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 maintains symmetry at all times, but quickly destroys the band structure. Gaussian reduction to tridiagonal also maintains symmetry but destroys the band structure. 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 the factors mostly symmetric (unlike any previous method), and it is fast.

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

1. Introduction

We propose a new method for the direct solution of a linear system of equations $Ax = b$ where A is a banded symmetric indefinite matrix with half bandwidth m. Our method maintains a bounded banded structure in both the reduced matrix and in the factors, it produces perfectly symmetric reduced matrices and mostly symmetric factors, and it is fast. No existing factorization method achieves all of these goals. 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 element is too small, we use a more complex elimination step, which usually eliminates two rows and columns. 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 unsymmetry in the factors depends on the number of these complex elimination steps. Section 2.1 describes the snap-back pivoting method in detail, and

This research was supported in part by an IBM Faculty Partnership Award,by grant 572/00 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.

Section 4 proves the correctness of the method (more specifically, of the banded variant that we describe in Section 3).

Existing direct factorization methods suffer from serious defects when applied to banded symmetric matrices. Unless the matrix is definite, either some form of pivoting or an orthogonal reduction must be employed to ensure stability. Of the existing methods, only Gaussian elimination with partial pivoting (GEPP) is really applicable to banded symmetric matrices. 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 of 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 (U) have half bandwidth at most $2m$. The left lower triangular factor (L), and also the lower triangular part of the reduced matrices, maintain half bandwidth m.

There are two factorization techniques that employ pivoting but maintain symmetry in the factors and in the reduced matrices. One technique, which is used in several algorithms [4, 8, 7, 9, 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 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 [8, 7, 9], but without any a-priori bound on the fill that pivoting might cause.

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, 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 regain 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. In 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 it 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.

Our banded snap-back factorization, which we describe in Section 3, combines the positive aspects of all of these algorithms. Like GEPP, its computational cost is proportional to nm^2, and its storage costs are proportional to nm. In particular, the bandwidth of the reduced matrix and of the factors remain proportional to m. Like the Bunch-Kaufman and similar 2-by-2 block algorithms, its computational and storage requirements are superior to those of GEPP when most of the diagonal elements are large enough to be used as 1-by-1 pivots. In practice, it is much faster than GEPP when m is relatively small and when A has only few positive or only few negative eigenvalues. But even when m is large and A is highly indefinite, our new algorithm is competitive with GEPP.

Compared with banded GEPP, our algorithm does two drawbacks. First, the half-bandwidth of the factors in our algorithm can exceed that of GEPP by a constant factor. Second, we have not developed a version of our algorithm that is blocked for cache efficiency, so for large m and highly-indefinite matrices, our algorithm can be slower than GEPP, which can be blocked.

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 grown ρ_A 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. Numerical experiments show that in practice, growth is much smaller than the theoretical 4^{n-1} bound (we have not observed growth larger than about 250), and forward errors are small, suggesting that the algorithm is backward stable. We note that the backward stability of the Aasen and Bunch-Kaufman algorithms, which were proposed in 1971 and 1977, respectively, were only formally proved by Higham in [12, 13].

Numerical results, which we present in Section 6, indicate not only that the algorithm is stable, but also that it is efficient. In particular, we show that in most cases the new algorithm is superior to the LAPACK implementation of banded GEPP, even though that implementation is blocked for cache efficiency and uses the level-3 BLAS.

Our research also leaves several interesting questions open: Can the number of admissible 1-by-1 pivots be related to the number of positive and negative eigenvalues of A? Can the algorithm be blocked for cache efficiency? Is snap-back pivoting applicable to general sparse matrices? We summarize our results and discuss these questions in Section 7.

2. 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. The subsection that follows provides all the mathematical details of the elementary transformations that are involved, and the last subsection proves that the factorization that our method computes always exists, that it is indeed quasi-symmetric, and that our algorithm is correct.

2.1. **An Overview of the Elimination Process.**

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, $\times$ symbols denote nonzero elements in a symmetric matrix (that is, if a_{ij} is denoted by a $\times$, 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 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 is always nonzero, using a Givens rotation that transforms the first and last rows. If element $\left[Y^{-1}AY^{-T}\right]_{11} = 0$, we get the following form,

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

The symbol $\ominus$ denotes a nonsymmetric nonzero, i.e., an a_{ij} that might be nonzero and might be different from a_{ji}. If $\left[Y^{-1}AY^{-T}\right]_{11} \neq 0$, then we get another form, which turns out to be simpler,

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

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),

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

(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,

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

If $\left[Y^{-1}AY^{-T}\right]_{11}$ was zero 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

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

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 first 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

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

We now use a sequence of symmetric Gauss or Givens transforms to eliminate all but 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,

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

The last step is to eliminate the off-diagonal 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.

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

This concludes the overview of our new elimination method.

2.2. **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 that 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

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

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 in principle be a Givens rotation, but to save work, we use 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. This leads to the following structure for Y

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

Each 2-by-2 block is a product of a row exchange (perhaps identity) and a row operation on one row.

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 that 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 switch to an elimination step of the third kind. The matrix P now brings the last row and column to be the second, 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 do 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.

3. 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. It turns out that with a careful selection of transformations, the bandwidth of the reduced 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

