SIAM J. MATRIX ANAL. APPL.
Vol. 28, No. 2, pp. 398–424

THE SNAP-BACK PIVOTING METHOD FOR SYMMETRIC BANDED INDEFINITE MATRICES*

DROR IRONY[†] AND SIVAN TOLEDO[†]

Abstract. The four existing stable factorization methods for symmetric indefinite matrices suffer serious defects when applied to banded matrices. Partial pivoting (row or column exchanges) maintains a band structure in the reduced matrix and the factors, but destroys symmetry completely once an off-diagonal pivot is used. Two-by-two block pivoting and Gaussian reduction to tridiagonal (Aasen's algorithm) maintain symmetry at all times, but quickly destroy the band structure in the reduced matrices. Orthogonal reductions to tridiagonal maintain both symmetry and the band structure, but are too expensive for linear-equation solvers.

We propose a new pivoting method, which we call *snap-back* pivoting. When applied to banded symmetric matrices, it maintains the band structure (like partial pivoting does), it keeps the reduced matrix symmetric (like 2-by-2 pivoting and reductions to tridiagonal), and it is fast.

Snap-back pivoting reduces the matrix to a diagonal form using a sequence of elementary elimination steps, most of which are applied symmetrically from the left and from the right (but some are applied unsymmetrically).

In snap-back pivoting, if the next diagonal element is too small, the next pivoting step might be unsymmetric, leading to asymmetry in the next row and column of the factors. But the reduced matrix snaps back to symmetry once the next step is completed.

Key words. symmetric-indefinite matrices, pivoting, banded matrices, matrix factorizations, element growth

AMS subject classifications. 15A06, 15A23, 65F05

DOI. 10.1137/040610106

1. Introduction.

We propose a new method for the direct solution of a linear system of equations $Ax = b$ where A is an n-by-n banded symmetric indefinite matrix with half bandwidth m. The method performs $O(nm^2)$ work. Our method reduces A to a diagonal matrix by eliminating one or two rows and columns in each step. After each elimination step, the reduced matrix is a banded symmetric matrix with half bandwidth at most $2m$. Although at each step the reduced matrix is symmetric, the factors corresponding to steps in which we eliminate two columns together may not be symmetric. The algorithm requires a mixture of symmetric and unsymmetric data. The element growth in the reduced matrices is bounded by 4^{n-1}. Elements of the factors are bounded by 3 or by the elements of the reduced matrices. Our method achieves these goals using an intricate elimination scheme that employs both Gaussian row and column operations and Givens rotations.

Our method reduces the matrix to a diagonal one or two rows/columns at a time. If the next diagonal element is large, an ordinary symmetric Gaussian elimination step will reduce the next row and column. Such a step adds a column to the left factor and its transpose to the right factor. Since a symmetric matrix is subtracted from the symmetric trailing submatrix, it remains symmetric. If the next diagonal

[*] Received by the editors June 16, 2004; accepted for publication (in revised form) by N. J. Higham November 30, 2005; published electronically May 26, 2006. This research was supported in part by an IBM Faculty Partnership Award, by grants 572/00 and 848/04 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), and by grant 2002261 from the United-States-Israel Binational Science Foundation.

http://www.siam.org/journals/simax/28-2/61010.html

[†] School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel (irony@tau.ac.il, stoledo@tau.ac.il).

element is too small, we use a more complex elimination step. During that step, the trailing submatrix becomes unsymmetric, but it snaps back to symmetry at the end of the step. That is why we call our method *snap-back* pivoting. Such a complex elimination step contributes one row to the right factor that is not a transpose of a column in the left factor. Therefore, the amount of asymmetry in the factors depends on the number of these complex elimination steps.

The paper is organized as follows. The next section surveys existing factorization methods that can be applied to banded symmetric matrices. Section 3 describes the new method; the presentation is not specific to banded matrices. Section 4 shows how to adapt the general method to preserve the band structure. Section 5 bounds the element growth in the reduced matrices and in the factors. Section 6 presents the results of experiments that we conducted to assess the performance and stability of our method. We present our conclusions from this research and list open problems in section 7.

2. Related work.

Several existing direct factorization methods can reduce a banded symmetric matrix to a diagonal, tridiagonal, or block-diagonal form. If the matrix is indefinite, either some form of pivoting or an orthogonal reduction must be employed to ensure stability.

Gaussian elimination with partial pivoting (GEPP) maintains a band structure in the reduced matrices, but not symmetry. The first off-diagonal pivot that is chosen completely destroys symmetry in the reduced matrix, so from that row/column until the end of the matrix, both the reduced matrix and the factors become unsymmetric. The loss of symmetry results in doubling the computational and storage costs. On the other hand, GEPP maintains a band structure in the reduced matrix and in the factors. No matter how many off-diagonal pivots are chosen, the reduced matrix and the right upper-triangular factor have half bandwidth at most $2m$. The left lower triangular factor, and also the lower triangular part of the reduced matrices, maintain half bandwidth m.

Two other factorization techniques employ pivoting but maintain symmetry in the factors and in the reduced matrices. One technique, which is used in several algorithms [4, 6, 7, 8, 14], uses 2-by-2 block pivots. In other words, it reduces the matrix to a block diagonal form with 1-by-1 and 2-by-2 blocks. This technique usually cannot be applied to banded matrices, because every 2-by-2 pivot might increase the half-bandwidth by $m-2$. Therefore, the half bandwidth can quickly expand. This bandwidth expansion can lead to catastrophic increase in the computational and storage requirements of the algorithm. Still, in some special cases this technique can be applied to banded matrices. Bunch and Kaufman showed that if $m \leq 2$, then one of the variants of their algorithm (variant D) maintains the band structure [4]. They also showed that the number of 2-by-2 pivots is bounded by the minimum of the number of positive and the number of negative eigenvalues of A. This observation led Jones and Patrick [14] to propose that the Bunch–Kaufman algorithm can be used when A has very few negative or very few positive eigenvalues; in such cases, the bandwidth expansion might still be preferable to GEPP. A combination of 1-by-1 and 2-by-2 pivoting steps is also used in multifrontal factorization algorithms for general sparse matrices [6, 7, 8], but without any a priori bound on the fill that pivoting might cause.

In the 2-by-2 pivoting methods of Bunch and Kaufman, the element growth in the reduced matrices is bounded by 2.57^{n-1}, but the magnitude of elements in the lower-triangular factor is not bounded by the magnitude of elements in the reduced matrices. (In most other factorization methods, the elements of the factors are bounded by 1 or by the elements of the reduced matrices.) This fact, along with some disappointing practical experience, led some to question the reliability of these methods [2]. However, the methods were found to be formally backward stable [11, 12].

Another pivoting symmetric factorization technique reduced A to a tridiagonal form using a sequence of 1-by-1 but off-diagonal pivots. The resulting tridiagonal matrix is subsequently factored using GEPP, but the cost of that step is usually insignificant. This technique was initially proposed for dense matrices by Parlett and Reid [17], and later improved by Aasen [1]. The trouble with these methods is that they can also quickly expand the bandwidth, so they have never been applied to banded matrices.

Another way to reduce a banded symmetric matrix to a tridiagonal form is to employ orthonormal transformations. This technique is used in eigensolvers, which can only use orthonormal transformations. Algorithms that use this technique [3, 5, 15, 16, 18, 19] annihilate in every step one row and column, or even just one element. The annihilation creates fill in the form of a bulge, which expands the band. To avoid gradual band expansion, the algorithm then "chases" the bulge down the matrix, so that the matrix regains the original half bandwidth m before the next annihilation step. The trouble with these approaches is that chasing the bulge is expensive, so the total computational cost of these algorithm is proportional to n^2m. By contrast, the cost of GEPP, as well as the cost of our algorithm, is only nm^2. For $m \ll n$, the difference can be enormous.

From the band-preservation viewpoint, the difference between 2-by-2 block algorithms and the orthonormal algorithms lies not in the elementary transformations that are used, but in whether the bulge is chased or not. In the 2-by-2 block algorithms of Bunch and Kaufman, as well as in the Parlett–Reid and Aasen algorithms, the bulge is not chased, so the band expands without a bound. In the orthonormal reductions to tridiagonal, the bulge is chased, so the bandwidth remains bounded, but at an unacceptable cost to linear-equation solvers. (The use of orthonormal transformations also causes the bulge to always appear, whereas in the other factorization algorithms the size of the bulge depend on the choice of pivots, but that is not the main point here.) We believe that a bulge-chasing variant of the other algorithms can also be developed, but that its cost will still be proportional to n^2m, just like the orthonormal reductions.

3. Snap-back pivoting.

This section presents the new pivoting strategy that we proposed, called *snap-back* pivoting. We mostly ignore the issue of the bandwidth of the matrix in this section and focus instead on the mathematical and algorithmic ideas. Bandwidth issues do narrow down some of the algorithmic design space that we explore; the text explicitly highlights these cases. The next section explains how to apply snap-back pivoting to banded matrices.

Let A be a symmetric n-by-n nonsingular matrix. We show how to perform a sequence of elementary elimination steps that reduce the matrix to a diagonal form. The sequence is *quasi-symmetric*: the trailing uneliminated submatrix is always symmetric. Most, but not all, of the elementary transformations are applied symmetrically from the left and from the right. We use three classes of elementary transformations: Gauss transforms, Givens rotations, and permutations. Each elimination step, which often consists of applying multiple elementary transformations, eliminates the offdiagonals in either one row and column or in two. The first row and column are always eliminated; in some cases, another row and columns, not necessarily the second, are also eliminated.

Our method uses three kinds of elimination steps. The next subsection provides a high-level overview of the possible elimination steps and the one that follows specifies the transformation matrices in detail. To further clarify the algorithm, we have produced a Matlab implementation (of the banded variant, described below). This implementation is freely available.[1] It stops after every elimination step, shows the nonzero structure of the reduced matrix, and prints the class of transformation that it just applied. It also produces transformation matrices that correspond exactly to the notation in this section.

3.1. An overview of the elimination process.

We begin the presentation with a high-level overview of the structure of the algorithm.

Elimination steps of the first kind. When a_{11} is nonzero (to avoid growth it will need to be not only nonzero, but large; we ignore numerical issues for now), we can eliminate the offdiagonals in the first row and column using ordinary symmetric Gaussian elimination,

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & & & & \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \end{bmatrix} = L^{-1}AL^{-T}.$$

In this notation, × symbols denote nonzero elements in a symmetric matrix (that is, if a_{ij} is denoted by a ×, then $a_{ij} = a_{ji}$, and they are not necessarily zero). Elements that are blank are zeros. The matrix product on the right means that we transform the matrix on the left, A, to the matrix on the right by multiplying A by a lower triangular matrix L^{-1} and its transpose. We also use this kind of elimination step, with L an identity matrix, when the first row and column are identically zero. We defer the exact specification of the transformation matrices to the next subsection; here L is an elementary Gauss transform.

