

$$ (A, B) \leftarrow \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ \circ & x & x & x & x \\ \circ & \circ & x & x & x \\ \circ & \circ & \circ & x & x \\ \circ & \circ & \circ & \circ & x \end{pmatrix}, \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ \circ & x & x & x & x \\ \circ & \circ & x & x & x \\ \circ & \circ & \circ & x & x \\ \circ & \circ & \circ & \circ & x \end{pmatrix}.$$

Clearly, the idea can be repeated for the second column of $A$ and the third column of $B$, and so on:

$$ \begin{pmatrix} x & x & x & x & x \\ \circ & x & x & x & x \\ \circ & \circ & x & x & x \\ \circ & \circ & \circ & x & x \\ \circ & \circ & \circ & \circ & x \end{pmatrix}, \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ \circ & x & x & x & x \\ \circ & \circ & x & x & x \\ \circ & \circ & \circ & x & x \\ \circ & \circ & \circ & \circ & x \end{pmatrix}.$$

After a total of $n - 2$ steps, the matrix $A$ will be in upper Hessenberg form and $B$ will be in upper triangular form, i.e., the reduction to Hessenberg-triangular form will be complete. This is the gist of the new algorithm. The reduction is carried out by $n - 2$ conventional reflectors applied from the left to reduce columns of $A$ and $n - 2$ opposite reflectors applied from the right to reduce columns of $B$.

A naive implementation of the algorithm sketched above would require as many as $\Theta(n^4)$ operations simply because each of the $n - 2$ iterations requires the solution of a dense linear system with the unreduced part of $B$, whose size is roughly $n/2$ on average. In addition to this unfavorable complexity, the arithmetic intensity of the $\Theta(n^3)$ flops associated with the application of individual reflectors will be very low. The following two ingredients aim at addressing both of these issues:

1. The arithmetic intensity is increased for a majority of the flops associated with the application of reflectors by performing the reduction in panels (i.e., a small number of consecutive columns), delaying some of the updates, and using compact WY representations. The details resemble the blocked algorithm for Hessenberg reduction [10, 24].

2. To reduce the complexity from $\Theta(n^4)$ to $\Theta(n^3)$, we avoid applying reflectors directly to $B$. Instead, we keep $B$ in factored form during the reduction of a panel:

$$ \tilde{B} = (I - USU^T)^T B (I - VTV^T). \quad (4) $$
Since $B$ is triangular and the other factors are orthogonal, this reduces the cost for solving a system of equations with $\tilde{B}$ from $\Theta(n^3)$ to $\Theta(n^2)$. For reasons explained in Section 3.2.2 below, this approach is not always numerically backward stable. A fall-back mechanism is therefore necessary to guarantee stability. The new algorithm uses a fall-back mechanism that only slightly degrades the performance. Moreover, iterative refinement is used to avoid triggering the fall-back mechanism in many cases. After the reduction of a panel is completed, $\tilde{B}$ is returned to upper triangular form in an efficient manner.

### 3.2 Panel reduction

Let us suppose that the first $s-1$ (with $0 \leq s-1 \leq n-3$) columns of $A$ have already been reduced (and hence $s$ is the first unreduced column) and $B$ is in upper triangular form (i.e., not in factored form). The matrices $A$ and $B$ take the shapes depicted in Figure 1 for $j = s$. In the following, we describe a reflector-based algorithm that aims at reducing the panel containing the next $n_b$ unreduced columns of $A$. The algorithmic parameter $n_b$ should be tuned to maximize performance (see also Section 4 for the choice of $n_b$).

![Figure 1: Illustration of the shapes and sizes of the matrices involved in the reduction of a panel at the beginning of the $j$th step of the algorithm, where $j \in [s, s + n_b)$.](image)

#### 3.2.1 Reduction of the first column ($j = s$) of a panel

In the first step of a panel reduction, a reflector $I - \beta uu^T$ is constructed to reduce column $j = s$ of $A$. Except for entries in this particular column, no other entries of $A$ are updated at this point. Note that the first $j$ entries of $u$ are zero and hence the first $j$ columns of $\tilde{B} = (I - \beta uu^T)B$ will remain in upper triangular form. Now to reduce column $j+1$ of $\tilde{B}$, we need to solve, according to
Section 2.2: the linear system

\[ \begin{bmatrix} \tilde{B}_{j+1:n,j+1:n} \end{bmatrix} x = (I - \beta u_{j+1:n} u_{j+1:n}^T) B_{j+1:n,j+1:n} x = e_1. \]

The solution vector is given by

\[ x = B_{j+1:n,j+1:n}^{-1} (I - \beta u_{j+1:n} u_{j+1:n}^T) e_1 = B_{j+1:n,j+1:n}^{-1} \left( e_1 - \beta u_{j+1:n} u_{j+1:n}^T \right). \]

In other words, we first form the dense vector \( y \) and then solve an upper triangular linear system with \( y \) as the right-hand side. Both of these steps are backward stable [16] and hence the resulting Householder reflector \((I - \gamma vv^T)\) reliably yields a reduced \((j+1)\)th column in \((I - \beta uu^T) B (I - \gamma vv^T)\).

We complete the reduction of the first column of the panel by initializing 

\[ U \leftarrow u, \quad S \leftarrow [\beta], \quad V \leftarrow v, \quad T \leftarrow [\gamma], \quad Y \leftarrow \beta Av. \]

Remark 3.1: For simplicity, we assume that all rows of \( Y \) are computed during the panel reduction. In practice, the first few rows of \( Y = AVT \) are computed later on in a more efficient manner as described in [24].

3.2.2 Reduced of subsequent columns \((j > s)\) of a panel

We now describe the reduction of column \( j \in (s, s + nb) \), assuming that the previous \( k = j - s \geq 1 \) columns of the panel have already been reduced. This situation is illustrated in Figure 1. At this point, \( I - USU^T \) and \( I - VTV^T \) are the compact WY representations of the \( k \) previous reflectors from the left and the right, respectively. The transformed matrix \( \tilde{B} \) is available only in the factored form (4), with the upper triangular matrix \( B \) remaining unmodified throughout the entire panel reduction. Similarly, most of \( A \) remains unmodified except for the reduced part of the panel.

a) Update column \( j \) of \( A \). To prepare its reduction, the \( j \)th column of \( A \) is updated with respect to the \( k \) previous reflectors:

\[ A_{:,j} \leftarrow A_{:,j} - YV_{:,j}^T, \]
\[ A_{:,j} \leftarrow A_{:,j} - USU^T A_{:,j}. \]

Note that due to Remark 3.1, actually only rows \( s + 1 : n \) of \( A \) need to be updated at this point.