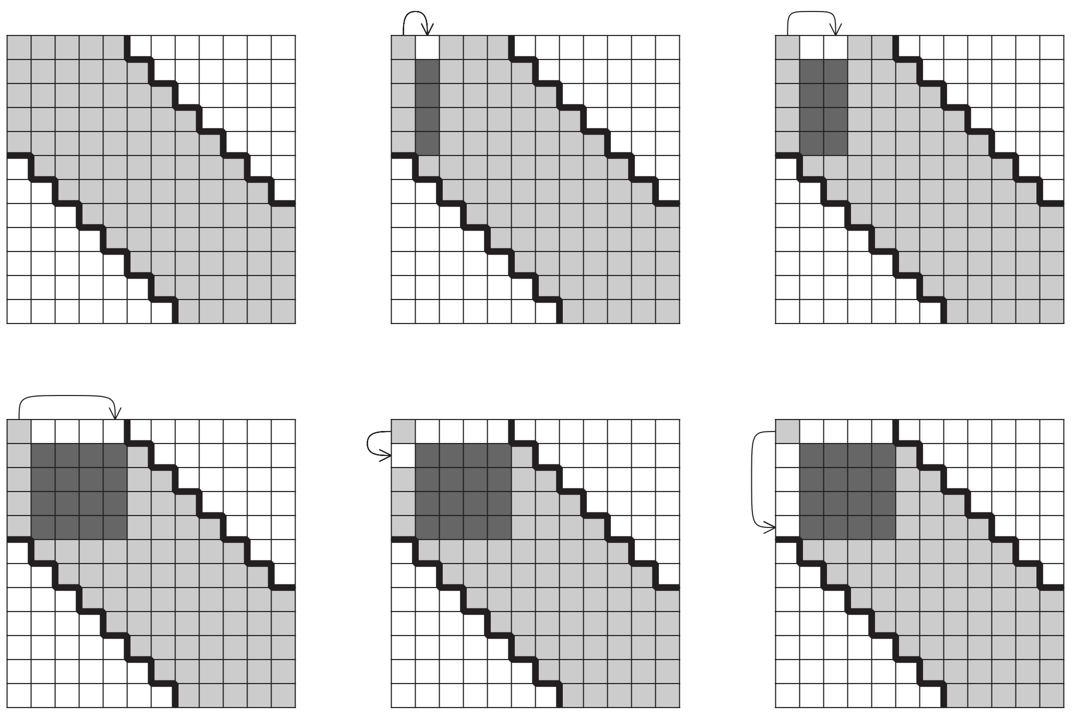

FIGURE 3.1. An elimination step of the first kind applied to a banded matrix. Here the matrix is 12-by-12 and the half-bandwidth is 4, which means that the matrix has 9 nonzero diagonals. The illustrations depict nonzero elements in gray; light gray signify that an element has not been modified in this elimination step, and dark gray signify that an element has been modified. The original profile of the matrix is indicated by heavy black border. In the case shown here, of an elimination step of the first kind, the offdiagonal elements in the first row and column are eliminated using a sequence of column and row operations. The arrows show which column/row are scaled and subtracted from which other column/row to annihilate a nonzero.

structural zeros occur. This case is illustrated in Figure 3.1. Elimination steps of the second and third kinds require more care.

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 3.2 (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 structural nonzeros in row/column 1 (m' might be larger than m if the bandwidth increased 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

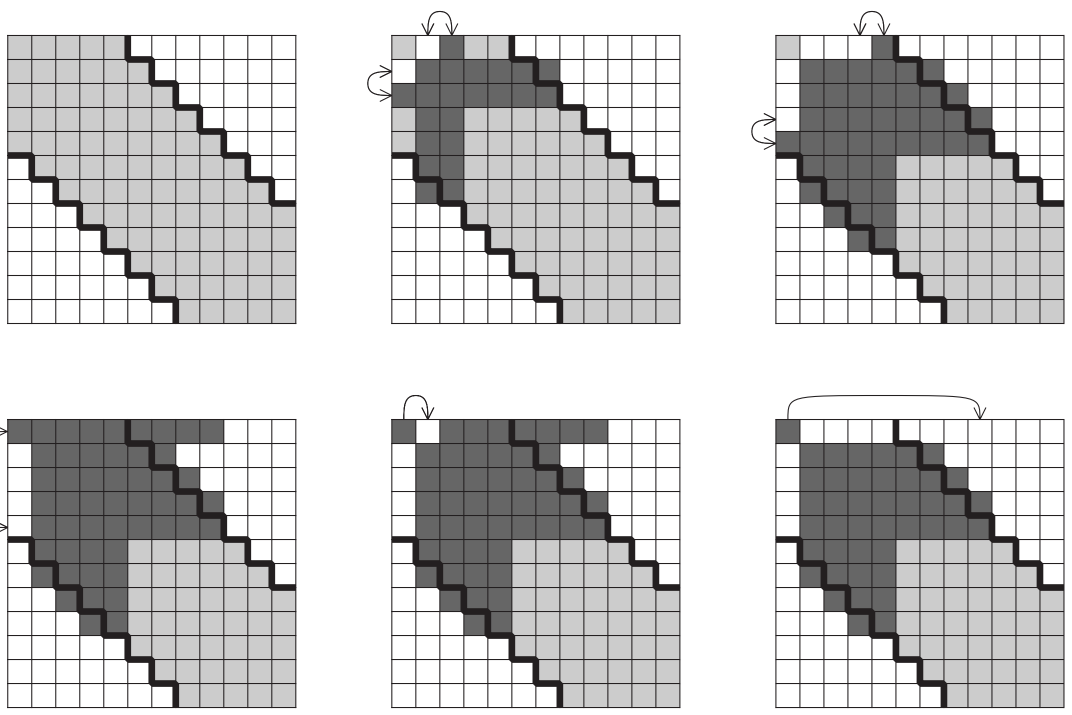

FIGURE 3.2. An elimination step of the second kind. The step starts with symmetric row and column operations that annihilate all but the last element in the first row and column (top row). To ensure stability, we use either Givens transforms or row/column exchanges followed by Gauss transforms. In either case, the two rows and columns that are involved might be modified, not just one. This increases the half-bandwidth in the relevant rows/columns by one. We denote Givens transforms by double arrows. Next, a Givens transform eliminates the last nonzero in the first column; this causes additional fill in the first row (bottom left). Finally, the offdiagonals in the first row are eliminated using a series of Gauss transforms applied from the right (column operations).

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 3.3. 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