Elimination steps of the second kind. When a_{11} is zero (or simply too small), our method uses a more elaborate sequence of elementary transformations. We begin with a series of either Givens or Gauss transforms that eliminate all the nonzeros in the first column except for the last,

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & & & & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} = Y^{-1}AY^{-T}.$$

Next, we eliminate element $(n, 1)$ in the reduced matrix, which we shall show later is always nonzero, using a Givens rotation that transforms the first and last rows. If element $\left[Y^{-1}AY^{-T}\right]_{11} = 0$, the Givens rotation is essentially a row exchange, so we

[1] The code is available from http://www.tau.ac.il/∼stoledo/research.html.

get the following form:

$$\begin{bmatrix} \times & & & & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & \otimes & \otimes & \otimes & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & & & & \times \end{bmatrix} = G^{-1}Y^{-1}AY^{-T},$$

where

$$G = \begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ -1 & \cdots & 0 \end{bmatrix}.$$

The symbol $\otimes$ denotes a nonsymmetric nonzero, i.e., an a_{ij} that might be nonzero and might be different from a_{ji}. Here, when a_{ij} is denoted by a $\otimes$, $a_{ji} = 0$, so it is left blank. When both a_{ij} and a_{ji} might be nonzeros and different from each other, we denote one by $\otimes$ and the other by $\oplus$ to emphasize the lack of symmetry, as in the next equation. If $\left[Y^{-1}AY^{-T}\right]_{11} \neq 0$, then we get another form, which turns out to be simpler,

$$\begin{bmatrix} \times & & & & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & \otimes & \otimes & \otimes & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \oplus & \oplus & \oplus & \times \end{bmatrix} = G^{-1}Y^{-1}AY^{-T}.$$

This is a simpler case than the previous one, because the last row and the last column, while not a transpose of each other, are symmetric up to a scaling factor (and up to the first element, which is zero in the row but not zero in the column). In either case, we now eliminate the offdiagonals in the first row, which have just filled. We use an ordinary sequence of column operations (a Gauss transform applied from the right),

$$\begin{bmatrix} \times & \otimes & \otimes & \otimes & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \oplus & \oplus & \oplus & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & & & & \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \oplus & \oplus & \oplus & \times \end{bmatrix} = G^{-1}Y^{-1}AY^{-T}U^{-1}$$

(the offdiagonals in the last row might be zero). If $\left[Y^{-1}AY^{-T}\right]_{11} \neq 0$, then the last row and column are symmetric up to a scaling, so we scale the reduced matrix back to symmetry,

$$\begin{bmatrix} \times & & & & \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \oplus & \oplus & \oplus & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & & & & \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \end{bmatrix} = S^{-1}G^{-1}Y^{-1}AY^{-T}U^{-1}.$$

If $\left[Y^{-1}AY^{-T}\right]_{11}$ was zero, then scaling does not symmetrize the matrix, so instead of scaling we switch to an elimination step of the third kind. To achieve numerical stability we also switch to an elimination of the third kind if $\left[Y^{-1}AY^{-T}\right]_{11}$ was not zero, but it was so small that after the multiplication by G^{-1}, the (n, n) element was larger than all the other elements in the last row.

Elimination steps of the third kind. When we switch from an elimination of the second kind to an elimination of the third kind, the reduced matrix has the form

$$\begin{bmatrix} \times & & & & \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \oplus & \oplus & \oplus & \times \end{bmatrix} = G^{-1}Y^{-1}AY^{-T}U^{-1},$$

where the offdiagonals in the last row might be either all zeros (this is the only case in exact arithmetic), or else they are identical up to a scaling factor to the last column. We treat both cases, although some of the transformations in the identically-zero case can be skipped. We begin by permuting the last row and column to be the second; this is a bandwidth issue, and if the matrix is not banded, we could continue with the last row and column in place. We now have

$$\begin{bmatrix} \times & & & & \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \times & \times & \times & \otimes \\ & \oplus & \oplus & \oplus & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & & & & \\ & \times & \oplus & \oplus & \oplus \\ & \otimes & \times & \times & \times \\ & \otimes & \times & \times & \times \\ & \otimes & \times & \times & \times \end{bmatrix} = P^TG^{-1}Y^{-1}AY^{-T}U^{-1}P.$$

We now use a sequence of symmetric Gauss or Givens transforms to eliminate all but the last element in the second row and column. Because the second row and column are scaled copies of each other, the same transformations applied to both the rows and the columns eliminate the nonzeros in both,

$$\begin{bmatrix} \times & & & & \\ & \times & \oplus & \oplus & \oplus \\ & \otimes & \times & \times & \times \\ & \otimes & \times & \times & \times \\ & \otimes & \times & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & & & & \\ & \times & & & \oplus \\ & & \times & \times & \times \\ & & \times & \times & \times \\ & \otimes & \times & \times & \times \end{bmatrix}$$
$$= X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}.$$

The last step is to eliminate the offdiagonal nonzero in the second row and column. We eliminate them using two elementary unsymmetric Gauss transforms. We show later that this is always possible and stable,

$$\begin{bmatrix} \times & & & & \\ & \times & & & \oplus \\ & & \times & \times & \times \\ & & \times & \times & \times \\ & \otimes & \times & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & & & & \\ & \times & & & \\ & & \times & \times & \times \\ & & \times & \times & \times \\ & & \times & \times & \times \end{bmatrix}$$
$$= K^{-1}X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}\hat{K}^{-T}.$$

This concludes the overview of our new elimination method.

3.2. Specification of the transformations. We now specify exactly each one of the transformations that are involved in the new elimination method. More specifically, we show how to construct the matrices L, Y, G, U, S, P, X, K, and $\hat{K}$. Some of them can be constructed in many different ways; we present the different options, but focus on the ones that guarantee stability and maintain the bandwidth of the matrix.

The matrix *L*. When a_{11} is large in absolute value relative to the rest of the first column, the column can be eliminated using a conventional Gauss transform

$$
\begin{aligned}
A &= \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \\
&= \begin{bmatrix} 1 & & & \\ l_{2,1} & 1 & & \\ \vdots & & \ddots & \\ l_{n,1} & & & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a^{(1)}_{2,2} & \cdots & a^{(1)}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a^{(1)}_{n,2} & \cdots & a^{(1)}_{n,n} \end{bmatrix} \begin{bmatrix} 1 & l_{2,1} & \cdots & l_{n,1} \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix},
\end{aligned}
$$

where $l_{i,1} = a_{i,1}/a_{1,1}$. If all the elements in the first row and column are zeros, then we simply set L to be the identity matrix.

The matrix *Y*. The task of Y is to zero rows $2, \ldots, n-1$ in the first column of A, even when a_{11} is zero or small. There are many choices for Y. For example, we could define Y to be a product of a permutation matrix that exchanges row n with the row whose first element is largest in absolute value, and a Gauss transform that uses the last row to zero the first element in the other rows. For example, if the largest element in the first column is in row 2,

$$
Y^{-1} = \begin{bmatrix} 1 & & & & & 0 \\ & 1 & & & & y_{n,2} \\ & & 1 & & & y_{n,3} \\ & & & 1 & & \vdots \\ & & & & 1 & y_{n,n-1} \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & & \\ & 0 & & & & 1 \\ & & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & \\ & 1 & & & & 0 \end{bmatrix}.
$$

However, this would ruin the banded structure, so we resort to a slightly more complex transformation. Our strategy is to annihilate the nonzeros in the first column sequentially from top to bottom, and, in particular, to annihilate nonzero in position $(i, 1)$ using a row operation involving only rows i and $i+1$. That row operation can be either a Givens rotation or a Gauss transform possibly preceded by a row exchange, to ensure that the first element in row $i+1$ is larger or equal in absolute value to the first element in row i. Our implementation uses a Givens rotation. The matrix Y has the following structure:

$$
Y^{-1} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & y^{(n-1)}_{n-1,n-1} & y^{(n-1)}_{n-1,n} \\ & & & & y^{(n-1)}_{n,n-1} & y^{(n-1)}_{n,n} \end{bmatrix} \cdots \begin{bmatrix} 1 & & & & & \\ & y^{(2)}_{2,2} & y^{(2)}_{2,3} & & & \\ & y^{(2)}_{3,2} & y^{(2)}_{3,3} & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix}.
$$

Each 2-by-2 block is either a Givens rotation, or (if we use Gauss transforms) a product of a row exchange, perhaps identity, and a row operation on one row. After the application of Y^{-1} and Y^{-T} to A, the $(n, 1)$ element of the reduced matrix cannot be zero: if it is, then all the subdiagonal elements of A in the first column were zeros, and we would have eliminated the first column using an elimination step of the first kind.

The matrix G. The role of G is to annihilate the $(n, 1)$ element in the reduced matrix. If we are using G, then that element is larger than the $(1, 1)$ element, otherwise we would have used an elimination step of the first kind. Therefore, there are essentially two options for G^{-1}: an exchange of rows 1 and n, followed by a row operation that reduces row n using row 1, or a Givens transform on rows 1 and n. We always use a Givens transform, but the other choice is valid as well.

The matrix U. After the application of G^{-1}, the $(1, 1)$ element of the reduced matrix is always nonzero, although it is not necessarily large relative to the rest of row 1. We use a matrix U^{-1}, an elementary Gauss transform applied from the right to annihilate elements $(1, 2), (1, 3), \ldots, (1, n)$ using column operations with the first column. Because only the first element in the first column is nonzero, the trailing submatrix is not modified. This will prove to be important when we analyze the numerical stability of the algorithm, since U is potentially ill-conditioned. Formally,

$$U^{-1} = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}.$$

The matrix S. The job of S is simply to scale the last row so that it becomes identical to the last column, so it is a diagonal matrix with 1s on the diagonal, except for the last element, which is nonzero but different from 1.

The matrix P. If an appropriate S does not exist or if it would be too ill-conditioned, the algorithm switches to an elimination step of the third kind. The matrix P now moves the last row and column to the second position, so that they can be eliminated. If there are no band issues, we could simply use a single row and column exchange. But to maintain the band, we do not exchange rows 2 and n. Instead, we use a cyclic permutation that puts row n in row 2 and shifts rows $2, \ldots, n-1$ one row down each,

$$P^{-1} = \begin{bmatrix} 1 & & & & \\ & 0 & & & 1 \\ & 1 & 0 & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{bmatrix}.$$

The matrix X. The matrix X has the same structure as Y, except that it operates on the second column, not the first. Although row 2 and column 2 in the reduced matrix are not transposes of each other, they are transposes up to a scaling except for the diagonal element. Since X and X^T do not use the diagonal element, they reduce the matrix to the desired form when applied symmetrically.

The matrices K and $\hat{K}$. These two matrices use the $(2, 2)$ element in the reduced matrix to annihilate the $(n, 2)$ and $(2, n)$ elements in the reduced matrix. These elements are different, so we use two different Gauss transforms from the left and the right.