b) Reduce column \( j \) of \( A \) from the left. Construct a reflector \( I - \beta uu^T \) such that it reduces the \( j \)th column of \( A \) below the first sub-diagonal:

\[ A_{:,j} \leftarrow (I - \beta uu^T) A_{:,j}. \]

The new reflector is absorbed into the compact WY representation by

\[ U \leftarrow [U \quad u], \quad S \leftarrow \begin{bmatrix} S & -\beta SU^T u \\ 0 & \beta \end{bmatrix}. \]

c) Attempt to solve a linear system in order to reduce column \( j + 1 \) of \( \tilde{B} \). This step aims at (implicitly) reducing the \((j + 1)\)th column of \( \tilde{B} \) defined in (4) by an opposite reflector from the right. As illustrated in Figure 1, \( \tilde{B} \) is block upper triangular:

\[ \tilde{B} = \begin{bmatrix} \tilde{B}_{11} & \tilde{B}_{12} \\ 0 & \tilde{B}_{22} \end{bmatrix}, \quad \tilde{B}_{11} \in \mathbb{R}^{j \times j}, \quad \tilde{B}_{22} \in \mathbb{R}^{(n-j) \times (n-j)}. \]
To simplify the notation, the following description uses the full matrix $\tilde{B}$ whereas in practice we only need to work with the sub-matrix that is relevant for the reduction of the current panel, namely, $\tilde{B}_{s+1:n,s+1:n}$.

According to Section 2.2 we need to solve the linear system

$$\tilde{B}_{22}x = c, \quad c = e_1$$

in order to determine an opposite reflector from the right that reduces the first column of $\tilde{B}_{22}$. However, because of the factored form (4), we do not have direct access to $\tilde{B}_{22}$ and we therefore instead work with the enlarged system

$$\bar{B}y = \begin{bmatrix} \tilde{B}_{11} & \tilde{B}_{12} \\ 0 & \tilde{B}_{22} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 \\ c \end{bmatrix}.$$  \hspace{1cm} (6)

From the enlarged solution vector $y$ we can extract the desired solution vector $x = y_2 = \tilde{B}_{22}^{-1}c$. By combining (4) and the orthogonality of the factors with (6) we obtain

$$x = E^T (I - VTV^T)^T B^{-1} (I - USU^T) \begin{bmatrix} 0 \\ e_1 \end{bmatrix}, \quad \text{with} \quad E = \begin{bmatrix} 0 \\ I_{n-j} \end{bmatrix}.$$  \hspace{1cm} (7)

We are lead to the following procedure for solving (5):

1. Compute $c = (I - USU^T) \begin{bmatrix} 0 \\ c \end{bmatrix}$.
2. Solve the triangular system $B\tilde{y} = \tilde{c}$ by backward substitution.
3. Compute the enlarged solution vector $y = (I - VTV^T)^T \tilde{y}$.
4. Extract the desired solution vector $x = y_{j+1:n}$.

While only requiring $\Theta(n^2)$ operations, this procedure is in general not backward stable for $j > s$. When $\tilde{B}$ is significantly more ill-conditioned than $\tilde{B}_{22}$ alone, the intermediate vector $y$ (or, equivalently, $\tilde{y}$) may have a much larger norm than the desired solution vector $x$ leading to subtractive cancellation in the third step. As HT reduction has a tendency to move tiny entries on the diagonal of $B$ to the top left corner [26], we expect this instability to be more prevalent during the reduction of the first few panels (and this is indeed what we observe in the experiments in Section 4).

To test backward stability of a computed solution $\hat{x}$ of (5) and perform iterative refinement, if needed, we compute the residual $r = c - \tilde{B}_{22}\hat{x}$ as follows:

1. Compute $w = (I - VTV^T) \begin{bmatrix} 0 \\ \hat{x} \end{bmatrix}$.
2. Compute $w = Bw$.
3. Compute $w = (I - USU^T)w$.
4. Compute $r = c - w_{j+1:n}$.

We perform the iterative refinement procedure described in Section 2.3 as long as $\|r\|_2 > \text{tol} = 2u\|\tilde{B}\|_F$ but abort after ten iterations. In the rare case when this procedure does not converge, we prematurely stop the current panel reduction and absorb the current set of reflectors as described in Section 3.3 below. We then start over with a new panel reduction starting at column $j$. It is important to note that the algorithm is now guaranteed to make progress since when $k = 0$ we have $\tilde{B} = B$ and therefore solving (5) is backward stable.
d) Implicitly reduce column \( j+1 \) of \( \tilde{B} \) from the right. Assuming that the previous step computed an accurate solution vector \( x \) to (5), we can continue with this step to complete the implicit reduction of column \( j+1 \) of \( \tilde{B} \). If the previous step failed, then we simply skip this step. A reflector \( I - \gamma vv^T \) that reduces \( x \) is constructed and absorbed into the compact WY representation as in

\[
V \leftarrow [V \ v], \quad T \leftarrow \begin{bmatrix} T & -\gamma TV^T v \\ 0 & \gamma \end{bmatrix}.
\]

At the same time, a new column \( y \) is appended to \( Y \):

\[
y \leftarrow \gamma(Av - YV^Tv), \quad Y \leftarrow \begin{bmatrix} Y & y \end{bmatrix}.
\]

Note the common sub-expression \( V^Tv \) in the updates of \( T \) and \( Y \). Following Remark 3.1, the first \( s \) rows of \( Y \) are computed later in practice.

3.3 Absorption of reflectors

The panel reduction normally terminates after \( k = nb \) steps. In the rare event that iterative refinement fails, the panel reduction will terminate prematurely after only \( k \in [1, nb) \) steps. Let \( k \in [1, nb] \) denote the number of left and right reflectors accumulated during the panel reduction. The aim of this section is to describe how the \( k \) left and right reflectors are absorbed into \( A, B, Q, \) and \( Z \) so that the next panel reduction is ready to start with \( s \leftarrow s + k \).

We recall that Figure 1 illustrates the shapes of the matrices at this point. The following facts are central:

Fact 1. Reflector \( i = 1, 2, \ldots, k \) affects entries \( s+i: n \). In particular, entries \( 1:s \) are unaffected.

Fact 2. The first \( j-1 \) columns of \( A \) have been updated and their rows \( j+1:n \) are zero.

Fact 3. The matrix \( \tilde{B} \) is in upper triangular form in its first \( j \) columns.

In principle, it would be straightforward to apply the left reflectors to \( A \) and \( Q \) and the right reflectors to \( A \) and \( Z \). The only complications arise from the need to preserve the triangular structure of \( B \). To update \( B \) one would need to perform a transformation of the form

\[
B \leftarrow (I - USU^T)^T B(I - VTV^T).
\]

However, once this update is executed, the restoration of the triangular form of \( B \) (e.g., by an RQ decomposition) would have \( \Theta(n^3) \) complexity, leading to an overall complexity of \( \Theta(n^3) \). In order to keep the complexity down, a very different approach is pursued. This entails additional transformations of both \( U \) and \( V \) that considerably increase their sparsity. In the following, we use the term absorption (instead of updating) to emphasize the presence of these additional transformations, which affect \( A, Q, \) and \( Z \) as well.