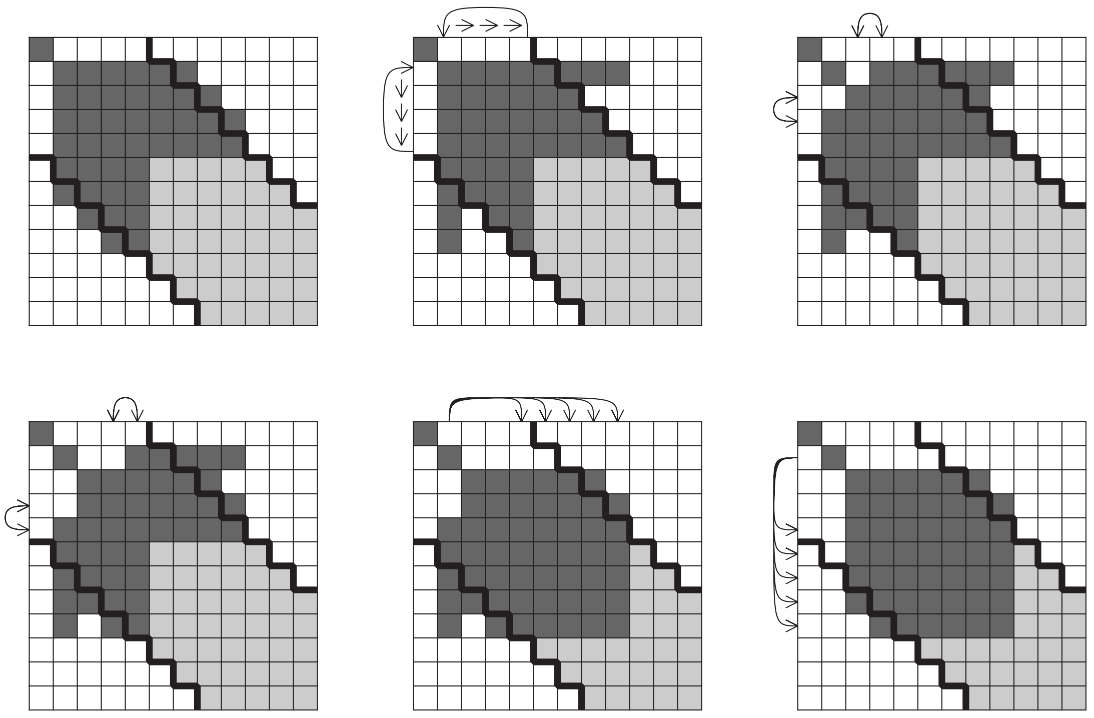

FIGURE 3.3. An elimination step of the third kind. The top right illustration describes the matrix after all the operations of a second-kind elimination step have been applied. The algorithm switched to a third-kind elimination because row 5 cannot be stably scaled back to symmetry with column 5. The elimination continues with a cyclic permutation that brings row and column 5 to position 2 and shifts rows/columns 2, 3, and 4 one position down (top center). Next, a series of Givens transforms annihilates offdiagonal nonzeros in row/column 2 (bottom left). Not all the offdiagonals in row/column 2 are annihilated in this way. At some point, the algorithm switches to Gauss transforms to avoid expanding the band too much; these are first applied to row 2 (bottom center) and then to column 2 (bottom right). The switching is done to maintain the band, as explained in the text.

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.

3.1. **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 structural 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 3.1. *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 3.3 and 3.4, which we state and prove below. But we first state and prove a technical lemma:

Lemma 3.2. *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 $2, ..., r^1_{\widetilde{A}^{(k)}}$ 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 $2, ..., r^1_{\widetilde{A}^{(k)}}$ behaves in a symmetric way. For j such that $1 \leq j < r^1_{\widetilde{A}^{(k)}} - 1$ we have (see figure 3.4)

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

and

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

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

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

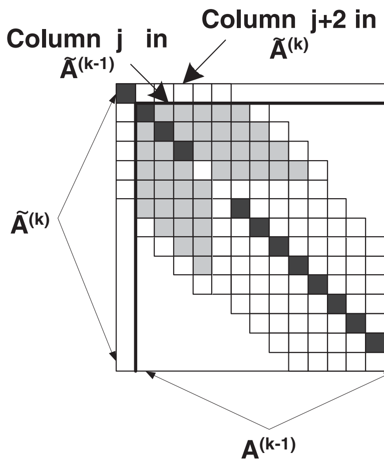
Column j in $\tilde{A}^{(k-1)}$
Column j+2 in $\tilde{A}^{(k)}$
$\tilde{A}^{(k)}$
$A^{(k-1)}$

FIGURE 3.4. This figure illustrates the proof of lemma 3.2

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 3.3. *Denote by η_M the set of structural non-zero 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 l-th 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 that we consider their influence on the structure of $A^{(k)}$ 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 non-symmetric 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

$$\eta_B \subseteq \eta_{\widetilde{A}^{(k+1)}} \tag{3.1}$$

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) $1, ..., r^{(1)}_{A^{(k+2)}} - 1$ of B is moved one location to the right (down). From the combination of 3.1 with lemma 3.2 we have that the structures of columns (rows) $2, ..., r^{(1)}_{A^{(k+2)}}$ in C are contained in the structures of these columns (rows) in $\widetilde{A}^{(k+1)}$ respectively. We get

$$\eta_{C_{[k]}} \subseteq \eta_{\widetilde{A}^{(k+1)}_{[k]}} \tag{3.2}$$

where $M_{[k]}$ denotes the $k \times k$ 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

$$r^1_{A^{(k+2)}} \leq r^1_{\widetilde{A}^{(k+2)}} \leq r^2_{\widetilde{A}^{(k+2)}} + 1 \leq r^1_{\widetilde{A}^{(k+1)}} + 1 \tag{3.3}$$

by the induction assumption, lemma 3.2 and the trivial fact $r^2_{\widetilde{A}^{(k+2)}} \leq r^1_{\widetilde{A}^{(k+1)}}$. Denote the matrix we get by applying X^{-1} on C by D. By combining 3.3 with 3.2 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 non-symmetric gauss reduction. The effect on the fill that this operation may have on D is limited to elements d_{ij}, where

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

The fact that $\widetilde{A}^{(k+1)}$ has a non-zero element in entry $(r^1_C + 1, r^1_{A^{(k+2)}})$ (because B has a non zero there, and due to 3.1) and lemma 3.2 bring us to the conclusion that the elements of $\widetilde{A}^{(k)}$ in entries i, j where $r^1_{A^{(k+2)}} - 1 \leq i, j \leq r^1_C$ are all non-zeros. We got

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

□

We now prove the following lemma on ALG2:

Lemma 3.4. *Let m be the half-bandwidth of A. Then $r^i_{\widetilde{A}^{(k)}} < 2m$ for $0 < k \leq n$, $1 \leq i \leq k$.*

Proof. Let us focus in a non-zero 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 non-zero is the first to fill the criterion and that the reduced matrix $F = \widetilde{A}^{(k)}$ is the first in which the non-zero appears (see figure 3.5:A). The appearance of the non-zero in the j-th column of the i-th row of $\widetilde{A}^{(k)}$ implies that the $i+1, j+2$ element in $\widetilde{A}^{(k+1)}$ is non-zero (figure 3.5: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 3.5: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 3.5: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 3.5:E). This implies that