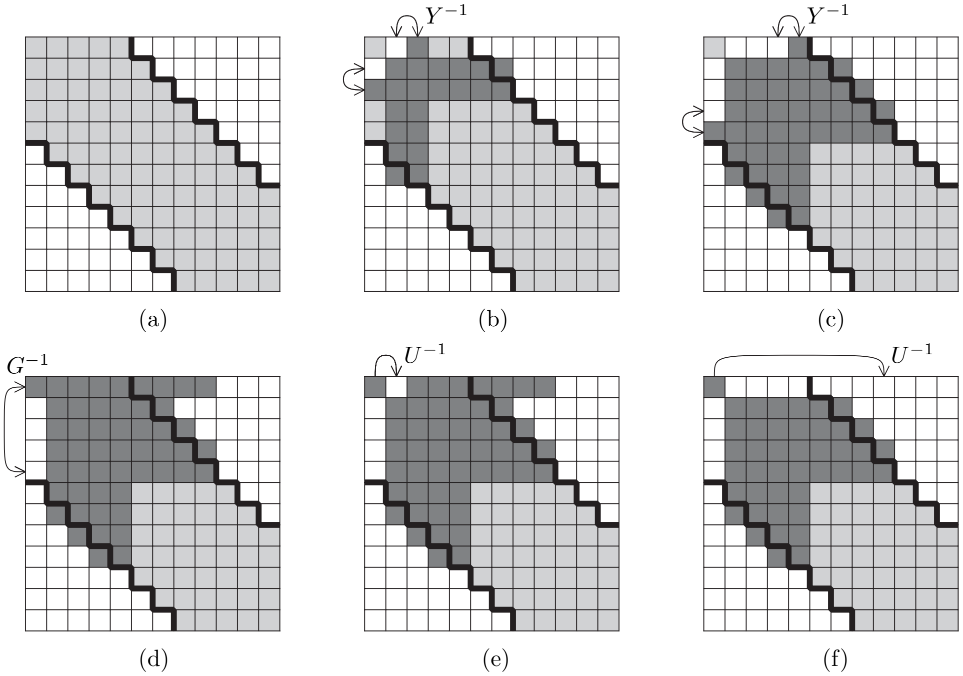
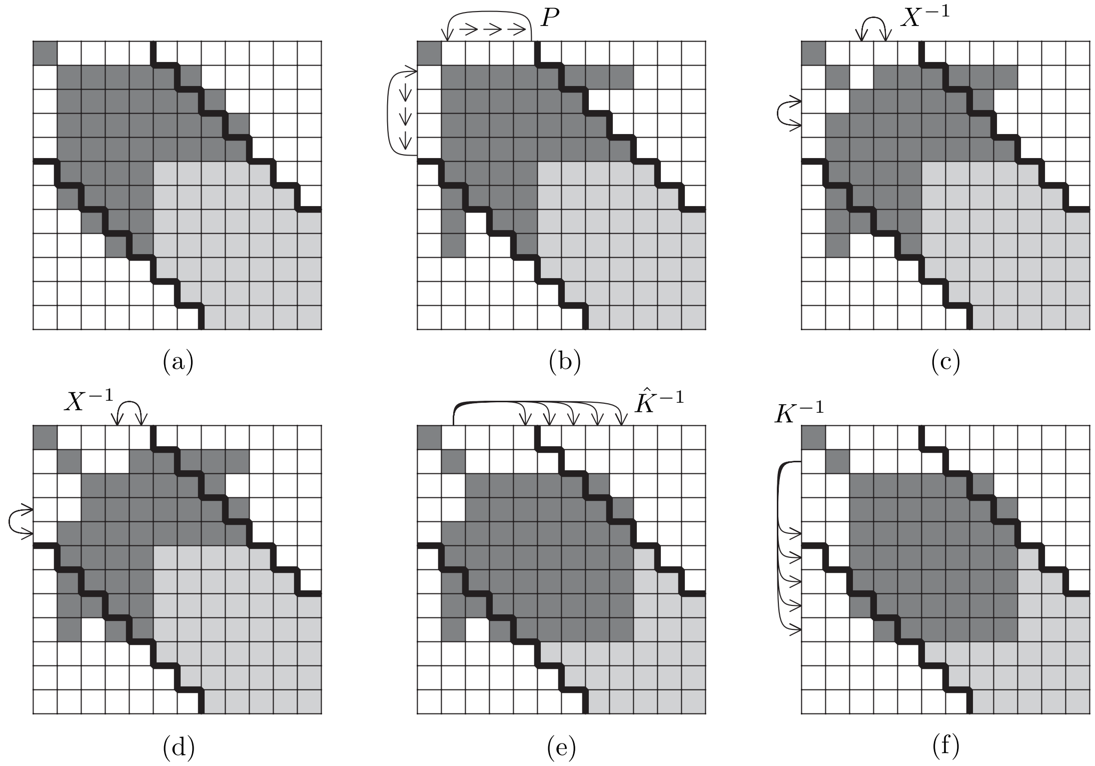


FIG. 1. *An elimination step of the second kind.*

4. Applying the new method to banded matrices. When A is banded, we need to modify the elimination algorithm to avoid increasing the band too much. To simplify the description of the algorithm, we assume that no exact cancellation occurs; but the algorithm itself copes easily with exact cancellations. It turns out that with a careful selection of transformations, the bandwidth of the reduced matrix grows, but not by much. We denote the half-bandwidth of A by m, so all but $2m+1$ of A's diagonals are zero. An elimination step of the first kind does not fill the matrix at all, so it does not require any special attention, except to ensure that no computation on zeros occurs. Elimination steps of the second and third kinds require more care.

Figures 1 and 2 illustrate elimination steps of the second and third kinds. They depict nonzero elements in gray; light gray signifies that an element has not been modified in this elimination step, and dark gray signifies that an element has been modified. The original profile of the matrix is indicated by a heavy black border. The arrows show which column/row are scaled and subtracted from which other column/row to annihilate a nonzero. The letters indicate the transformation matrix that performs the elimination.

When an elimination step of the second kind is applied to a banded matrix, we must construct transformation Y in a way that A does not fill too much. Specifically, we eliminate element i in the first row and column using columns/rows i and $i+1$, but we stop if elements $i+2,\ldots,n$ are all zeros. We do not “roll” the last nonzero in the row/column down to the end of the row/column. This is illustrated in Figure 1 (top row). The rest of the step proceeds without modification. As shown in the figure, an elimination step of the second kind can increase the half-bandwidth of the matrix, but only by one, and only in rows and columns $2,\ldots,m'$, where $m'+1$ is the number of nonzeros in row/column 1 (m' might be larger than m if the bandwidth increased



(a) (b) (c)

(d) (e) (f)

FIG. 2. *An elimination step of the third kind. The top left illustration describes the matrix after all the operations of a second-kind elimination step have been applied.*

in previous elimination steps).

Applying an elimination step of the third kind to a banded matrix is more involved. After the permutation P has been applied, the second row and column are eliminated. The elimination of these nonzeros can destroy the band structure if done carelessly. We can eliminate all but one of the nonzeros using a series X of Givens transforms, as we have done (using Y) in the beginning of the second-kind elimination. We can also use column operations to annihilate the second row using its diagonal element as a pivot, and then row operations to annihilate the second column; these are the transformations K and $\hat{K}$ that we described above. In the dense case, we use K and $\hat{K}$ only to annihilate a single nonzero in the row and a single nonzero in the column. But in the banded case, we need to restrict X to the annihilation of just $m' - 2$ nonzeros near the diagonal, and use K and $\hat{K}$ to annihilate the rest. As we shall see later, this does not introduce growth in the reduced matrix, since the diagonal element in row 2 is larger than the offdiagonal nonzeros.

The strategy that we use is shown in Figure 2. It is easy to see that if we use Givens transforms to annihilate all but the last element in the second row and column, the band will grow by one in rows/columns higher than m', which we try to avoid. On the other hand, if we use only diagonal Gauss transforms, an entire bulge outside the band would fill, which we also seek to avoid. Therefore, we use Givens transforms, which only increase the band locally by one, to annihilate $m' - 2$ nonzeros in the first row and column, and Gauss transforms, which introduce no fill at all, to annihilate the rest.

4.1. Bound on half-bandwidth. We now prove that this strategy ensures that the half-bandwidth of the reduced matrix is bounded by $2m-1$. We do this in two steps. We first define a variant of our algorithm, which we call ALG2, and show that the fill in our algorithm is a subset of the fill created by ALG2. This variant always uses reductions of the second type; it may be unstable, so it is not useful in practice. We only use it to bound the bandwidth of the reduced matrix.

In the context of this section we ignore any numerical considerations. Our analysis deals only with the structural aspects of our algorithm (and of the variant ALG2). In particular, we assume that A has no zeros inside its band and we ignore zeros that may appear during the reduction process.

We start with a few definitions. A reduction step in our algorithm starts with a symmetric banded matrix $A^{(k)}$, where k denotes the size of the matrix (in the first step, $A^{(n)} = A$), and outputs a reduced symmetric matrix $A^{(k-s)}$ of size $(k-s)$-by-$(k-s)$. The number s of eliminated rows and columns may be 1 or 2. This notation is borrowed from [4]. We denote by $a_{ij}^{(k)}$ both the ij element of $A^{(k)}$ at the start and during the reduction step. The *local half-bandwidth* of a banded matrix B in column (row) j, defined as the minimum $r' \geq 0$ for which $b_{ij} = 0$ for all $i > j + r'$, is denoted by r_B^j . In the case of the input matrix A, all the columns (rows) have the same local half bandwidth m.

We now define the variant algorithm, ALG2. It is similar to our actual algorithm, which we denote by ALG1, except that it only uses reduction steps of the second type, and that it eliminates the offdiagonal nonzeros in the first column only using Givens rotations. (ALG1 uses Gauss transforms, perhaps with row exchanges.) ALG2 eliminates one row and column in each reduction step, so it uses exactly $n-1$ reduction steps. We denote by $\widetilde{A}^{(k)}$ the reduced matrix we get after applying the first $n-k$ reduction steps of ALG2 on A, where $0 < k \leq n$. Note that $\widetilde{A}^{(k)}$ is k-by-k.

The following theorem states the main result of this section.

THEOREM 4.1. *Let m be the bandwidth of A. Then the local half-bandwidth of the reduced matrices in our algorithm* (ALG1) *satisfies $r^i_{A^{(k)}} < 2m$ for $0 < k \leq n$, $1 \leq i \leq k$.*

This theorem is an immediate corollary of Lemmas 4.3 and 4.4, which we state and prove below. But we first state and prove a technical lemma.

LEMMA 4.2. *In* ALG2, $r^{i+1}_{\widetilde{A}^{(k)}} \leq r^{i}_{\widetilde{A}^{(k)}} \leq r^{i+1}_{\widetilde{A}^{(k)}} + 1$*, where* $1 < k \leq n$ *and* $1 \leq i < k$.