3.3.1 Absorption of right reflectors

The aim of this section is to show how the right reflectors \( I - VTV^T \) are absorbed into \( A, B, \) and \( Z \) while (nearly) preserving the upper triangular structure of \( B \). When doing so we restrict ourselves to adding transformations only from the right due to the need to preserve the structure of the pending left reflectors, see \( \tilde{F} \).

a) Initial situation. We partition \( V \) as \( V = \begin{bmatrix} 0 \\ V_1 \end{bmatrix} \), where \( V_1 \) is a lower triangular \( k \times k \) matrix starting at row \( s+1 \) (Fact 1). Hence \( V_2 \) starts at row \( j+1 \) (recall that \( k = j-s \)). Our initial aim is to absorb the update

\[
B \leftarrow B(I - VTV^T) = B \begin{bmatrix} 0 \\ V_1 \end{bmatrix} T \begin{bmatrix} 0 & V_1^T & V_2^T \end{bmatrix}.
\]
The shapes of $B$ and $V$ are illustrated in Figure 2(a).

![Figure 2](image-url)

Figure 2: Illustration of the shapes of $B$ and $V$ when absorbing right reflectors into $B$: (a) initial situation, (b) after reduction of $V$, (c) after applying orthogonal transformations to $B$, (d) after partially restoring $B$.

**b) Reduce $V$.** We reduce the $(n-j) \times k$ matrix $V$ to lower triangular from via a sequence of QL decompositions from top to bottom. For this purpose, a QL decomposition of rows 1,...,$2k$ is computed, then a QL decomposition of rows $k+1,...,3k$, etc. After a total of $r \approx (n-j-k)/k$ such steps, we arrive at the desired form:

$$
\begin{align*}
\begin{pmatrix}
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\end{pmatrix}
\rightarrow
\begin{pmatrix}
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\end{pmatrix}
\rightarrow
\begin{pmatrix}
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\end{pmatrix}
\rightarrow
\begin{pmatrix}
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\end{pmatrix}
\rightarrow
\begin{pmatrix}
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\ddots \\
\end{pmatrix}
\end{align*}
$$

This corresponds to a decomposition of the form

$$V_2 = \hat{Q}_1 \cdots \hat{Q}_r \hat{L} \quad \text{with} \quad \hat{L} = \begin{bmatrix} 0 \\ \hat{L}_1 \end{bmatrix}, \quad (9)$$

where each factor $\hat{Q}_j$ has a regular WY representation of size at most $2k \times k$ and $\hat{L}_1$ is a lower triangular $k \times k$ matrix.

**c) Apply orthogonal transformations to $B$.** After multiplying (8) with $\hat{Q}_1 \cdots \hat{Q}_r$ from the right, we get

$$B \leftarrow B \begin{pmatrix} I - \begin{bmatrix} 0 \\ V_1 \\ V_2 \end{bmatrix} T \begin{bmatrix} 0 & V_1^T & V_2^T \end{bmatrix} \end{pmatrix} \begin{bmatrix} I \\ \hat{Q}_1 \cdots \hat{Q}_r \end{bmatrix},$$

$$= B \begin{pmatrix} I \\ \hat{Q}_1 \cdots \hat{Q}_r \end{pmatrix} - \begin{pmatrix} I \\ \hat{Q}_1 \cdots \hat{Q}_r \end{pmatrix} T \begin{bmatrix} 0 & V_1^T & \hat{L}^T \end{bmatrix},$$

$$= B \begin{pmatrix} I \\ \hat{Q}_1 \cdots \hat{Q}_r \end{pmatrix} \left( I - \begin{bmatrix} 0 \\ V_1 \\ \hat{L} \end{bmatrix} T \begin{bmatrix} 0 & V_1^T & \hat{L}^T \end{bmatrix} \right). \quad (10)$$

Hence, the orthogonal transformations nearly commute with the reflectors, but $V_2$ turns into $\hat{L}$. The shape of the correspondingly modified matrix $V$ is displayed in Figure 2(b).
Additionally exploiting the shape of $\hat{L}$, see (10), we update columns $s + 1 : n$ of $B$ according to (10) as follows:

1. $B_{:,j+1:n} \leftarrow B_{:,j+1:n} \hat{Q}_1 \cdots \hat{Q}_r$,
2. $W \leftarrow B_{:,s+1:j} V_1 + B_{:,n-k+1:n} \hat{L}_1$,
3. $B_{:,s+1:j} \leftarrow B_{:,s+1:j} - WTV_1^T$,
4. $B_{:,n-k+1:n} \leftarrow B_{:,n-k+1:n} - WT\hat{L}_1^T$.

In Step 1, the application of $\hat{Q}_1 \cdots \hat{Q}_r$ involves multiplying $B$ with $2k \times 2k$ orthogonal matrices (in terms of their WY representations) from the right. This will update columns $j + 1 : n$ from the left. Note that this will transform the structure of $B$ as illustrated in Figure 3. Step 3 introduces fill-in in columns $s + 1 : j$ while Step 4 does not introduce additional fill-in. In summary, the transformed matrix $B$ takes the form sketched in Figure 2(c).

d) Apply orthogonal transformations to $Z$. Replacing $B$ by $Z$ in (10), the update of columns $s + 1 : n$ of $Z$ takes the following form:

1. $Z_{:,j+1:n} \leftarrow Z_{:,j+1:n} \hat{Q}_1 \cdots \hat{Q}_r$,
2. $W \leftarrow Z_{:,s+1:j} V_1 + Z_{:,n-k+1:n} \hat{L}_1$,
3. $Z_{:,s+1:j} \leftarrow Z_{:,s+1:j} - WTV_1^T$,
4. $Z_{:,n-k+1:n} \leftarrow Z_{:,n-k+1:n} - WT\hat{L}_1^T$.

e) Apply orthogonal transformations to $A$. The update of $A$ is slightly different due to the presence of the intermediate matrix $Y = AVT$ and the panel which is already reduced. However, the basic idea remains the same. After post-multiplying with $\hat{Q}_1 \cdots \hat{Q}_r$, we get

$$A \leftarrow (A - Y [0 \ V_1^T \ V_2^T]) \begin{bmatrix} I & I \hat{Q}_1 \cdots \hat{Q}_r \end{bmatrix}$$

$$= A \begin{bmatrix} I & I \hat{Q}_1 \cdots \hat{Q}_r \end{bmatrix} - Y [0 \ V_1^T \ \hat{L}_1^T] .$$

The first $j - 1$ columns of $A$ have already been updated (Fact 2) but column $j$ still needs to be updated. We arrive at the following procedure for updating $A$:

1. $A_{:,j+1:n} \leftarrow A_{:,j+1:n} \hat{Q}_1 \cdots \hat{Q}_r$,
2. \( A_{:,j} \leftarrow A_{:,j} - Y(V_1)_{:,j}^T \).
3. \( A_{:,n-k+1:n} \leftarrow A_{:,n-k+1:n} - Y\hat{L}_{1:k}^T \).