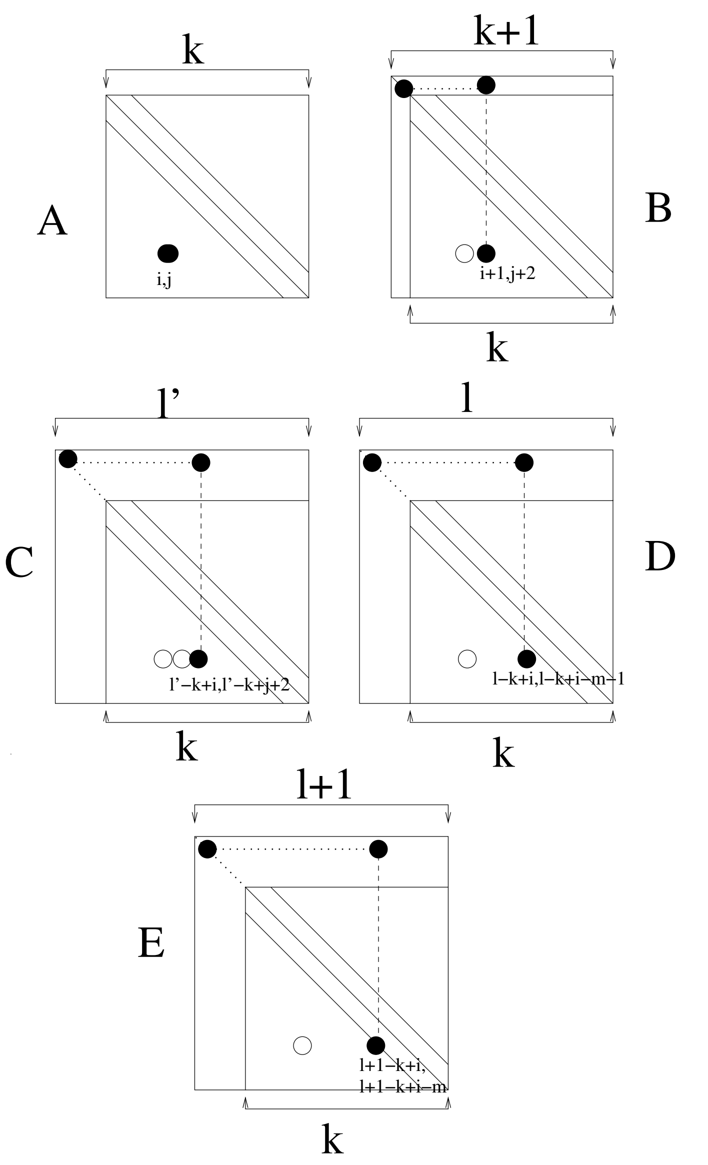

FIGURE 3.5. This figure illustrates the main steps in the proof of lemma 3.4.

$r^1_{\widetilde{A}^{(l+1)}} \geq 2i - j - 2m - 1$. From 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. $\square$

3.2. **Storing the factors.** In this subsection we explain the way by which we store the information about the transformations that we apply during the reduction process. This information is crucial for solving a linear system, which is the main motivation of our work.

Our algorithm combines symmetric and non-symmetric reduction transformations. The representation of these transformations should exploit the symmetry whenever exists in order to reduce the amount of consumed memory. For the same reason, we are interested in a representation that is constructed in-place, namely, in the reduction step of $A^{(k)}$, the s ($s = 1$

or $s = 2$) reduced columns/rows of $A^{(k)}$ will be replaced by the reduction transformations. The right bottom $k - s \times k - s$ block of $A^{(k)}$ will be replaced by $A^{(k-s)}$. Additional memory should be supplied in case the transformations representation consume more memory than the reduced columns/rows.

Gaussian elimination transformations that are applied during a single reduction step of the first reduction type, are simply stored in one column. This column replaces the first column of $A^{(k)}$. The first element of the column is $a^{(k)}_{1,1}$. The element located at entry $i, 1$, where $1 < i \leq r^1_{A^{(k)}} + 1$, is $a^{(k)}_{i,1}/a^{(k)}_{1,1}$.

The second type of reduction is non-symmetric. As a consequence, the applied transformations consume more memory then the Gaussian elimination transformations. More specifically, the transformations of a reduction of the second type which are applied on $A^{(k)}$ are stored in place of the first row and first column of $A^{(k)}$. Additional memory is supplied if needed. The first column contains a representation of Y, the single step of GEPP (or givens rotation) that is applied from the left and also the scaling transformation. All this information can be stably compacted in place of the first column using the technique in [20]. The first row of $A^{(k)}$ is changed by the transformations applied from the left. This changed row represents the right transformation which is applied on $A^{(k)}$ in order to get $A^{(k-1)}$. It should be noted, however, that this row may consume more memory then the first row of $A^{(k)}$, and therefore, additional memory should be supplied if needed. The amount of additional memory is limited by at most 4 times the memory needed by a single row of A.

The third type reduction starts with a sequence of transformations which is identical to the one applied for the second type reduction (not including the scaling transformation). The transformations that belong to this sequence are stored as in the second type reduction. The rest of the third type reduction transformations are stored in place of the second column of $A^{(k)}$. The transformation X is stored instead of elements $a^{(k)}_{3,2}$, $a^{(k)}_{4,2}$, ... , $a^{(k)}_{r^1_{A^{(k)}},2}$ in a similar way to the one by which Y is stored. The non-symmetric gauss transformation is stored in place of the rest of the elements in the second column of $A^{(k)}$.

4. Existence and Correctness

We now prove that our algorithm is correct and that the factorization that we presented always exists. Formally, we prove the following theorem.

Theorem 4.1. *Let A be a real symmetric matrix. Then A can always be reduced to diagonal form using the elimination method presented in Sections 2 and 3.*

The proof consists of a series of simple lemmas that show that the elementary steps that we use always exist and that at the end of a full elimination step, after one or two rows and columns have been eliminated, the reduced matrix remains symmetric. The matrices that we refer to, L, Y, G, U, S, P, X, K, and $\hat{K}$, are fully described in sections 2 and 3.

Lemma 4.2. *Given a real symmetric matrix A, assume that the $(1, 1)$ element in A is nonzero. Then L exists and $L^{-1}AL^{-T}$ is symmetric.*

Proof. Trivial; Omitted. □

Lemma 4.3. *Given a real symmetric matrix A, Y exists and $Y^{-1}AY^{-T}$ is symmetric.*

Proof. In the previous section Y^{-1} is defined as a product of matrices . The rightmost factor in Y^{-1}annihilates the element $(2, 1)$ using a row operation involving the second and third rows. In case the second element is larger in its absolute value than the third element, the rightmost factor includes the permutation that exchanges the rows. If the second and third elements are both zeros, then the term may be the identity matrix.

Similarly, each term appearing in the definition of Y^{-1} annihilates one nonzero in the first column. All the structural nonzeors in the first column, excluding the first and last nonzeros, are annihilated from top to bottom. Clearly, $Y^{-1}AY^{-T}$ is symmetric. □

Lemma 4.4. *Given a real symmetric matrix A, let $(r,1)$ be the last structural nonzero in the first column of $Y^{-1}AY^{-T}$. Then G and U exist, and there is a constant c s.t.*

$$c[G^{-1}Y^{-1}AY^{-T}U^{-1}]_{r,i} = [G^{-1}Y^{-1}AY^{-T}U^{-1}]_{i,r}$$

for $i \neq r$ and

$$[G^{-1}Y^{-1}AY^{-T}U^{-1}]_{j,i} = [G^{-1}Y^{-1}AY^{-T}U^{-1}]_{i,j}$$

for $i,j \neq r$.