Proof. We prove the lemma by induction on k. The lemma holds for $A = \widetilde{A}^{(n)}$. We assume that the lemma holds for $\widetilde{A}^{(k)}$. During the $n-k+1$th reduction step, each of the columns $2, \ldots, r^1_{\widetilde{A}^{(k)}}$ gets the structure of the column to its right. This is because

$$r^{i}_{\widetilde{A}^{(k)}} \leq r^{i+1}_{\widetilde{A}^{(k)}} + 1$$

by the induction assumption. Each of the rows $2, \ldots, r^1_{\widetilde{A}^{(k)}}$ behaves in a symmetric way. For j such that $1 \leq j < r^1_{\widetilde{A}^{(k)}} - 1$, we have (see Figure 3)

$$r^{j}_{\widetilde{A}^{(k-1)}} = r^{j+2}_{\widetilde{A}^{(k)}} + 1$$

and

$$r^{j+1}_{\widetilde{A}^{(k-1)}} = r^{j+3}_{\widetilde{A}^{(k)}} + 1.$$

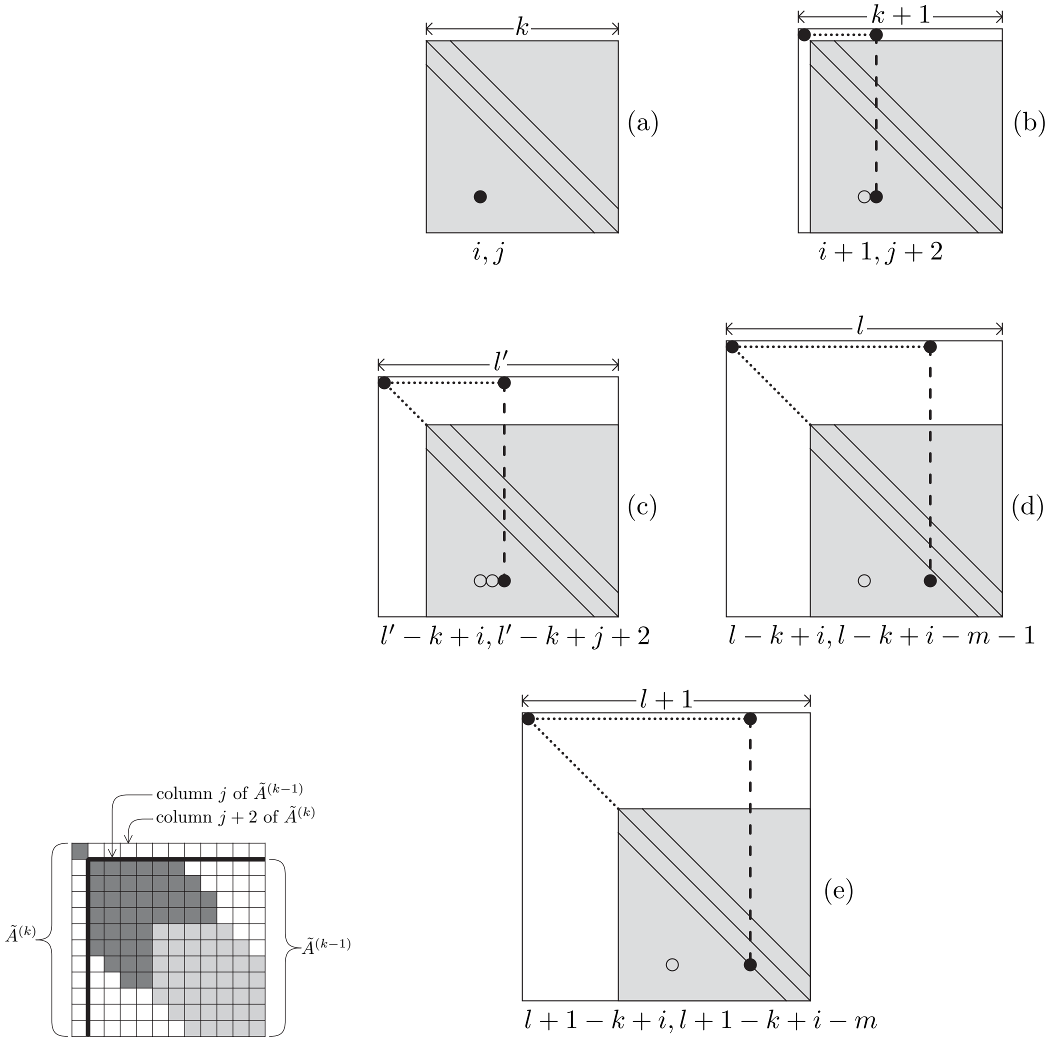


FIG. 3. *Left: An illustration of the proof of Lemma* 4.2. *Right: An illustration of the main steps in the proof of Lemma* 4.4. *The submatrix depicted in gray is always k-by-k. The row and column indices of the black element are shown below each matrix (the black element within the gray submatrix).*

From this and the induction assumption,

$$r^{j+1}_{\widetilde{A}^{(k-1)}} \leq r^{j}_{\widetilde{A}^{(k-1)}} \leq r^{j+1}_{\widetilde{A}^{(k-1)}} + 1,$$

where $1 \leq j < r^{1}_{\widetilde{A}^{(k)}} - 1$. We also have

$$r^{j}_{\widetilde{A}^{(k-1)}} = r^{j+2}_{\widetilde{A}^{(k)}} + 1 = r^{j+1}_{\widetilde{A}^{(k-1)}} + 1$$

for $j = r^{1}_{\widetilde{A}^{(k)}} - 1$, which means that

$$r^{j+1}_{\widetilde{A}^{(k-1)}} \leq r^{j}_{\widetilde{A}^{(k-1)}} \leq r^{j+1}_{\widetilde{A}^{(k-1)}} + 1$$

is true also for such j. For $r^1_{\widetilde{A}^{(k)}} - 1 < j < k - 1$ the lemma is immediate from the induction assumption. □

Lemma 4.3. *Denote by η_M the set of nonzero entries in a given matrix M. If $A^{(k)}$ is a reduced matrix that ALG1 generates when it is applied to A, and if $\widetilde{A}^{(k)}$ is a reduced matrix that ALG2 generates when it is applied to A, then*

$$\eta_{A^{(k)}} \subseteq \eta_{\widetilde{A}^{(k)}}.$$

Proof. Let $A^{(k)}$ be the reduced matrix we get from the lth reduction step of ALG1, when applied to A.

The proof is immediate for $l = 0$ (and so $k = n$) when no reduction step has been applied to A yet.

Otherwise, the output of the previous reduction step is either $A^{(k+1)}$ or $A^{(k+2)}$. If the output of reduction step $l - 1$ is $A^{(k+1)}$, then by the induction assumption we have

$$\eta_{A^{(k+1)}} \subseteq \eta_{\widetilde{A}^{(k+1)}}.$$

Moreover, it is guaranteed that either Gaussian elimination or a reduction of the second kind is applied to $A^{(k+1)}$ in order to get $A^{(k)}$. Due to the facts that the reduction applied on $\widetilde{A}^{(k+1)}$ by ALG2 is of the second kind and that the Gaussian elimination, if applied by ALG1 to $A^{(k+1)}$, does not increase the band, we have

$$\eta_{A^{(k)}} \subseteq \eta_{\widetilde{A}^{(k)}}.$$

If the output of the $l - 1$th reduction step is $\mathrm{A}^{(k+2)}$, then a reduction of the third kind is applied to $A^{(k+2)}$ in order to get $A^{(k)}$. The operations whose effect on the structure of $A^{(k)}$ we need to consider are applying Y^{-1}, bringing column and row $r^{(1)}_{A^{k+2}} + 1$ to the second position, applying X^{-1}, and finally applying a nonsymmetric Gauss reduction. Let B be the $k + 1 \times k + 1$ right bottom block of the matrix we get after applying Y^{-1} to $A^{(k+2)}$. We have

$$\eta_B \subseteq \eta_{\widetilde{A}^{(k+1)}} \tag{4.1}$$

due to similar arguments to those by which we have shown that $\eta_{A^{(k)}} \subseteq \eta_{\widetilde{A}^{(k)}}$, in case the output of the $l - 1$th reduction step is $\mathrm{A}^{(k+1)}$. Let C be the matrix we get after bringing the $r^{(1)}_{A^{(k+2)}}$ column and row of B to the first column and row, respectively. During this operation, each of the columns (rows) $1, \ldots, r^{(1)}_{A^{(k+2)}} - 1$ of B is moved one location to the right (down). From the combination of (4.1) with Lemma 4.2 we have that the structures of columns (rows) $2, ldots, r^{(1)}_{A^{(k+2)}}$ in C are contained in the structures of these columns (rows) in $\widetilde{A}^{(k+1)}$, respectively. We get

$$\eta_{C_{[k]}} \subseteq \eta_{\widetilde{A}^{(k+1)}_{[k]}}, \tag{4.2}$$

where $M_{[k]}$ denotes the k-by-k right bottom block of the matrix M. The next structure-relevant part of a third-kind reduction is X^{-1}. The last element elimination performed during applying X^{-1} on C in the first column is done by the $r^1_{A^{(k+2)}}$th row. The last element elimination performed during the Y^{-1} part in the ALG2 reduction step of $\widetilde{A}^{(k+1)}$ in the first column is done by the $r^1_{\widetilde{A}^{(k+1)}} + 1$th row. For these two indices, we have

$$r^1_{A^{(k+2)}} \leq r^1_{\widetilde{A}^{(k+2)}} \leq r^2_{\widetilde{A}^{(k+2)}} + 1 \leq r^1_{\widetilde{A}^{(k+1)}} + 1 \tag{4.3}$$

by the induction assumption, Lemma 4.2, and the trivial fact $r^2_{\widetilde{A}^{(k+2)}} \le r^1_{\widetilde{A}^{(k+1)}}$. Denote the matrix we get by applying X^{-1} on C by D. By combining (4.3) with (4.2) we get that the structure of $D_{[k]}$ is contained in the structure of $\widetilde{A}^{(k)}$. The last operation performed during a reduction of the third kind is a nonsymmetric Gauss reduction. The effect on the fill that this operation may have on D is limited to elements d_{ij}, where

$$r^1_{A^{(k+2)}} \le i, j \le r^1_C + 1.$$

The fact that $\widetilde{A}^{(k+1)}$ has a nonzero element in entry $(r^1_C + 1, r^1_{A^{(k+2)}})$ (because B has a nonzero there, and due to (4.1)) and Lemma 4.2 bring us to the conclusion that the elements of $\widetilde{A}^{(k)}$ in entries i, j, where $r^1_{A^{(k+2)}} - 1 \le i, j \le r^1_C$ are all nonzeros. We got

$$\eta_{A^{(k)}} \subseteq \eta_{\widetilde{A}^{(k)}}. \qquad \square$$

We now prove the following lemma on ALG2.