**e) Partially restore the triangular shape of \( B \).** The absorption of the right reflectors is completed by reducing the last \( n - j \) columns of \( B \) back to triangular form via a sequence of RQ decompositions from bottom to top. This starts with an RQ decomposition of \( B_{n-k+1:n,n-2k+1:n} \).

After updating columns \( n-2k+1:n \) of \( B \) with the corresponding orthogonal transformation \( \tilde{Q}_1 \), we proceed with an RQ decomposition of \( B_{n-2k+1:n-k,n-3k+1:n-k} \), and so on, until all sub-diagonal blocks of \( B_{:,j+1:n} \) (see Figure 3) have been processed. The resulting orthogonal transformation matrices \( \tilde{Q}_1, \ldots, \tilde{Q}_r \) are multiplied into \( A \) and \( Z \) as well:

\[
A_{:,j+1:n} \leftarrow A_{:,j+1:n} \tilde{Q}_1^T \tilde{Q}_2^T \cdots \tilde{Q}_r^T, \\
Z_{:,j+1:n} \leftarrow Z_{:,j+1:n} \tilde{Q}_1^T \tilde{Q}_2^T \cdots \tilde{Q}_r^T.
\]

The shape of \( B \) after this procedure is displayed in Figure 2 (d).

**3.3.2 Absorption of left reflectors**

We now turn our attention to the absorption of the left reflectors \( I-USU^T \) into \( A, B, \) and \( Q \). When doing so we are free to apply additional transformations from left or right. Because of the reduced forms of \( A \) and \( B \), it is cheaper to apply transformations from the left. The ideas and techniques are quite similar to what has been described in Section 3.3.1 for absorbing right reflectors, and we therefore keep the following description brief.

**a) Initial situation.** We partition \( U \) as \( U = \begin{bmatrix} 0 \\ U_1 \\ U_2 \end{bmatrix} \), where \( U_1 \) is a \( k \times k \) lower triangular matrix starting at row \( s + 1 \) (Fact 1).

**b) Reduce \( U \).** We reduce the matrix \( U_2 \) to upper triangular form by a sequence of \( r \approx (n-j-k)/k \) QR decompositions as illustrated in the following diagram:

\[
\begin{bmatrix}
\text{x x x} \\
\text{x x x} \\
\text{x x x} \\
\text{x x x} \\
\text{x x x} \\
\text{x x x} \\
\text{x x x}
\end{bmatrix} \xrightarrow{\tilde{Q}_1} \begin{bmatrix}
\text{x x x} \\
\text{x x x} \\
\text{x x x} \\
\text{x x x} \\
\text{x x x} \\
\text{o o o} \\
\text{o o o}
\end{bmatrix} \xrightarrow{\tilde{Q}_2} \begin{bmatrix}
\text{x x x} \\
\text{r r r} \\
\text{o o o} \\
\text{o o o} \\
\text{o o o} \\
\text{o o o} \\
\text{o o o}
\end{bmatrix} \xrightarrow{\cdots} \begin{bmatrix}
\text{x x x} \\
\text{r r r} \\
\text{o o o} \\
\text{o o o} \\
\text{o o o} \\
\text{o o o} \\
\text{o o o}
\end{bmatrix} \xrightarrow{\tilde{Q}_r}.
\]

This corresponds to a decomposition of the form

\[
U_2 = \tilde{Q}_1 \cdots \tilde{Q}_r \hat{R} \quad \text{with} \quad \hat{R} = \begin{bmatrix} \hat{R}_1 \\ 0 \end{bmatrix},
\]

where \( \hat{R}_1 \) is a \( k \times k \) upper triangular matrix.

**c) Apply orthogonal transformations to \( B \).** We first update columns \( s+1:j \) of \( B \), corresponding to the “spike” shown in Figure 2 (d):

1. \( B_{s+1:j,s+1:j} \leftarrow B_{s+1:j,s+1:j} - U_1 S^T [U_1^T \ U_2^T] B_{s+1:n,s+1:j} \).
2. \( B_{j+1:n,s+1:j} \leftarrow 0. \)
d) Apply orthogonal transformations to $Q$ shown in Figure 3.

The triangular shape of updated and zero below row $j$.

f) Restore the triangular shape of $B$. At this point, the first $j$ columns of $B$ are in triangular form (see Part c), while the last $n-j$ columns are not and take the form shown in Figure 3, right. We reduce columns $j+1 : n$ of $B$ back to triangular form by a sequence of QR decompositions from top to bottom. This starts with a QR decomposition of $B_{j+1:n,j+1:k}$, and so on, until all subdiagonal blocks of $B_{:,j+1:n}$ have the form:

$$
\begin{pmatrix}
I & I \\
\tilde{Q}_1^T & \cdots & \tilde{Q}_n^T
\end{pmatrix}
\begin{pmatrix}
I - \begin{bmatrix} 0 \\ W_1 \\ W_2 \\ \vdots \\ U_n \\ R_1 \\
\end{bmatrix}
\end{pmatrix}
\begin{bmatrix}
0 & U_1^T & U_2^T \\
\end{bmatrix}
B
\begin{pmatrix}
I \\
\tilde{Q}_1^T & \cdots & \tilde{Q}_n^T
\end{pmatrix}
B.
$$

Additionally exploiting the shape of $\tilde{R}$, we update columns $j+1 : n$ of $B$ according to (12) as follows:

3. $B_{j+1:n,s+1:n} \leftarrow \tilde{Q}_1^T \cdots \tilde{Q}_n^T B_{j+1:n,s+1:n}$.

4. $W \leftarrow B_{s+1:j+k+1:n}^T \begin{bmatrix} U_1 \\ R_1 \\
\end{bmatrix}$.

5. $B_{s+1:j+k+1:n} \leftarrow B_{s+1:j+k+1:n} \begin{bmatrix} U_1 \\ R_1 \\
\end{bmatrix} S^T W^T$.

The triangular shape of $B_{j+1:n,j+1:n}$ is exploited in Step 3 and gets transformed into the shape shown in Figure 3, right.

d) Apply orthogonal transformations to $Q$. Replace $B$ with $Q$ in (12) and get

1. $Q_{:,j+1:n} \leftarrow Q_{:,j+1:n} \tilde{Q}_1 \cdots \tilde{Q}_n$.