Proof. We can eliminate $[Y^{-1}AY^{-T}]_{r,1}$using a Givens rotation G^{-1}that involves the first and the r-th rows. By applying G^{-1}, all the elements in the r-th row of $[Y^{-1}AY^{-T}]$, excluding the first one and the r-th one, are just scaled by some constant, since the only non-diagonal non-zeros in the first column and row of $Y^{-1}AY^{-T}$ are located at $(r,1)$ and at $(1,r)$. Let c be this constant, and so by lemma 4.3,

$$c[G^{-1}Y^{-1}AY^{-T}]_{r,i} = [G^{-1}Y^{-1}AY^{-T}]_{i,r}$$

for $i \neq r, 1$. The 2-elements vector $([Y^{-1}AY^{-T}]_{1,1}, [Y^{-1}AY^{-T}]_{r,1})$ cannot be zero (If this vector was zero, it would imply that all the elements in the first column are zero, and this, in turn, means that we could skip the elimination step). This with the fact that Givens rotation does not change the norm of a vector it is applied on, means that $[G^{-1}Y^{-1}AY^{-T}]_{1,1}$is non-zero. We have that U exists. By applying U^{-1}, all the non-diagonal elements in the first row of $[G^{-1}Y^{-1}AY^{-T}]$ are vanished. So we have

$$c[G^{-1}Y^{-1}AY^{-T}U^{-1}]_{r,1} = [G^{-1}Y^{-1}AY^{-T}U^{-1}]_{1,r} = 0.$$

Elements that are not of the first or r-th row are not touched by G^{-1} and U^{-1} and therefore, by lemma 4.3,

$$[G^{-1}Y^{-1}AY^{-T}U^{-1}]_{j,i} = [G^{-1}Y^{-1}AY^{-T}U^{-1}]_{i,j}$$

for $i,j \neq r$. □

Lemma 4.5. *Given a real symmetric matrix A, assume the constant c from lemma 4.4 differs from 0 and let $(r,1)$ be the last structural nonzero in the first column of $Y^{-1}AY^{-T}$. Then S exists and $S^{-1}G^{-1}Y^{-1}AY^{-T}U^{-1}$ is symmetric.*

Proof. Let S be the diagonal matrix in which all the diagonal elements excluding the r-th one are 1s, and the r-th diagonal element is c. From lemma 4.4 we have that $S^{-1}G^{-1}Y^{-1}AY^{-T}U^{-1}$ is symmetric. □

Lemma 4.6. *Given a real symmetric matrix A, let $(r,1)$ be the last structural nonzero in the first column of $Y^{-1}AY^{-T}$. Then*

$$[P^TG^{-1}Y^{-1}AY^{-T}U^{-1}P]_{3:n,3:n}$$

is symmetric and there is a constant c s.t.:

$$c[G^{-1}Y^{-1}AY^{-T}U^{-1}]_{2,i} = [G^{-1}Y^{-1}AY^{-T}U^{-1}]_{i,2}$$

for $i \neq 2$.

Proof. Immediate from lemma 4.4 and from the fact that P is a permutation matrix that brings the r-th row/column to the second place, and moves rows/columns $2, ..., r-1$ to places $3, ..., r$ respectively. □

Lemma 4.7. *Assume A is a real symmetric matrix and let $(r, 1)$ be the last structural nonzero in the first column of $Y^{-1}AY^{-T}$. Then X exists,*

$$[X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}]_{3:n,3:n}$$

is symmetric and there is a constant c s.t.:

$$c[X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}]_{2,i} = [X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}]_{i,2}$$

for $i \neq 2$.

Proof. Existence of X comes from the same considerations by which we concluded the existence of Y in lemma 4.3. To show that applying X^{-T} from the left indeed annihilates elements

$$[P^TG^{-1}Y^{-1}AY^{-T}U^{-1}P]_{2,3:r-1},$$

we assume without loss of generality that $c \neq 0$ and so, if we denote

$$F = P^TG^{-1}Y^{-1}AY^{-T}U^{-1}P,$$

then $[F]_{j,2} = 0$ if and only if $[F]_{2,j} = 0$. We then have:

$$\frac{[F]_{i,2}}{[F]_{j,2}} = \frac{c[F]_{i,2}}{c[F]_{j,2}} = \frac{[F]_{2,i}}{[F]_{2,j}}, \quad 2 < i, j \leq r, \ [F]_{j,2} \neq 0$$

This with the way of constructing X, implies that X^{-T} annihilates elements $[G^{-1}Y^{-1}AY^{-T}U^{-1}]_{2,3:m-1}$.

From lemma 4.6 and the facts that X^{-1} is applied symmetrically and that it manipulates only rows and columns $3, ..., r$, we have that

$$[X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}]_{3:n,3:n}$$

is symmetric. In addition, it is easy to show that the values in the second row and column of

$$X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}$$

are the same as the values in the second row and column of $P^TG^{-1}Y^{-1}AY^{-T}U^{-1}P$, respectively. The only exception is the r-th element in

$$X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}$$

which has the maximal value among elements $3, ..., r$ in the second row/column of $P^TG^{-1}Y^{-1}AY^{-T}U^{-1}P$. This with lemma 4.6 implies that there is a constant c s.t.:

$$c[G^{-1}Y^{-1}AY^{-T}U^{-1}]_{2,i} = [G^{-1}Y^{-1}AY^{-T}U^{-1}]_{i,2}$$

for $i \neq 2$. □

Lemma 4.8. *Given a real symmetric matrix A, let $(r, 1)$ be the last structural nonzero in the first column of $Y^{-1}AY^{-T}$ and assume that $[G^{-1}Y^{-1}AY^{-T}U^{-1}]_{r,r}$ differs from 0. Then K and $\hat{K}$ exist, and*

$$K^{-1}X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}\hat{K}^{-1}$$

is symmetric.