LEMMA 4.4. *Let m be the half-bandwidth of A. Then $r^i_{\widetilde{A}^{(k)}} < 2m$ for $0 < k \le n$, $1 \le i \le k$.*

Proof. Let us focus in a nonzero element located at i, j such that $i - j$ is maximal over all elements that appear during the factorization. More formally, i and j are such that $i - j = max_{i',j',t}\{i' - j' : \widetilde{a}^{(t)}_{i'j'} \neq 0\}$, $i > j$. Assume, without loss of generality, that this nonzero is the first to fill the criterion and that the reduced matrix $F = \widetilde{A}^{(k)}$ is the first in which the nonzero appears (see Figure 3:A). The appearance of the nonzero in the jth column of the ith row of $\widetilde{A}^{(k)}$ implies that the $i+1, j+2$ element in $\widetilde{A}^{(k+1)}$ is nonzero (Figure 3:B). This, in turn, implies that there is some $l' > k + 1$ s.t. $\widetilde{a}^{(l')}_{l'-k+i,l'-k+j+2} \neq 0$ (Figure 3:C). If we continue similarly, we get to the observation that there is some $l > k+i-j-m-2$ s.t. $\widetilde{a}^{(l)}_{l-k+i,l-k+i-m-1} \neq 0$ (Figure 3:D). Assume, without loss of generality, that l is maximal, which means that $\widetilde{a}^{(l+1)}_{l+1-k+i,l+1-k+i-m-1} = 0$ (Figure 3:E). This implies that $r^1_{\widetilde{A}^{(l+1)}} \ge 2i - j - 2m - 1$. On the other hand, $r^1_{\widetilde{A}^{(l+1)}} < i - j$. We get $2i - j - 2m - 1 < i - j$, i.e., $i < 2m + 1$ and therefore $i - j < 2m$. This proves the lemma. $\square$

4.2. Storing the factors. We store the input matrix and the factors in a format that is quite similar to LAPACK's banded storage format. The code expects to receive only the upper triangle of A. The representation of the matrix A will be stored in a two-dimensional array, with n columns and with a number of rows related to the half-bandwidth m. On input, each column of A will be stored in the corresponding column of the representation. Each row of the representation will correspond to a diagonal of A, with the main diagonal given as the first row of the representation, the first superdiagonal of A as the second row of the representation, and so forth. Rows of A will correspond to diagonals of the representation, filling the first m rows of the representation. As the algorithm runs, rows of A are overwritten with elements of the factors.

The factors of A require more storage than A itself. We accomodate the factors by requiring that the row dimension of the representation array be larger than the half-bandwidth m. This storage format is similar to the input/output format of LAPACK's DGBTRF routine, which implements Gaussian elimination with partial pivoting, where reduced rows also expand.

Strictly speaking, the leading dimension of the array should be at least eight times the half-bandwidth, to allow for storage of the output. Since most matrices do not require that much storage to factor, the code can be called with a smaller leading dimension. In particular, we have never encountered a matrix that requires more than six times the half-bandwidth, so all the runs reported later in this paper use a leading dimension that is six times the half-bandwidth.

The details of how we pack the representation of the factors into the array are not important and are omitted. Only three aspects of the data structure are important. First, the fact that rows of A are not packed, but stored with spaces in between them, allows the algorithm to reduce the remaining rows in each elimination step. That is, the algorithm is essentially a right-looking algorithm. A fully packed storage would not have allowed a right-looking algorithm. Second, the coefficients of the transformations that eliminate a row are stored in the memory that was used to store the row of A and additional memory locations that are statically preallocated for that row in the input/output array. Third, the representation of each Givens rotation is stored in one word using the scheme of [20].

We note that it is possible to reduce the storage requirements of the algorithm using dynamic storage allocation for the factors, rather than using static preallocated areas. The static allocation leaves some unused storage.

5. Numerical stability: Bounding the factors. In this section we show that the growth during the algorithm is bounded from above by 4^{n-1}, and that the factors are bounded. The growth factor ρ_A is the ratio of the largest element in absolute value in the reduced matrices to the largest element in A,

$$\rho_A = \frac{\max_{i,j,k} \left| a_{i,j}^{(k)} \right|}{\max_{i,j} |a_{i,j}|}.$$

A bound of this form, even an exponential bound like the one that we prove here, is often associated with backward stable elimination algorithms. In particular, the growth in GEPP is bounded by 2^{n-1}, the growth in Bunch and Kaufman's algorithm is bounded by 2.57^{n-1}, and the growth in Aasen's algorithm is bounded by 4^{n-2}. In practice, such growth factors are rarely encountered; the growth in practice is usually small. Researchers have shown that when the growth is small, these algorithms are backward stable [11, 12, 21, 22]. Since large growth factors are rare, these algorithms are stable in practice [9, 13]. Although we do not show that a similar implication holds for our algorithm, we believe that one does.

Moreover, the fact that a bound on the growth exists at all, even if the bound is exponential, is usually a reflection of a sound numerical design of the pivoting strategy. This observation does hold for our algorithm. The careful design of the snap-back pivoting strategy ensures that the elements of the reduced matrix can grow by at most a factor of 4 in every step.

Growth is measured on the reduced matrices, the intermediate matrices after some elimination steps have been carried out. In some algorithms, such as GEPP and Aasen, the entries in the factors are also bounded. In GEPP, for example, the magnitude of the entries of the lower triangular factor are bounded by 1 and the magnitudes in the upper triangular factor are bounded by the magnitude of elements in reduced matrices. Bounded factors are a stronger property than bounded growth. In particular, in some cases, bounded factors imply backward stability [2, Appendix B]. We show in this section that the factors in our algorithm are bounded, although this does not formally imply backward stability. We note that in the algorithms of Bunch and Kaufman, the factors are not bounded: they can have large entries even when the reduced matrices remain small (see, for example, [13, p. 219]).

Before we analyze growth, we should define formally the conditions by which we choose the next type of reduction step to perform. If possible, Gaussian elimination is applied. The condition for applying this reduction is the same as the condition that appears in the Bunch–Kaufman algorithm. This condition involves a constant α with a value between 0 and 1 and the values $\gamma_1^{(k)} = \max_{1<l\leq k} \mid a_{l,1}^{(k)} \mid$ and $\gamma_t^{(k)} = \max_{1<l\leq k} \mid a_{l,t}^{(k)} \mid$, where t is the index of the row of the entry with the maximum magnitude in the first column. Using this notation, we use a reduction of the first kind if one of the following conditions is satisfied:

1. $\mid a_{1,1}^{(k)} \mid > \alpha \cdot \gamma_1^{(k)}$, or
2. $\mid a_{1,1}^{(k)} \mid \cdot \gamma_t^{(k)} > \alpha (\gamma_1^{(k)})^2$ (optional; see below).

Otherwise, a reduction of one of the other two types is applied. Condition 2 is optional in the sense that if we use it, growth in the reduced matrices is still bounded, but the factors may grow; if we only use a reduction of the first type when condition 1 is satisfied, then both the reduced matrices and the factors are bounded.

The two other types of reductions start with the same sequences of elementary transformations. The decision about the chosen type is taken only after this shared sequence of transformations terminates. If the element $a_{r^1_{A^{(k)}}+1, r^1_{A^{(k)}}+1}$ is less than or equal (a threshold may be used) to all the other elements in its row, then the second type of reduction is chosen and S^{-1} is applied. Otherwise, the reduction proceeds as a third type reduction.

We now analyze the growth.

Consider first symmetric Gauss transformations. The analysis of this case is the same as in [4, 2]. For an element $a_{ij}^{(k)}$ in the input matrix $A^{(k)}$ and an element $a_{ij}^{(k-1)}$ in the reduced matrix $A^{(k-1)}$ we have

$$a_{ij}^{(k-1)} = a_{ij}^{(k)} - \frac{a_{i1}^{(k)} a_{1j}^{(k)}}{a_{11}^{(k)}}, \quad i > 1, \quad j > 1. \tag{5.1}$$

Let μ be the magnitude of the largest entry in $A^{(k)}$ and μ' the maximum magnitude of any entry in the reduced matrix $A^{(k-1)}$. If $a_{11}^{(k)} > \alpha \cdot \max_{1<l\leq k}(a_{l,1}^{(k)})$, then from (5.1) we get

$$\mu' \leq \mu + \frac{\mid a_{i1}^{(k)} a_{1j}^{(k)} \mid}{\mid a_{11}^{(k)} \mid} \leq \mu \left(1 + \frac{1}{\alpha}\right). \tag{5.2}$$

If $\mid a_{11}^{(k)} \mid \cdot \gamma_t^{(k)} \geq \alpha \cdot (\gamma_1^{(k)})^2$, then using (5.1)

$$\mu' \leq \mu + \frac{(\gamma_1^{(k)})^2}{\mid a_{11}^{(k)} \mid} \leq \mu + \frac{\gamma_t^{(k)}}{\alpha} \leq \mu \left(1 + \frac{1}{\alpha}\right). \tag{5.3}$$

Now, consider the second and third kinds of transformations. In order to analyze the growth for these kinds of transformations, we look at the elementary transformations from which these two complex kinds are made up. We may divide the elementary transformations into four categories. The first category includes the transformation that annihilates an element in the first or second column (row) by its nearby element in the same column (row). In our algorithm, this transformation is applied as a part of an elimination of a full column. During such an elimination, the first annihilated element is the one directly under the diagonal. The next elementary transformation annihilates the element located two entries under the diagonal, and so on. If an element to be annihilated is greater than the element under it, the rows of these two elements and the symmetric columns are swapped first. The second category of elementary transformations is the Gauss transformations, and the third includes the Givens transformation that annihilates one nondiagonal element by a diagonal element, both in the first column. The fourth category includes the one-row-scaling diagonal transformation S^{-1}, in which all of the diagonal is 1's, except, maybe, for one diagonal element, whose magnitude is smaller than 1.

The last three categories of transformations with the conditions for applying them trivially imply an upper bound of 2 on the growth the transformations induce on the matrix they are applied on. We now focus on the first category. We show it has an upper bound of 4 on the growth it induces. Assume that a sequence of Q elementary transformations of the first category, as described above, is applied on a matrix $B^{(0)}$, with a maximum magnitude element ν, in order to eliminate its first column. Denote the matrix we get after applying $q \le Q$ such elementary transformations by $B^{(q)}$, and its i, j element by $b_{ij}^{(q)}$.

LEMMA 5.1. A. *If* $i \ge q+2$ *and* $j \ge q+2$*, then* $\mid b_{ij}^{(q)} \mid \le \nu$.

B. *If* $1 \le i < q+2$ *and* $j \ge q+2$ *(or* $i \ge q+2$ *and* $1 \le j < q+2$*), then* $\mid b_{ij}^{(q)} \mid \le 2 \cdot \nu$.

C. *If* $1 \le i < q+2$ *and* $1 \le j < q+2$*, then* $\mid b_{ij}^{(q)} \mid \le 4 \cdot \nu$.

Proof. The lemma is trivially correct for $q = 0$.

Assume the lemma is correct for some $q \le Q-1$.