2. $W \leftarrow Q_{:,s+1:j+k} \begin{bmatrix} U_1 \\ R_1 \\
\end{bmatrix}$.

3. $Q_{:,s+1:j+k} \leftarrow Q_{:,s+1:j+k} - W S \begin{bmatrix} U_1^T & \bar{R}_n^T \\
\end{bmatrix}$.

e) Apply orthogonal transformations to $A$. Exploiting that the first $j-1$ columns of $A$ are updated and zero below row $j$ (Fact 2), the update of $A$ takes the form:

1. $A_{j+1:n,j:n} \leftarrow \tilde{Q}_1^T \cdots \tilde{Q}_n^T A_{j+1:n,j:n}$.

2. $W \leftarrow A_{s+1:j+k+1:n}^T \begin{bmatrix} U_1 \\ R_1 \\
\end{bmatrix}$.

3. $A_{s+1:j+k+1:n} \leftarrow A_{s+1:j+k+1:n} \begin{bmatrix} U_1 \\ R_1 \\
\end{bmatrix} S^T W^T$.
been processed. The resulting orthogonal transformation matrices $\hat{Q}_1, \ldots, \hat{Q}_r$ are multiplied into $A$ and $Q$ as well:

$$A_{j+1:n,j:n} \leftarrow \hat{Q}_r^T \cdots \hat{Q}_2^T \hat{Q}_1^T A_{j+1:n,j:n},$$

$$Q_{:,j+1:n} \leftarrow Q_{:,j+1:n} \hat{Q}_1 \hat{Q}_2 \cdots \hat{Q}_r.$$

This completes the absorption of right and left reflectors.

### 3.4 Summary of algorithm

Summarizing the developments of this section, Algorithm 1 gives the basic form of our newly proposed Householder-based method for reducing a matrix pencil $A - \lambda B$, with upper triangular $B$, to Hessenberg-triangular form. The case of iterative refinement failures can be handled in different ways. In Algorithm 1 the last left reflector is explicitly undone, which is arguably the simplest approach. In our implementation, we instead use an approach that avoids redundant computations at the expense of added complexity. The differences in performance should be minimal.

**Algorithm 1**: $[H,T,Q,Z] = \text{HouseHT}(A,B)$

```
// Initialize
1 Q ← I; Z ← I;
2 Clear out V, T, U, S, Y;
3 k ← 0; // k keeps track of the number of delayed reflectors
// For each column to reduce in A
4 for j = 1 : n - 2 do
  // Reduce column j of A
5     Update column j of A from both sides w.r.t. the k delayed updates (see Section 3.2.2a);
6     Reduce column j of A with a new reflector $I - \beta uu^T$ (see Section 3.2.2b);
7     Augment $I - USU^T$ with $I - \beta uu^T$ (see Section 3.2.2b);
// Implicitly reduce column j + 1 of B
8     Attempt to solve the triangular system (see Section 3.2.2) to get vector x;
9     if the solve succeeded then
10        Reduce x with a new reflector $I - \gamma vv^T$ (see Section 3.2.2b);
11        Augment $I - VTV^T$ with $I - \gamma vv^T$ (see Section 3.2.2b);
12        Augment Y with $I - \gamma vv^T$ (see Section 3.2.2b);
13        k ← k + 1;
14     else
15        Undo the reflector $I - \beta uu^T$ by restoring the jth column of A, removing the last column of U, and removing the last row and column of S;
// Absorb all reflectors
16 if k = nb or the solve failed then
17       Absorb reflectors from the right (see Section 3.3.1);
18       Absorb reflectors from the left (see Section 3.3.2);
19       Clear out V, T, U, S, Y;
20       k ← 0;
// We are done
21 return $[A,B,Q,Z];$
```

The algorithm has been designed to require $\Theta(n^3)$ floating point operations (flops). Instead of a tedious derivation of the precise number of flops (which is further complicated by the occasional need for iterative refinement), we have measured this number experimentally; see Section 4. Based on empirical counting of the number of flops for both DGGHD3 and HouseHT on large random matrices (for which few iterative refinement iterations are necessary) we conclude that HouseHT requires
roughly $2.1 \pm 0.2$ times more flops than DGGRD3. Note that on more difficult problems this factor will increase.

### 3.5 Varia

In this section, we discuss a couple of additions that we have made to the basic algorithm described above. These modifications make the algorithm better at handling some types of difficult inputs (Section 3.5.1) and also slightly reduces the number of flops required for absorption of reflectors (Section 3.5.2).

#### 3.5.1 Preprocessing

A number of applications, such as mechanical systems with constraints [17] and discretized fluid flow problems [15], give rise to matrix pencils that feature a potentially large number of infinite eigenvalues. Often, many or even all of the infinite eigenvalues are induced by the sparsity of $B$. This can be exploited, before performing any reduction, to reduce the effective problem size for both the HT-reduction and the subsequent eigenvalue computation. As we will see in Section 3.5.2 such a preprocessing step is particularly beneficial to the newly proposed algorithm; the removal of infinite eigenvalues reduces the need for iterative refinement when solving linear systems with the matrix $B$.

We have implemented preprocessing for the case that $B$ has $\ell > 1$ zero columns. We choose an appropriate permutation matrix $Z_0$ such that the first $\ell$ columns of $BZ_0$ are zero. If $B$ is diagonal, we also set $Q_0 = Z_0$ to preserve the diagonal structure; otherwise we set $Q_0 = I$. Letting $A_0 = Q_0^T$ $AZ_0$, we compute a QR decomposition of its first $\ell$ columns: $A_0(:, 1: \ell) = Q_1 [A_{11}]$, where $Q_1$ is an $n \times n$ orthogonal matrix and $A_{11}$ is an $\ell \times \ell$ upper triangular matrix. Then

$$
A_1 = (Q_0 Q_1)^T A Z_0 = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}, \quad B_1 = (Q_0 Q_1)^T B Z_0 = \begin{bmatrix} 0 & B_{12} \\ 0 & B_{22} \end{bmatrix},
$$

where $A_{22}, B_{22} \in \mathbb{R}^{(n-\ell) \times (n-\ell)}$. Noting that the top left $\ell \times \ell$ part of $A_1 - \lambda B_1$ is already in generalized Schur form, only the trailing part $A_{22} - \lambda B_{22}$ needs to be reduced to Hessenberg-triangular form.

#### 3.5.2 Accelerated reduction of $V_2$ and $U_2$

As we will see in the numerical experiments in Section 3.5 below, Algorithm 1 spends a significant fraction of the total execution time on the absorption of reflectors. Inspired by techniques developed in [19] Sec. 2.2 for reducing a matrix pencil to block Hessenberg-triangular form, we now describe a modification of the algorithms described in Sections 3.3.1 and 3.3.2 that attains better performance by reducing the number of flops. We first describe the case when absorption takes place after accumulating nb reflectors and then briefly discuss the case when absorption takes place after an iterative refinement failure.