Proof. Existence of K and $\hat{K}$ is immediate from the observation that

$$[X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}]_{2,2} \neq 0$$

which is a result of the assumption that $[G^{-1}Y^{-1}AY^{-T}U^{-1}]_{r,r} \neq 0$. For showing symmetry, denote

$$X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}$$

by B and

$$K^{-1}X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}\hat{K}^{-1}$$

by D. Using lemma 4.7 we have for $c \neq 0$:

$$[D]_{i,j} = [B]_{i,j} - \frac{[B]_{i,2}}{[B]_{2,2}}[B]_{2,j} = [B]_{j,i} - \frac{[B]_{2,i}/c}{[B]_{2,2}}c[B]_{j,2} = [B]_{j,i} - \frac{[B]_{2,i}}{[B]_{2,2}}[B]_{j,2} = [D]_{j,i}, \quad i > 2, \ \ j > 2$$

which means that

$$K^{-1}X^{-1}P^TG^{-1}Y^{-1}AY^{-T}U^{-1}PX^{-T}\hat{K}^{-1}$$

is symmetric. Clearly, symmetry holds for $c = 0$. □

5. Numerical Stability: Bounding the Growth

In this section we show that the growth during the algorithm is bounded from above by 4^{n-1}. 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,

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

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 never encountered; the growth in practice is almost always very small. Researchers have shown that when the growth is small, these algorithms are backward stable [12, 13, 21, 22]. Since we never see large growth factors, these algorithms are stable in practice [10, 11]. 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.

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 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 number of the row of the entry with the maximum magnitude in the first column. Using these terms, we use a reduction of the first kind if $\mid a_{1,1}^{(k)} \mid > \alpha \cdot \gamma_1^{(k)}$ or $\mid a_{1,1}^{(k)} \mid \cdot \gamma_t^{(k)} > \alpha(\gamma_1^{(k)})^2$. Otherwise, a reduction of one of the other two types is applied. 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/equal (a threshold may be used) than all the other elements in the 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

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

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 from. 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 elimination, the first annihilated element is the one that 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 non-diagonal 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 the diagonal is 1's, except, maybe, for one diagonal element.

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 $q \leq Q$ such elementary transformations by $B^{(q)}$, and its i, j element by $b_{ij}^{(q)}$.

Lemma 5.1. *A. If $i \geq q+2$ and $j \geq q+2$ then $\mid b_{ij}^{(q)} \mid \leq \nu$.*

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

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

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

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

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

We also have $\mid b^{(q+1)}_{ij} \mid \leq 2\nu$ for $i \geq q + 3$ and $j = q + 2$ and for $i = q + 2$ and $j \geq q + 3$, 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, $\mid b^{(q+1)}_{ij} \mid \leq 2\nu$ for $i \geq q + 3$ and $1 \leq j < q + 2$ and for $1 \leq i < q + 2$ and $j \geq q + 3$. This completes the proof of B.

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

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 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 Bunch-Kaufman 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

$$\mu(1+\frac{1}{\alpha}) = 4\mu$$

We get

$$\alpha = \frac{1}{3}$$

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

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 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 factorization routine expects the input matrix to be in LAPACK symmetric-banded format, using the upper-triangular variant (only the upper triangle is explicitly stored). In this format, a banded matrix with half-bandwidth m and dimension n is stored in a rectangular array with n columns and $m' > m$ rows. Like virtually all the LAPACK routines, our implementation factors the matrix in place, overwriting the matrix with the representation of the factors. Since the bandwidth of the factors may be higher than m, the user must pass the matrix to our factorization routine in an array whose height m' is significantly larger than m. We have found $m' = 3m$ to be a safe value, and of course $m' = 4m$ is provably safe. If the factorization finds, during the factorization, that m' is too small, it returns with an appropriate error code. This input-output format is essentially the same as the format that LAPACK's banded LU-with-partial-pivoting (GEPP) routine uses.

The threshold value α that our implementation uses is $\alpha = 1/3$, the optimal theoretical value. We never observed elimination steps of the second kind with this value, only first and third kind. When α is higher (not shown in any of our experiments), second-kind eliminations also occur.

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 non-blocked factorization codes, such as DGBTF2. From the practical point of view, it is also important to know how does 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 issue that influence the performance of DGBTRF.) We show below that our code is usually faster than DGBTF2, and that it is faster than DGBTRF when the bandwidth is small or when most of the eigenvalues have the same sign.

Comparisons of the performance of our algorithm versus the symmetric band reduction of Bischof, Lang, and Sun [3] showed that our code is often hundreds of times faster. We conclude that the symmetric band reduction is only appropriate for the orthogonal reduction of symmetric matrices to tridiagonal form, and that it is not appropriate as a linear solver. We do not present detailed results.

6.2. **Test Environment.** We conducted the experiments on an 800 MHz Pentium III computer running Windows 2000. We compiled our code, as well as LAPACK, using Microsoft's Visual C++ compiler version 6.0. (We compiled LAPACK from C sources that were generated from the Fortran sources using f2c.) We linked our code, as well as LAPACK, with the ATLAS implementation of the Basic Linear Algebra Subroutines (BLAS), version 3.4.2. We built ATLAS using the gcc compiler under the cygwin environment.

6.3. **Test Matrices.** We used two classes of matrices in the experiments below. The first class consists of nonsingular random banded symmetric matrices with a given dimension n, a given half bandwidth m, and a given inertia. We construct these matrices as follows. We start with a real normally distributed random non-symmetric n-by-n matrix T, and compute its QR factorization, $T = QR$, where Q is unitary and R is upper triangular. We then select n 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 second class of matrices were generated from real-world symmetric positive-definite matrices, obtained from public matrix collections, and shifted by various amounts to produce matrices with different inertias and condition numbers. To produce such matrices, we computed all the eigenvalues λ_i of the input matrix R using MATLAB's eig function. To obtain a (usually) well conditioned matrix with a given number ν of negative eigenvalues, we use

$$A = R - \frac{\lambda_\nu + \lambda_{\nu+1}}{2} I \ .$$

To produce an ill-conditioned matrix with a given inertia, we use $A = R - \lambda_\nu I$. (We note that this method may fail to produce a matrix with a given inertia when there are multiple eigenvalues at λ_ν; for the purpose of our experiments, this is irrelevant.) These two classes of matrices are fairly typical for the matrices that are factored within a shift-invert eigensolvers.

6.4. **Performance and Growth as a Function of Inertia.** The first set of experiments investigates the performance of the new algorithm and the growth in the factors as a function of the negative/positive eigenvalue ratio, which we denote by ϕ. We say that ϕ is balanced when its value is near 1 and unbalanced when its value is near 0 or ∞.

Figure 6.1 shows that the running time of the algorithm is highly dependent on ϕ. When ϕ is unbalanced, the algorithm runs quickly, and when ϕ is balanced, the algorithm is slower. Figure 6.2 explains this phenomenon. It shows that when ϕ is balanced, more third-kind elimination steps are performed. Elimination steps of the third kind are more expensive per-column than steps of the first kind. The large number of third-kind eliminations near $\phi = 1$ also results in denser factors and in higher bandwidths in the trailing submatrices. This is shown in Figures 6.3 and 6.4.

Figures 6.5 and 6.6 show that growth also increases when ϕ is roughly balanced. However, the growth is fairly small in all the experiments, and not very sensitive to ϕ.