The only rows that may change due to the $q+1$th left elementary transformation are the $q+2$ and $q+3$ rows, and similarly, the $q+2$ and $q+3$ columns are the only columns that may change due to the $q+1$th right elementary transformation.

Elements $b_{i,q+3}^{(q+1)}$ and $b_{q+3,i}^{(q+1)}$, where $i > q+3$ are equal either to elements $b_{i,q+3}^{(q)}$ and $b_{q+3,i}^{(q)}$, where $i > q+3$ or to elements $b_{i,q+2}^{(q)}$ and $b_{q+2,i}^{(q)}$, where $i > q+3$, respectively, depends on whether or not a swap is part of the $q+1$th elementary transformation. Similarly, element $b_{q+3,q+3}^{(q+1)}$ equals either to $b_{q+3,q+3}^{(q)}$ or to $b_{q+2,q+2}^{(q)}$. By the induction assumption we have $\mid b_{ij}^{(q+1)} \mid \le \nu$ for $i \ge q+3$ and $j \ge q+3$. By that we proved A.

We also have $\mid b_{ij}^{(q+1)} \mid \le 2\nu$ for $i \ge q+3$ and $j = q+2$ and for $i = q+2$ and $j \ge q+3$, because each of these elements is computed as a sum of two elements in $B^{(q)}$, which are at most ν by the induction assumption. In addition, from the induction assumption, $\mid b_{ij}^{(q+1)} \mid \le 2\nu$ for $i \ge q+3$ and $1 \le j < q+2$ and for $1 \le i < q+2$ and $j \ge q+3$. This completes the proof of B.

In order to prove C it is enough to show that $\mid b_{ij}^{(q+1)} \mid \le 4 \cdot \nu$ for $i = q+2$ and $1 < j < q+3$ (and for $1 < i < q+3$ and $j = q+2$). When $i \ne j$ this bound holds because each of the elements is computed as the sum of two elements in $B^{(q)}$, which are at most 2ν by the induction assumption. If $i = j = q+2$ then $b_{i,j}^{(q+1)}$, is the sum of the four elements $b_{i,j}^{(q)}$, $b_{i+1,j}^{(q)}$, $b_{i,j+1}^{(q)}$, and $b_{i+1,j+1}^{(q)}$. By the induction assumption, each of these elements is at most ν. This completes the proof. ∎

A similar lemma holds for the case where the second column and row are eliminated using a sequence of transformations of the first category. Such a case may be interpreted simply as eliminating the first row and column of a matrix C, which is $B^{(0)}$ without its first row and column and with some permutation on its columns/rows.

This extraction of the lemma bounds the growth induced by the second sequence of transformations of the first category applied during the third type of reduction.

From the above upper bounds on growth during the first type of reduction and on growth induced by the four categories of elementary transformations, we can now conclude an upper bound on the growth during the full reduction step. In case the first type of elimination is chosen, the growth is bounded by $1 + \frac{1}{\alpha}$. In case the second or third type is applied, we have from Lemma 5.1 that the sequences of the first category of elementary transformations (Y^{-1}or X^{-1}) causes each a growth bounded by 4. The other elementary transformations may cause growth only to elements that didn't grow above the old maximum element in the matrix during the last applied Y^{-1} (or X^{-1}) transformation. It means that at the end of a second type reduction step, the growth is bounded by 4. It means also that this is the case just before applying X^{-1}. The growth caused by the X^{-1} transformation itself and the elementary transformation that follows, is again bounded by 4 .This brings us to total growth bounded by 16 for the third type of reduction.

We now can use a similar approach to the one used by the Bunch–Kaufman algorithm to determine the value α. We have growth bounded by 16 for a double column elimination done during the third type of reduction, and bounded by 4 for a single column elimination done during the second type. It means that the elimination of a column during the second or third reduction types induces a growth of 4. In order to minimize the bound that we have on a general reduction step, we need to find α such that

$$\mu \left(1 + \frac{1}{\alpha}\right) \leq 4\mu.$$

Therefore, we need to ensure that $\alpha \geq 1/3$; by setting $\alpha = 1/3$, we maximize the opportunity to use elimination steps of the first type, which is the cheapest type.

In total we have an upper bound of 4^{n-1} on the growth factor during the factorization.

We now prove a bound on the entries of the factors.

Lemma 5.2. *Suppose that we use a reduction of the first type only when* $\mid a_{1,1}^{(k)} \mid > \alpha \cdot \gamma_1^{(k)}$ *for some* $0 < \alpha \leq 1$. *Then the magnitude of elements of the factors is bounded by* α^{-1} *or by the elements of the reduced matrices. By reduced matrix we refer here to the matrix after the application of both full and partial elimination steps, not just after some rows and columns have been completely eliminated.*

Proof. We show that elements of the nine matrices L, Y, G, U, S, P, X, K, and $\hat{K}$ are all bounded. L is a Gauss transform in which the pivot is smaller than the elements in its row and column by at most a factor of α, so the elements of L are bounded by α^{-1}. K is a similar transform, except that the pivot is larger or equal to the rest of its row. Therefore, the elements of K are bounded by $1 \leq \alpha^{-1}$. G represents a single Givens rotation, so its entries are bounded by 1. Y and X represent either a sequence of Givens rotations, or a sequence of offdiagonal Gauss transforms in which the pivot is larger than the element that it annihilates. If we use Givens rotations, Y and X are orthonormal so their entries are bounded by 1. If we use Gauss transforms, Y and X are row permutations of unit lower-triangular matrices with subdiagonal elements bounded by 1. The matrix U represents a series of column operations that annihilate a row in a reduced matrix. Thus, U is an upper triangular matrix, with one row that contains a copy of the annihilated row in the reduced matrix, and with a unit diagonal elsewhere. Therefore, the elements of U are elements of a reduced matrix and 1's, so they are bounded. The matrix $\hat{K}$ is similar, except that it represents row operations. P is a permutation matrix, so its entries are bounded by 1. S is a transformation that scales a single row *down* (S^{-1} scales a row up), so it is a diagonal matrix, with diagonal entries that are 1 except for one entry, which is smaller than 1. □

The qualification that the factors are bounded with respect to partially-eliminated reduced matrices, not with respect to reduced matrices after the full elimination of some rows and columns, is not a significant one. Our previous analysis of the growth in the reduced matrices covers partially-eliminated reduced matrices as well.

6. Implementation and results. This section describes our implementation of the algorithm and presents results of experiments that investigate the behavior and performance of the algorithm.

6.1. Implementation and benchmark codes. We have implemented the algorithm in the C language[2] using LAPACK-style interfaces. The implementation includes two externally-visible routines, one to factor a symmetric banded matrix and another to solve a linear system using a previously-computed factorization.

The threshold value α that our implementation uses is $\alpha = 1/3$, the value that minimizes worst-case element growth. The implementation uses Givens rotations in the Y and X factors.

We tested this implementation against three other codes: LAPACK's banded LU with partial pivoting (DGBTRF), LAPACK's banded LU with partial pivoting but without blocking (DGBTF2; this is an internal LAPACK routine), and the symmetric band reduction, an orthogonal factorization code for banded symmetric matrices [3]. Our code is not blocked. That is, it does not partition the matrix into blocks to achieve high cache efficiency. Thus, from the algorithmic point of view, it is most appropriate to compare it to other nonblocked factorization codes, such as DGBTF2. From the practical point of view, it is also important to know how our implementation compares to the best existing factorization code, which is DGBTRF. (This comparison, however, does not reveal much about our algorithm, since it is difficult to separate the algorithmic and cache issues that influence the performance of DGBTRF.)

6.2. Test environment. We conducted the experiments on two different machines. Some of the experiments were conducted on a 3.2 GHz Pentium 4 computer running Linux with 2 GB of main memory. We compiled our code using GCC version 4.0.0. The version of LAPACK that we used was compiled using GCC 2.91 from C sources that were generated from the Fortran sources using `f2c`. We linked our code as well as LAPACK with an implementation of the BLAS by Kazushige Goto, version 0.99 for Pentium 4 (Coppermine). We measured this LAPACK/BLAS combination against Intel's Math Kernel Library (MKL) version 7.2.1 and found that the performance of the banded GEPP subroutines in the two implementations was similar. Therefore, the performance that we report does not appear to be affected by the use of the Fortran-to-C translation or of GCC. We used this LAPACK/BLAS combination because it delivered better performance for our algorithm than MKL.

Other experiments were performed on a dual 3 GHz AMD Opteron machine with 8 GB of main memory. On this machine we used GCC 3.4.2 and Goto's BLAS version 0.97 for 64-bit Opterons (we do not report detailed timing data on this machine, only accuracy data).

[2] The code is available from http://www.tau.ac.il/∼stoledo/research.html.

Table 1
Test matrices from the Gould–Scott study.

#	name	n	m
1	linverse	11999	4
2	spmsrtls	29995	4
3	dtoc	24993	5
4	dixmaanl	60000	7
5	sit100	10262	396
6	tuma2	12992	322
7	stokes64	12546	384
8	stokes64s	12546	384
9	aug2d	29008	198
10	aug2dc	30200	202
11	tuma1	22967	483
12	crystk02	13965	821
13	mario001	38434	356
14	aug3d	24300	759
15	crystk03	24696	1034
16	bratu3d	27792	945
17	cont-201	80595	403
18	ncvxqp1	12111	2692
19	aug3dcqp	35543	994
20	bcsstk37	25503	1427
21	ncvxqp9	16554	2216
22	stokes128	49666	768
23	bcsstk39	46772	817
24	dawson5	51537	867
25	bcsstk35	30237	1764
26	vibrobox	12328	4535
27	helm3d01	32226	2593
28	ncvxbqp1	50000	1682
29	k1_san	67759	1574
30	olesnik0	88263	1214
31	cont-300	180895	603
32	copter2	55476	2304
33	qa8fk	66127	2016

Table 2
Test matrices from John Betts.

#	name	n	m
1	traj02	1665	457
2	traj06a	1665	466
3	traj06b	1665	466
4	traj15a	1999	1882
5	traj15b	1999	1882
6	traj15c	1999	1882
7	traj15d	1999	1882

We used two different machines because we had the MKL performance of GEPP as a baseline only on Intel Pentium 4 machines, but 32-bit Pentium 4 machines could not factor some of the larger test matrices. The 64-bit Opteron allowed us to factor large matrices.

6.3. Reliability, stability, and accuracy. We performed several experiments in order to assess the reliability, stability, and accuracy of the algorithm. These were all conducted on the Opteron machine. We used three families of test matrices for these experiments. One family consists of the 61 matrices that Gould and Scott used to evaluate sparse symmetric indefinite direct solvers [10]. We reordered the rows and columns using reverse Cuthill–McKee ordering, to reduce their bandwidth. Of the 61 matrices, our algorithm ran out of memory on 28 on a machine with 8 GB of memory, leaving for our experiment the 33 matrices listed in Table 1. Another family consisted of 7 matrices that were collected by Roger Grimes from John Betts of Boeing, listed in Table 2. They are KKT matrices extracted out of a sparse nonlinear optimization package called SOCS. These problems were among those that led to the discovery that the Bunch–Kaufman algorithm can create large entries in the lower-triangular factor. Solving linear systems with these factors resulted in very-low-accuracy solutions, which led to convergence failures in the nonlinear optimization algorithm.