**Reduction of $V_2$.** We first consider the reduction of $V_2$ from Section 3.3.1 b) and partition $B$, $V_2$ into blocks of size nb $\times$ nb as indicated in Figure 3 (a). Recall that the algorithm for reducing $V_2$ proceeds by computing a sequence of QL decompositions of two adjacent blocks. Our proposed modification computes QL decompositions of $\ell \geq 3$ adjacent blocks at a time. Figure 3 (b)–(d) illustrates this process for $\ell = 3$, showing how the reduction of $V_2$ affects $B$ when updating it with the corresponding transformations from the right. Compared to Figure 3 the fill-in increases from overlapping $2\text{nb} \times 2\text{nb}$ blocks to overlapping $3\text{nb} \times 3\text{nb}$ blocks on the diagonal. For a matrix $V_2$ of size $n \times \text{nb}$, the modified algorithm involves around $(n - \text{nb})/3\text{nb}$ transformations, each corresponding to a WY representation of size $\text{nb} \times \text{nb}$. This compares favorably with the original algorithm which involves around $(n - \text{nb})/\text{nb}$ WY representations of size $2\text{nb} \times \text{nb}$. For $\ell = 3$ this implies that the overall cost of applying WY representations is reduced by between 10% and 25%, depending on how much of their triangular structure is exploited; see also [19]. These reductions quickly flatten out when increasing $\ell$ further. (Our implementation uses $\ell = 4$, which we found to be nearly optimal for the matrix sizes and computing environments considered in Section 4.) To
keep the rest of the exposition simple, we focus on the case $\ell = 3$; the generalization to larger $\ell$ is straightforward.

![Diagram](image)

**Figure 4:** Reduction of $V_2$ to lower triangular form by successive QL decompositions of $\ell = 3$ blocks and its effect on the shape of $B$. The diagonal patterns show what has been modified relative to the previous step. The thick lines aim to clarify the block structure. The red regions identify the sub-matrices of $V_2$ that will be reduced in the next step.

**Block triangular reduction of $B$ from the right.** After the reduction of $V_2$, we need to return $B$ to a form that facilitates the solution of linear systems with $B$ during the reduction of the next panel. If we were to reduce the matrix $B$ in Figure 4 (d) fully back to triangular form then the advantages of the modification would be entirely consumed by this additional computational cost. To avoid this, we reduce $B$ only to block triangular form (with blocks of size $2nb \times 2nb$) using the following procedure. Consider the RQ decomposition of an arbitrary $2nb \times 3nb$ matrix $C$:

$$C = RQ = \begin{bmatrix} 0 & R_{12} & R_{13} \\ 0 & 0 & R_{23} \end{bmatrix} \begin{bmatrix} Q_{11} & Q_{12} & Q_{13} \\ Q_{21} & Q_{22} & Q_{23} \\ Q_{31} & Q_{32} & Q_{33} \end{bmatrix}.$$

Compute an LQ decomposition of the first block row of $Q$:

$$E_1^TQ = [Q_{11} \quad Q_{12} \quad Q_{13}] = [D_{11} \quad 0 \quad 0] \tilde{Q},$$

where $E_1 = [I_k \quad 0 \quad 0]^T$. In other words, we have

$$E_1^TQ\tilde{Q}^T = [D_{11} \quad 0 \quad 0]$$

with $D_{11}$ lower triangular. Since the rows of this matrix are orthogonal and the matrix is triangular it must in fact be diagonal with diagonal entries $\pm1$. The first $nb$ columns of $Q\tilde{Q}^T$ are orthogonal and each therefore has unit norm. But since the top $nb \times nb$ block has $\pm1$ on the diagonal there is simply no room for any other non-zero entry on the same row and column of the matrix. In other words, the first block column of $Q\tilde{Q}^T$ must be $E_1D_{11}$. Thus, when applying $\tilde{Q}^T$ to $C$ from the right...
we obtain
\[
C\tilde{Q}^T = RQ\tilde{Q}^T = \begin{bmatrix} 0 & R_{12} & R_{13} \\ 0 & 0 & R_{23} \end{bmatrix} \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & \tilde{Q}_{22} & \tilde{Q}_{23} \\ 0 & \tilde{Q}_{32} & \tilde{Q}_{33} \end{bmatrix} = \begin{bmatrix} 0 & \tilde{C}_{12} & \tilde{C}_{13} \\ 0 & \tilde{C}_{22} & \tilde{C}_{23} \end{bmatrix}.
\]

Note that multiplying with \( \tilde{Q}^T \) from the right reduces the first block column of \( C \). Of course, the same effect could be attained with \( Q \) but the key advantage of using \( \tilde{Q} \) instead of \( Q \) is that \( \tilde{Q} \) consists of only \( nb \) reflectors with a WY representation of size \( 3nb \times nb \) compared with \( Q \) which consists of \( 2nb \) reflectors with a WY representation of size \( 3nb \times 2nb \). This makes it significantly cheaper to apply \( \tilde{Q} \) to other matrices.

Analogous constructions as those above can be made to efficiently reduce the last block row of a \( 3nb \times 2nb \) matrix by multiplication from the left. Replace \( C = RQ \) with \( C = QR \) and replace the LQ decomposition of \( E_1^TQ \) with a QL decomposition of \( QE_3 \). The matrix \( \tilde{Q}^TQ \) will have special structure in its last block row and column (instead of the first block row and column).

We apply the procedure described above\(^1\) to \( B \) in Figure 5 (a) starting at the bottom and obtain the shape shown in Figure 5 (b). Continuing in this manner from bottom to top eventually yields a block triangular matrix with \( 2nb \times 2nb \) diagonal blocks, as shown in Figure 5 (a)–(d).

![Figure 5](image_url)

**Figure 5:** Successive reduction of \( B \) to block triangular form. The diagonal patterns show what has been modified from the previous configuration. The thick lines aim to clarify the block structure. The red regions identify the sub-matrices of \( B \) that will be reduced in the next step.

**Reduction of \( U_2 \).** When absorbing reflectors from the left we reduce \( U_2 \) to upper triangular form as described in Section 3.3.2 b). The reduction of \( U_2 \) can be accelerated in much the same way as the reduction of \( V_2 \). However, since \( B \) is block triangular at this point, the tops of the sub-matrices of \( U_2 \) chosen for reduction must be aligned with the tops of the corresponding diagonal blocks of \( B \). Figure 6 gives a detailed example with proper alignment for \( \ell = 3 \). In particular, note that the first reduction uses a \( 2nb \times nb \) sub-matrix in order to align with the top of the first (i.e., bottom-most) diagonal block. Subsequent reductions use \( 3nb \times nb \) except the final reduction which is a special case.