6.5. **Comparisons with Banded Gaussian Elimination with Partial Pivoting.** We compared our code to the LAPACK routines DGBTRF and DGBTF2. DGBTRF applies Gaussian elimination with partial pivoting to a banded matrix, not necessarily symmetric. It

FIGURE 6.1. Comparison of runtime as a function of the number of negative eigenvalues. The algorithm is applied to 1000×1000 matrices with half bandwidth 50. The runtime is compared to LAPACK routines. The algorithm is faster then both the blocked and non-blocked LAPACK versions of LU.

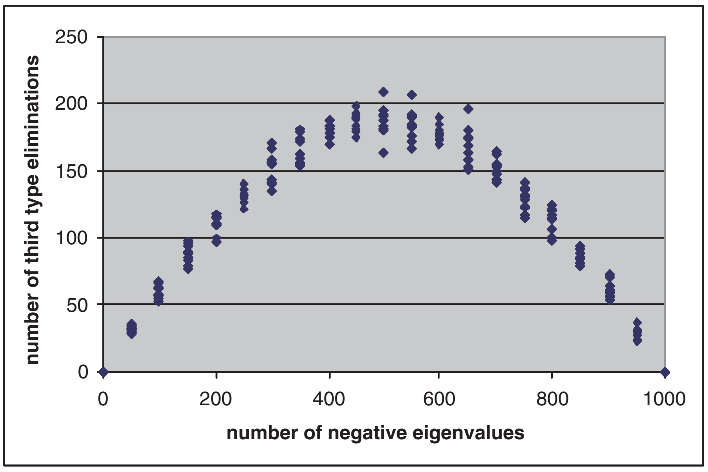

FIGURE 6.2. The number of third type reductions depends on ϕ. For every i in the range 0 to 20, we have generated 10 random banded matrices of size 1000×1000 and bandwidth 50 with $50i$ negative eigenvalues and $1000 - 50i$ positive eigenvalues. We applied our algorithm to all the matrices. For every tested number of negative eigenvalues $50i$ for $i = 0...20$, this figure shows the number of third type reductions that were performed by the algorithm on each of the 10 matrices. Clearly, when ϕ is getting closer to 1, the number of third type reductions increases.

is blocked for cache and uses level-3 BLAS routines, as well as a lower-level LAPACK routine, DGBTF2. This routine (DGBTF2) also factors a general banded matrix, but column-by-column, without blocking for cache.

Results on random matrices are shown in figures 6.1 and 6.7. When the half-bandwidth is 50, our algorithm outperforms both LAPACK routines, because cache blocking is not effective. When the half-bandwidth is 100, the cache-blocked DGBTRF outperforms our algorithm, which is not blocked, on balanced ϕ. Our algorithm still outperforms DGBTRF when ϕ is unbalanced, and it outperforms DGBTF2 for all ϕ. Results on three real-world test matrices, shifted to have different ϕ values, appear in figures 6.8, 6.9 and 6.10. On these matrices, our algorithm outperforms both DGBTRF and DGBTF2 even on bandwidths of about 100, and even though the comparison of DGBTRF and DGBTF2 shows that the former does exploit the cache.

6.6. **Performance as a function of bandwidth.** The performance of our algorithm on matrices with $\phi = 1/20$ is shown in figure 6.11. Many symmetric indefinite matrices in sparse-matrix collections[5, 6] are even more unbalanced, so this experiment can also be considered

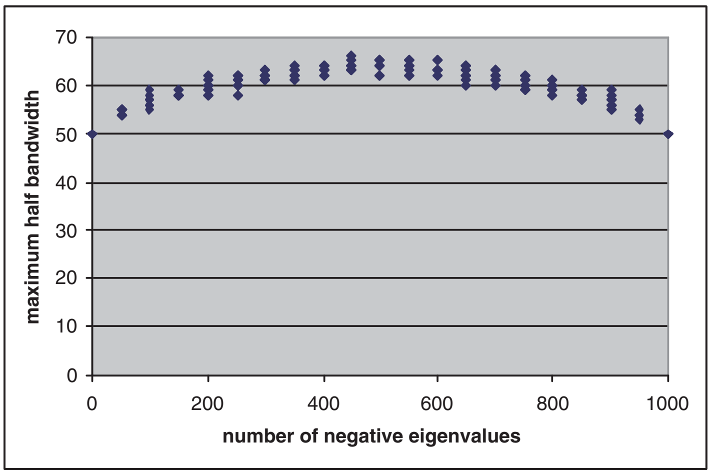

FIGURE 6.3. Maximal half bandwidth in the reduced matrices as a function of number of negative eigenvalues. Tests are the same as described in Figure 6.2.

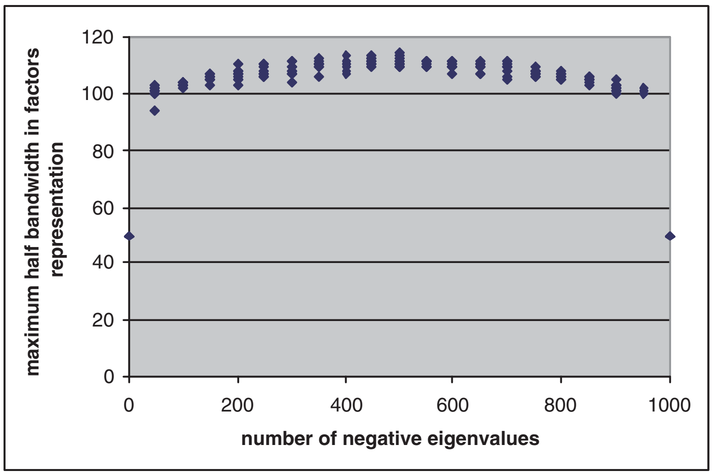

FIGURE 6.4. Half bandwidth in the representation of the factors, as a function of number of negative eigenvalues. Tests are the same as described in Figure 6.2.

to reflect real-world matrices. The matrices are random and were generated in the same way as described above.

6.7. **Growth during factorization of real-world matrices.** The growth factors for all the collection matrices are shown in figure 6.12. The growth for these matrices seems to be much lower then for the random matrices. Other tests we conducted were performed on the same matrices, shifted in a way that makes them highly ill conditioned. Namely, we shifted each matrix to one of its eigenvalues. The poor conditioning did not have any effect on the growth or on the number of third type reductions.