Figure 4 shows the growth and the norm of the residual on the Gould–Scott matrices. These runs were performed with $\alpha = 1/3$. The right-hand-side b was generated by multiplying A by a random solution vector x. The figure shows the growth in the reduced matrices in our algorithm and the residual in both our algorithm and LAPACK's GEPP. Our code failed on one matrix due to overflows in the solve phase (GEPP overflowed on 4), and produced an unacceptable residual on only one matrix. That matrix suffered from large growth. The data suggests that very large growth leads to backward instability in the algorithm. The data also suggests that problems in the solve phase can occur even when there is no growth. On the other

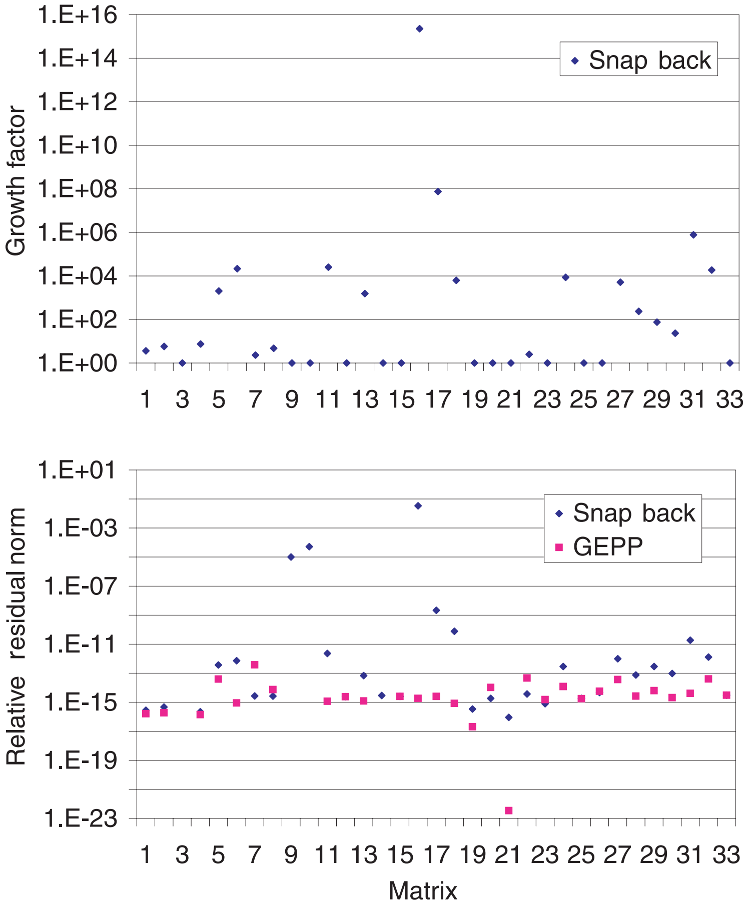


FIG. 4. *Growth in the reduced matrices in our algorithm and the residual in both our algorithm and GEPP on the* 33 *matrices from the Gould–Scott study.*

hand, our algorithm suffers from fewer problems in the solve phase than GEPP. On the 7 Betts matrices the residual was always at least a factor of 10^{12} smaller than the right-hand side; we did not detect any problems.

We also conducted experiments to assess the behavior of the algorithm under various α-thresholds. For this experiment we selected two of the Betts matrices and two particularly difficult matrices from the Gould–Scott collection, CONT-300 and BRATU3D. These two matrices caused large growth under $\alpha = 1/3$; CONT-300 caused instability, but BRATU3D did not. Figure 5 shows the norms of the forward errors, normalized with respect to $\|x\|$, as a function of α. The results show that increasing α can improve significantly the accuracy of the algorithm. This is particularly pronounced in the difficult Gould–Scott matrices. The accuracy appears to be roughly monotone with α. We note that a small α improves performance, as we shall see later, but even with a large α our theoretical results concerning growth and bandwidth hold.

6.4. Performance. Next, we describe several experiments that evaluate the performance of our algorithm. These experiments were all carried out on the Pentium 4 machine.

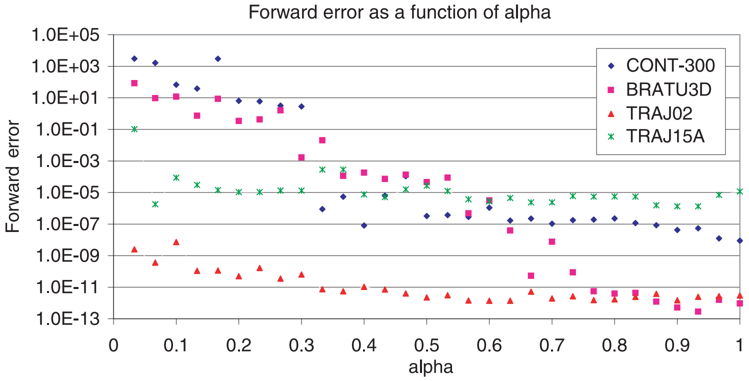


FIG. 5. *Forward accuracy as a function of α on four matrices.*

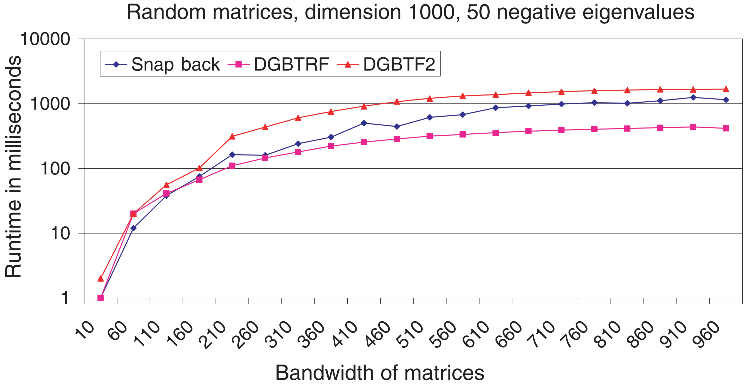


FIG. 6. *Performance as a function of the bandwidth.*

Figure 6 shows the performance of our algorithm relative to LAPACK's implementations of GEPP. The experiment was performed on random 1000-by-1000 matrices with 50 negative eigenvalues. We generated the matrices as follows. We start with a real normally distributed random nonsymmetric matrix T, and compute its QR factorization, $T = QR$, where Q is unitary and R is upper triangular. We then select 1000 uniform random values w_i between 0 and 25. From the w_i's we construct a diagonal matrix Λ with diagonal elements $\lambda_i = (-1)^{[i \le \nu]} 2^{w_i}$, where $[i \le \nu]$ is 1 when $i \le \nu$ and 0 otherwise. Next, we compute $Q\Lambda Q^T$, which has eigenvalues λ_i, exactly ν of which are negative and the rest positive. Finally, we apply the SBR library [3] to $Q\Lambda Q^T$ with the target bandwidth as input. This unitarily reduces $Q\Lambda Q^T$ to a banded symmetric matrix A with the same spectrum as $Q\Lambda Q^T$ and Λ. By the construction of the λ_i's, A is ill conditioned but far from numerical rank deficiency in double-precision IEEE 754 arithmetic.

The results show that our algorithm is always faster than LAPACK's subroutine DGBTF2, which is not blocked for cache efficiency. When the bandwidth is small (e.g., 60 in this experiment), our algorithm is also faster than DGBTRF, which partitions the matrix for cache efficiency. For matrices with higher bandwidth, DGBTRF is faster than our algorithm.

Figure 7 presents similar results on real-world, nonrandom matrices, S3RMT3M1

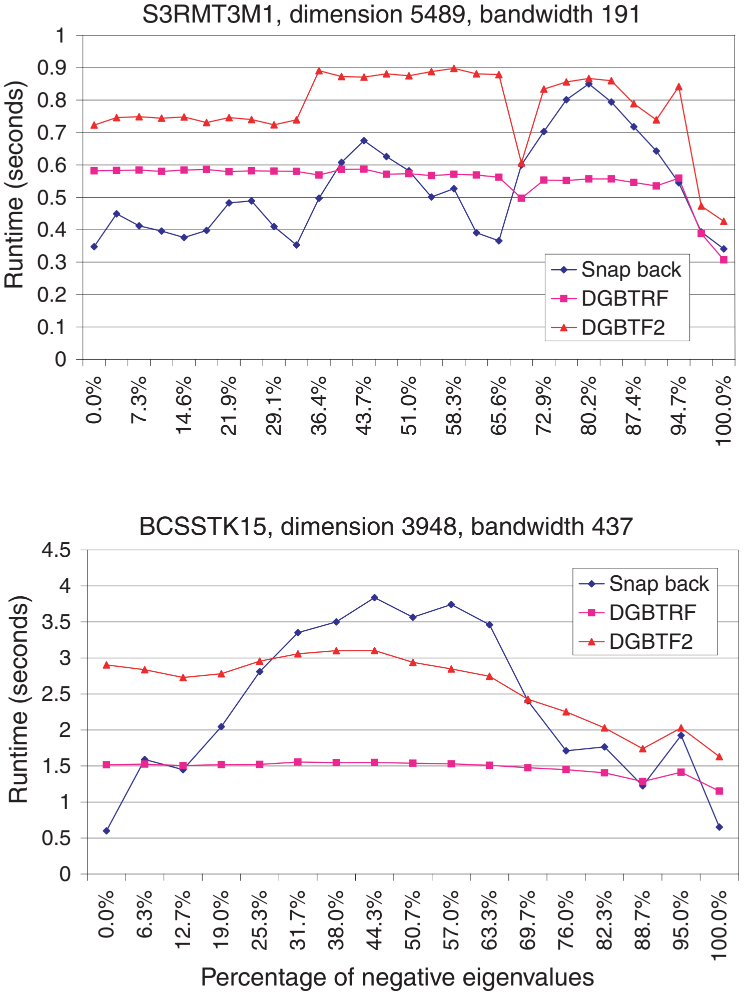


FIG. 7. *Performance on two real-world matrices, shifted to produce different numbers of negative eigenvalues.*

and BCSSTK15. Both are available from public sparse-matrix collections.[3] The figure plots the performance of the three algorithms on shifted versions of the two matrices. All the shifts are half-way between eigenvalues, so the matrices are relatively well conditioned. On S3RMT3M1, which has a relatively narrow bandwidth, our algorithm outperforms both GEPP implementations at many inertia values, and is always at least as fast as DGBTF2. On BCSSTK15, which has a much larger bandwidth, our algorithm is sometimes faster and sometimes slower than DGBTF2 but usually slower than DGBTRF. On this matrix our algorithm exhibits another behavior: higher performance at the ends of the inertia axis than in the middle. We further explore this behavior below.