**Block triangular reduction of \( B \) from the left.** The matrix \( B \) must now be reduced back to block triangular form. The procedure is analogous to the one previously described but this time the transformations are applied from the left, and, once again, we have to be careful with the alignment of the blocks. Starting from the initial configuration illustrated in Figure 7 a) for \( \ell = 3 \), the leading \( 2nb \times nb \) sub-matrix is fully reduced to upper triangular form. Subsequent steps of the reduction, illustrated in Figure 7 (b)–(d), use QR decompositions of \( 3nb \times 2nb \) sub-matrices to reduce the last \( nb \) rows of each block.

In Figure 4 (a) we assumed that the initial shape of \( B \) is upper triangular. This will be the case only for the first absorption. In all subsequent absorptions, the initial shape of \( B \) will be as

\(^1\) Our implementation actually computes RQ decompositions of full diagonal blocks (i.e., \( 3nb \times 3nb \) instead of \( 2nb \times 3nb \)). The result is essentially the same but the performance is slightly worse.
(a) Initial configuration.  
(b) 1st reduction.  
(c) 2nd reduction.  
(d) 3rd reduction.  
(e) 4th reduction.

Figure 6: Reduction of $U_2$ to upper triangular form by successive QR decompositions and its effect on the shape of $B$. The diagonal patterns show what has been modified from the previous configuration. The thick lines aim to clarify the block structure. The red regions identify the sub-matrices of $U_2$ that will be reduced in the next step.

Figure 7: Successive reduction of $B$ to block triangular form. The diagonal patterns show what has been modified from the previous configuration. The thick lines aim to clarify the block structure. The red regions identify the sub-matrix of $B$ that will be reduced in the next step.
are then reduced just as before (cf Figure 5) but this time the RQ decompositions will be computed from sub-matrices of size \(2nb \times (2nb + k)\), i.e., from sub-matrices with \(nb - k\) fewer columns than before. Note that the final WY transformations will involve only \(k\) reflectors (instead of \(nb\)), which is important for the sake of efficiency. Similarly, when reducing \(U_2\) the sub-matrices normally consist of \(2nb + k\) rows and the diagonal blocks of \(B\) will grow by \(k\) once more (cf Figure 6). The block triangular structure of \(B\) is finally restored by transformations consisting of \(k\) reflectors (cf Figure 7).

**Impact on Algorithm** The impact of the block triangular form in Figure 7(d) on Algorithm 1 is minor. Aside from modifying the way in which reflectors are absorbed (as described above), the only other necessary change is to modify the implicit reduction of column \(j + 1\) of \(B\) to accommodate a block triangular matrix. In particular, the residual computation will involve multiplication with a block triangular matrix instead of a triangular matrix and the solve will require block backwards substitution instead of regular backwards substitution. The block backwards substitution is carried out by computing an LU decomposition (with partial pivoting) once for each diagonal block and then reusing the decompositions for each of the \((up to)\) \(k\) solves leading up to the next wave of absorption.

4 Numerical Experiments

To test the performance of our newly proposed HouseHT algorithm, we implemented it in C++ and executed it on two different machines using different BLAS implementations. We compare with the LAPACK routine DGGHD3, which implements the block-oriented Givens-based algorithm from [19] and can be considered state of the art, as well as the predecessor LAPACK routine DGGHRD, which implements the original Givens-based algorithm from [23]. We created four test suites in order to explore the behavior of the new algorithm on a wide range of matrix pencils. For each test pair, the correctness of the output was verified by checking the resulting matrix structure and by computing \(\|H - QTAZ\|_F\) and \(\|T - QT^TBZ\|_F\).

The following table describes the computing environments used in our tests. The last row illustrates the relative performance of the machine/BLAS combinations, measuring the timing of the DGGHD3 routine for a random pair of dimension 4000, and rescaling so that the time for pascal with MKL is normalized to 1.00.

<table>
<thead>
<tr>
<th>machine name</th>
<th>pascal</th>
<th>kebnekaise</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor</td>
<td>2x Intel Xeon E5-2690v3 (12 cores each, 2.6GHz)</td>
<td>2x Intel Xeon E5-2690v4 (14 cores each, 2.6GHz)</td>
</tr>
<tr>
<td>RAM</td>
<td>256GB</td>
<td>128GB</td>
</tr>
<tr>
<td>operating system</td>
<td>Centos 7.4</td>
<td>Ubuntu 16.04</td>
</tr>
<tr>
<td>BLAS library</td>
<td>MKL 11.3.5, icpc 16.0.3</td>
<td>OpenBLAS 0.2.19, g++ 4.8.5</td>
</tr>
<tr>
<td>compiler</td>
<td>icpc 16.0.3</td>
<td>g++ 4.8.5</td>
</tr>
<tr>
<td>relative timing</td>
<td>1.00</td>
<td>1.38</td>
</tr>
</tbody>
</table>

For each computing environment, the optimal block sizes for HouseHT and DGGHD3 were first estimated empirically and then used in all four test suites. Unless otherwise stated, we use only a single core and link to single-threaded BLAS. All timings include the accumulation of orthogonal transformations into \(Q\) and \(Z\).

**Test Suite 1: Random matrix pencils.** The first test suite consists of random matrix pencils. More specifically, the matrix \(A\) has normally distributed entries while the matrix \(B\) is chosen as the triangular factor of the QR decomposition of a matrix with normally distributed entries. This test suite is designed to illustrate the behavior of the algorithm for a “non-problematic” input with no infinite eigenvalues and a fairly well-conditioned matrix \(B\). For such inputs, the HouseHT algorithm typically needs no iterative refinement steps when solving linear systems.

Figure 8a displays the execution time of HouseHT divided by the execution time of DGGHD3 for the different computing environments. The new algorithm has roughly the same performance as DGGHD3, being from about 20% faster to about 35% slower than DGGHD3, depending on the
machine/BLAS combination. Both algorithms exhibit far better performance than the LAPACK routine DGGHRD, which makes little use of BLAS3 due to its non-blocked nature.

Figure 8 shows the flop-rates of HouseHT and DGGHD3 for the pascal machine with MKL BLAS. Although the running times are about the same, the new algorithm computes about twice as many floating point operations, so the resulting flop-rate is about two times higher than DGGHD3. The flop-counts were obtained during the execution of the algorithm by interposing calls to the LAPACK and BLAS routines and instrumenting the code.

![Figure 8](image)

(a) Execution time of HouseHT and DGGHRD relative to execution time of DGGHD3.

(b) Flop-rate of HouseHT and DGGHD3 on the pascal machine with MKL BLAS.

Figure 8: Single-core performance of HouseHT for randomly generated matrix pencils (Test Suite 1).

The following table shows the fraction of the time that HouseHT spends in the three most computationally expensive parts of the algorithm. The results are from the pascal machine with MKL BLAS and \( n = 8000 \).

<table>
<thead>
<tr>
<th>part of HouseHT</th>
<th>% of total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>solving systems with ( B ), computing residuals</td>
<td>22.82%</td>
</tr>
<tr>
<td>absorption of reflectors</td>
<td>57.40%</td>
</tr>
<tr>
<td>assembling ( Y = AVT )</td>
<td>19.61%</td>
</tr>
</tbody>
</table>

HouseHT spends as much as 92.60% of its flops (and 52.77% of its time) performing level 3 BLAS operations, compared to DGGHD3 which spends only 65.35% of its flops (and 18.33% of its time) in level 3 BLAS operations.

**Test Suite 2: Matrix pencils from benchmark collections.** The purpose of the second test suite is to demonstrate the performance of HouseHT for matrix pencils originating from a variety of applications. To this end, we applied HouseHT and DGGHD3 to a number of pencils from the benchmark collections [1, 9, 22]. Table 1 displays the obtained results for the pascal machine with MKL BLAS. When constructing the Householder reflector for reducing a column of \( B \) in HouseHT, the percentage of columns that require iterative refinement varies strongly for the different examples. Typically, at most one or two steps of iterative refinement are necessary to achieve numerical stability. It is important to note that we did not observe a single failure, all linear systems were successfully solved in less than 10 iterations.

As can be seen from Table 1, HouseHT brings little to no benefit over DGGHD3 on a single core of pascal with MKL. A first indication of the benefits HouseHT may bring for several cores is seen by comparing the third and the fourth columns of the table. By switching to multithreaded BLAS and using eight cores, then for sufficiently large matrices HouseHT becomes significantly faster than DGGHD3.

**Remark 4.1** Percentage of columns for which an extra IR step is required depends slightly on the machine/BLAS combination due to different block size configurations; typically, it does not differ by
much, and difficult examples remain difficult. The performance of HouseHT vs DGGHD3 does vary much, as Figure 8a suggests. We briefly summarize the findings of the numerical experiments: when the algorithms are run on a single core, the ratios shown in the second column of the above table are, on average, about 20% smaller for pascal/OpenBLAS, about 5% larger for kebnekaise/MKL, and about 28% larger for kebnekaise/OpenBLAS. When the algorithms are run on 8 cores, the HouseHT algorithm gains more and more advantage over DGGHD3 with the increasing matrix size, regardless of the machine/BLAS combination. On average, the ratios shown in the third column are about 38% smaller for pascal/OpenBLAS, about 14% larger for kebnekaise/OpenBLAS, and about 50% larger for kebnekaise/MKL.

**Test Suite 3: Potential for parallelization.** The purpose of the third test is a more detailed exploration of the potential benefits the new algorithm may achieve in a parallel environment. For this purpose, we link HouseHT with a multithreaded BLAS library. Let us emphasize that this is purely indicative. Implementing a truly parallel version of the new algorithm, with custom tailored parallelization of its different parts, is subject to future work. Figure 9a shows the speedup of the HouseHT algorithm achieved relative to DGGHD3 for an increasing number of cores. We have used $8000 \times 8000$ matrix pencils, generated as in Test Suite 1. As shown in Figure 9b, the performance of DGGHD3, unlike the new algorithm, barely benefits from switching to multithreaded BLAS.
Figure 9: Performance of HouseHT for randomly generated matrix pencils of dimension 8000 (Test Suite 3) when using multithreaded BLAS.

Test Suite 4: Saddlepoint matrix pencils. The final test suite consists of matrix pencils designed to be particularly unfavorable for HouseHT. Let

\[ A - \lambda B = \begin{bmatrix} X & Y \\ Y^T & 0 \end{bmatrix} - \lambda \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}, \]

with a random positive definite matrix \( X \) and a random (full-rank) matrix \( Y \) with sizes chosen such that \( X \) is 3/4th the size of \( A \). The matrix \( B \) is split accordingly. For such matrix pencils, with many infinite eigenvalues, we expect that HouseHT will struggle with solving linear systems, requiring many steps of iterative refinement and being forced to prematurely absorb reflectors. This is, up to a point, what happens when we run the test suite. In Table 2 we see that HouseHT may be up to 4 times slower than DGGHD3 (on pascal/MKL) for smaller-sized matrix pencils. For about 5% of the columns the linear systems cannot be solved in a stable manner, even with the help of iterative refinement. In turn, the reflectors have to be repeatedly absorbed prematurely. However, in all of these cases, HouseHT still manages to successfully produce the Hessenberg-triangular form to full precision.

For example, for \( n = 4000 \), there are 67 columns for which the linear system cannot be solved with 10 steps of iterative refinement. The failure happens more frequently in the beginning of the algorithm: it occurs 14 times within the first 100 columns, only 6 times after the 700th column, and the last occurrence is at the 1082nd column. The same observation can be made for columns requiring extra (but fewer than 10) steps of IR; the last such column is the 2105th column.

For this, and many similar test cases, using the preprocessing of the zero columns as described in Section 3.5.1 may convert a difficult test case to a very easy one. The numbers in parentheses in Table 2 show the effect of preprocessing for the saddlepoint pencils. Note that we do not preprocess the input for DGGHD3 (which would benefit from it as well). With preprocessing on, there is barely any need for iterative refinement despite the fact that it does not remove all of the infinite eigenvalues.

5 Conclusions and future work

We described a fundamentally new algorithm for Hessenberg-triangular reduction. The algorithm relies on the unconventional and little-known possibility to use a Householder reflector applied from the right to reduce a matrix column [26]. In contrast, the current state of the art is entirely based on Givens rotations [19].

We explained that the algorithm is backwards stable but its performance may degrade when presented with a difficult problem. Extensive experiments on synthetic as well as real examples...
suggest that the performance degradation phenomenon is not a significant concern in practice and that simple preprocessing measures can be applied to greatly reduce the negative effects.

Compared with the state of the art [19], the new algorithm requires a small constant factor more floating point arithmetic operations but on the other hand these operations occur in computational patterns that allow for faster flop rates. In other words, the negative impact of the additional flops is at least partially counteracted by the increased speed by which these flops can be performed. Experiments suggest that the sequential performance of the new algorithm is comparable to the state of the art.

The primary motivation for developing the new algorithm was its potential for greater parallel scalability than the state-of-the-art parallel algorithm [2]. Early experiments using multi-threaded BLAS support this idea. Therefore, the design and evaluation of a task-based parallel algorithm is our next step.

Acknowledgments

This research was conducted using the resources of High Performance Computing Center North (HPC2N). Part of this work was done while the first author was a postdoctoral researcher at École polytechnique fédérale de Lausanne, Switzerland.

References