In another experiment on real-world matrices, we compared the performance of our algorithm to that of DGBTF2 and DGBTRF on the 33 matrices from the Gould–

[3] For example, from http://math.nist.gov/MatrixMarket/.

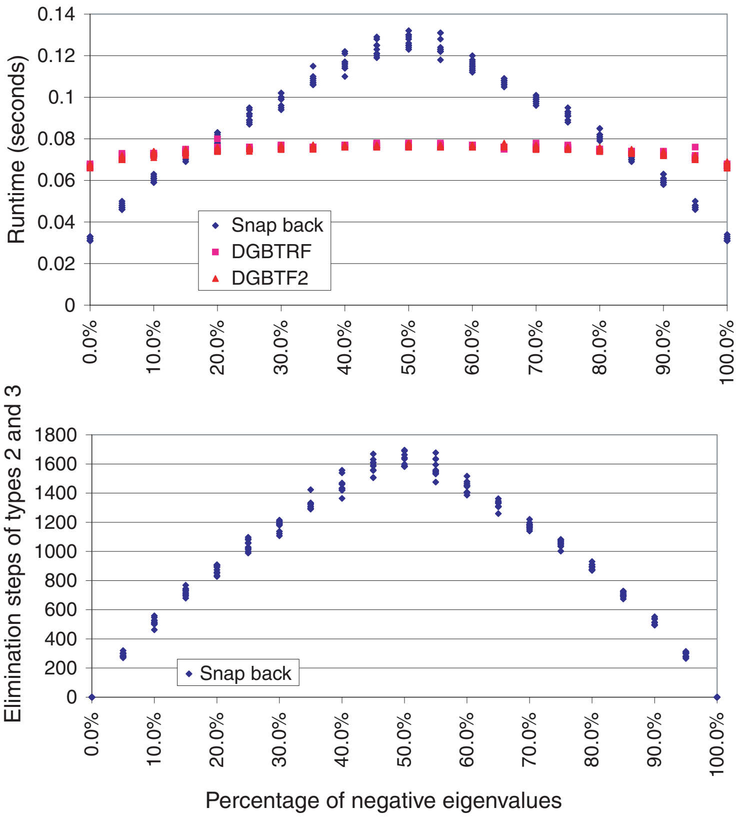


FIG. 8. *Running times of our algorithm on random* 5000*-by-*5000 *matrices with bandwidth* 50*, and the number of type* 2 *and type* 3 *eliminations steps (combined). Each dot corresponds to one matrix.*

Scott study, on the Opteron machine. Most of these matrices have high bandwidths. Our algorithm delivered similar performance to DGBTF2, but DGBTRF was much faster than our algorithm. We omit the detailed results of this experiment.

Figure 8 presents the results of another experiment. For this experiment, we used random 5000-by-5000 matrices with bandwidth 50 and with varying inertia, 10 matrices for each inertia value. We used the Pentium 4 machine for the experiment. The data shown in the top graph of Figure 8 shows that the algorithm runs fastest when most of the eigenvalues have the same sign, and it degrades as the ratio of positive to negative eigenvalues nears 1. This is consistent with the results on some real-world matrices, such as BCSSTK15. The algorithm does not always exhibit this behavior, as the results on S3RMT3M1 show, but it often does.

The bottom graph in Figure 8 explains this phenomenon. The graph shows that near the middle of the inertia axis, the number of type-two and -three elimination steps grows. These elimination steps are more expensive than type-one steps, and they produce more fill, so a larger number of these steps slows down the algorithm. It appears that at least in some cases, the number of type-two and -three steps depends on the inertia.

6.5. Comparisons to other symmetric-indefinite factorization codes. In addition for the tests introduced in this section, we have performed some tests on the SBR library. We have found that the performances of SBR and of our algorithm are almost incomparable: SBR is about three orders of magnitude (about 1000 times) slower.

We were not able to obtain the code of Jones and Patrick [14]. Therefore, we did not compare our algorithm to theirs. We believe that when matrices have only few negative (or few positive) eigenvalues, the behavior of our algorithm and of Jones and Patrick's Bunch–Kaufman code are similar. However, Jones and Patrick's algorithm is not designed for general symmetric banded matrices; it may suffer catastrophic fill when the ratio of positive to negative eigenvalues is near 1, whereas our algorithm degrades smoothly toward this case, and the degradation is never catastrophic (because the bandwidth of the reduced matrices is bounded).

7. Conclusions.

The algorithm that we propose in this paper is the first banded symmetric direct solver that exploits symmetry and achieves an $O(nm^2)$ running time and an $O(nm)$ storage requirement.

The reduced matrices in our algorithm may fill somewhat, but they remain banded. This behavior is similar to that of Gaussian elimination with partial pivoting (GEPP) when applied to a banded matrix. However, representation of the factors in our algorithm can require more memory than the representation of the GEPP factors. Also, our algorithm represents the factors in a product form, consisting of several elementary transformation matrices per elimination step. This representation does not allow our algorithm to be easily blocked for cache efficiency, so for large m and highly-indefinite matrices, our algorithm can be slower than the blocked version of GEPP.

When most of the diagonal elements are large enough to be used as 1-by-1 pivots, our algorithm performs much less work than GEPP. The performance of the algorithm seems to be related to the ratio of the numbers of positive and negative eigenvalues. When there are only a few negative (or only a few positive) eigenvalues, our algorithm uses mostly cheap symmetric Gaussian reduction steps. When there are many eigenvalues of both signs, the algorithm must resort to more expensive Givens and unsymmetric Gaussian reduction steps, so its performance degrades. But even when m is large and A is highly indefinite, our new algorithm is competitive with the unblocked version of GEPP. For small m, our algorithm outperforms even the blocked version of GEPP.

The new algorithm is reliable when implemented in floating-point arithmetic. More specifically, we believe that the algorithm is backward stable, but we formally show only a weaker result: that the element growth is bounded by 4^{n-1}. That is, the entries of the reduced matrices are at most a factor of 4^{n-1} larger in absolute value than the entries of A. In most of the existing elimination algorithms, including GEPP, Bunch–Kaufman and Aasen, a result of this type holds (2 in GEPP, 2.57 in Bunch–Kaufman and 4 in Aasen) and can be used to show backward stability. The existence of such a bound on the growth, in both existing algorithms and in our new algorithm, reflects a careful numerical design whose goal is to avoid catastrophic cancellation. In particular, the growth bound that we show in section 5 holds thanks to an intricate elimination strategy that is designed to simultaneously avoid growth, maintain symmetry, and maintain the band structure. Also, the elements of the factors in our method are bounded in magnitude by the maximum of 1 and the elements of the reduced matrices. We note that the backward stability of the Aasen and Bunch–

Kaufman algorithms, which were proposed in 1971 and 1977, respectively, was only formally proved by Higham in [11, 12].

Numerical experiments show that in practice, growth is almost always much smaller than the worst-case bound, and residuals are small, suggesting that the algorithm is backward stable. Furthermore, the accuracy (forward errors) of our algorithm is similar to that of GEPP even on notoriously difficult matrices. This suggests that the accuracy problems in Bunch–Kaufman, reported in [2], are not present in our algorithm.

This paper leaves a few interesting questions open.

- Can this algorithm be blocked for cache efficiency? That is, can this algorithm be accelerated by exploiting fast level-3 BLAS subroutines?
- Is this algorithm backward stable? Our bound on the element growth, the boundedness of the factors, and our numerical experiments suggest that it is, but we have not formally proved backward stability.
- Can a similar elimination scheme be developed for symmetric indefinite matrices with a general sparsity pattern?

REFERENCES

[1] J. O. Aasen, *On the reduction of a symmetric matrix to tridiagonal form*, Nordisk Tidskr. Informationsbehandling (BIT), 11 (1971), pp. 233–242.

[2] C. Ashcraft, R. G. Grimes, and J. G. Lewis, *Accurate symmetric indefinite linear equation solvers*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 513–561.

[3] C. H. Bischof, B. Lang, and X. Sun, *A framework for symmetric band reduction*, ACM Trans. Math. Software, 26 (2000), pp. 581–601.

[4] J. R. Bunch and L. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comput., 31 (1977), pp. 163–179.

[5] I. A. Cavers, *A hybrid tridiagonalization algorithm for symmetric sparse matrices*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 1363–1380.

[6] I. S. Duff and J. K. Reid, *MA27 – A Set of Fortran Subroutines for Solving Sparse Symmetric Sets of Linear Equations*, Tech. report AERE R10533, AERE Harwell Laboratory, London, UK, 1982.

[7] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[8] I. S. Duff and J. K. Reid, *MA47, a Fortran Code for Direct Solution of Indefinite Sparse Symmetric Linear Systems*, Tech. report RAL-95-001, Rutherford Appleton Laboratory, Didcot, Oxon, UK, 1995.

[9] L. Fox, H. D. Huskey, and J. H. Wilkinson, *Notes on the solution of algebraic linear simultaneous equations*, Quart. J. Mech. Appl. Math., 1 (1948), pp. 149–173.

[10] N. I. M. Gould and J. A. Scott, *Complete Results from a Numerical Evaluation of hsl Packages for the Direct-Solution of Large Sparse, Symmetric Linear Systems of Equations*, Tech. report, Numerical Analysis Internal Report 2003-2, Rutherford Appleton Laboratory, 2003. Available online from http://www.numerical.rl.ac.uk/reports/reports.shtml.

[11] N. J. Higham, *Stability of Aasen's method*, manuscript, 1997.

[12] N. J. Higham, *Stability of the diagonal pivoting method with partial pivoting*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 52–65.

[13] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, 2002.

[14] M. T. Jones and M. L. Patrick, *Bunch-Kaufman factorization for real symmetric indefinite banded matrices*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 553–559.

[15] B. Lang, *A parallel algorithm for reducing symmetric banded matrices to tridiagonal form*, SIAM J. Sci. Comput., 14 (1993), pp. 1320–1338.

[16] K. Murata and K. Horikoshi, *A new method for the tridiagonalization of the symmetric band matrix*, Information Processing in Japan, 15 (1975), pp. 108–112.

[17] B. N. Parlett and J. K. Reid, *On the solution of a system of linear equations whose matrix is symmetric but not definite*, BIT, 10 (1970), pp. 386–397.

[18] H. Rutishauser, *On Jacobi rotation patterns*, Proc. Sympos. Appl. Math. 15, Amer. Math. Soc., Providence, RI, 1963, pp. 219–239.

[19] H. R. Schwarz, *Tridiagonalization of a symmetric band matrix*, Numer. Math., 12 (1968), pp. 231–241.

[20] G. W. Stewart, *The economical storage of plane rotations*, Numer. Math., 25 (1976), pp. 137–138.

[21] J. H. Wilkinson, *Error analysis of direct methods of matrix inversion*, J. Assoc. Comput. Math., 8 (1961), pp. 281–330.

[22] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1963.