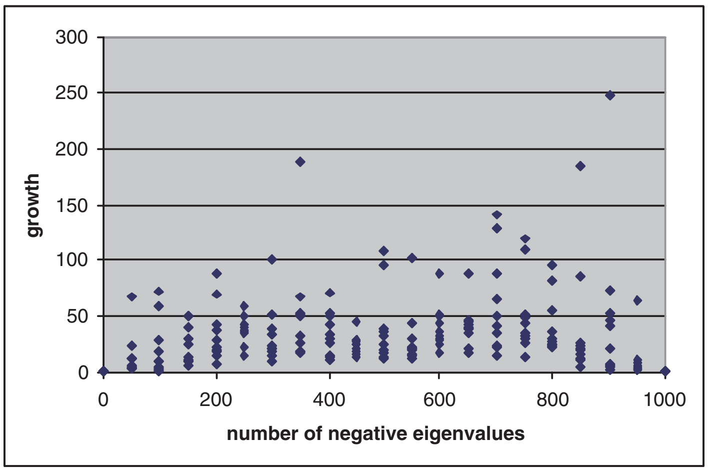

FIGURE 6.5. Growth as a function of number of negative eigenvalues. Tests are the same as described in Figure 6.2.

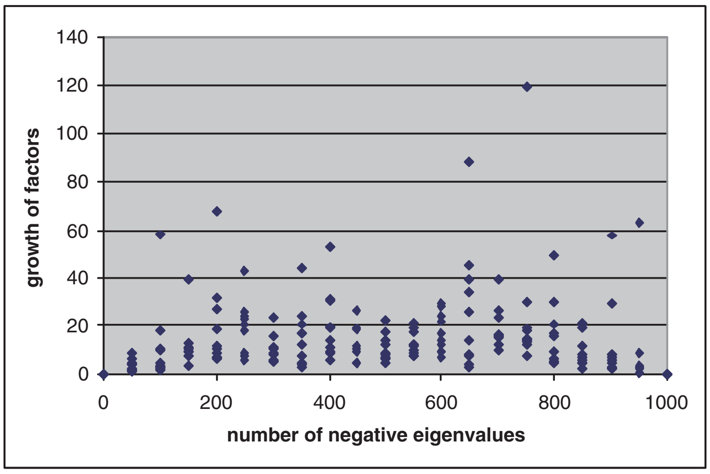

FIGURE 6.6. Growth in factors as a function of number of negative eigenvalues. Tests are the same as described in Figure 6.2.

6.8. **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 ϕ is not very small or very large, whereas our algorithm degrades smoothly and its performance is bounded for all ϕ.

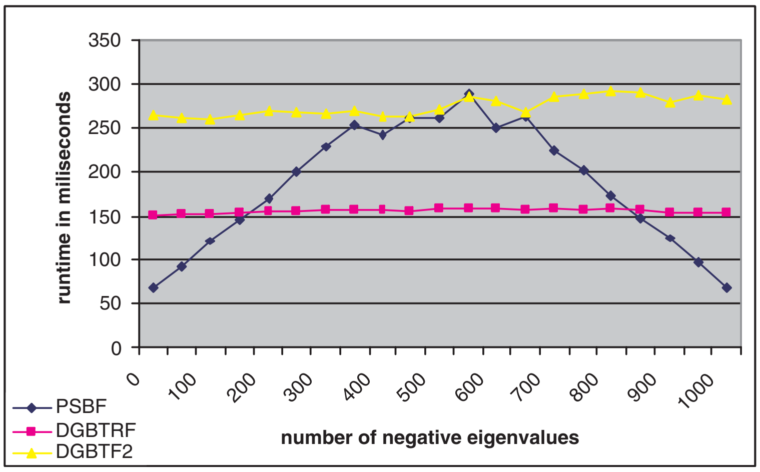

FIGURE 6.7. Comparison of runtime as a function of the number of negative eigenvalues. The algorithm is applied to 1000×1000 matrices with half bandwidth 100. The runtime is compared to GEPP LAPACK routines. The algorithm is faster then the non-blocked version of GEPP. It is slower then the blocked version when ϕ is balanced.

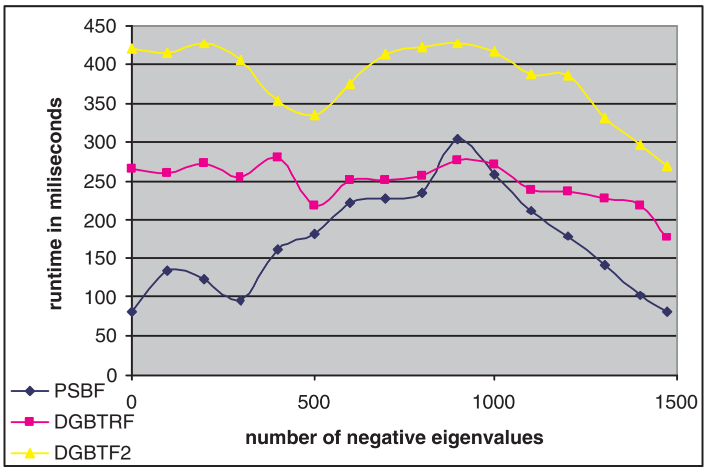

FIGURE 6.8. Comparison of runtime as a function of the number of negative eigenvalues. Test matrix is bcsstk12. Matrix was reordered using Cuthill-Mckee algorithm. Bandwidth after reordering is 105. Our algorithm outperforms both LAPACK versions.

7. CONCLUSIONS

We introduced a new algorithm for solving banded symmetric systems and gave both theoretic and practical justifications for using it. The proposed algorithm is the first to combine the advantages of running in $O(m^2n)$ time and $O(mn)$ memory while exploiting symmetry.

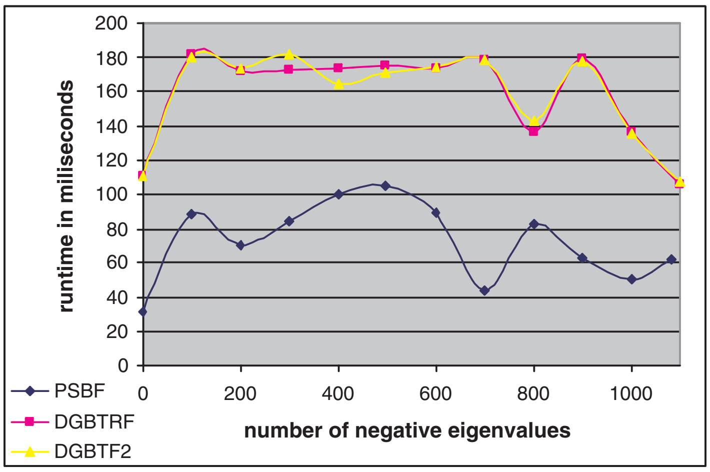

FIGURE 6.9. Comparison of runtime as a function of the number of negative eigenvalues. Test matrix is bcsstk9. Our algorithm outperforms both LAPACK versions.

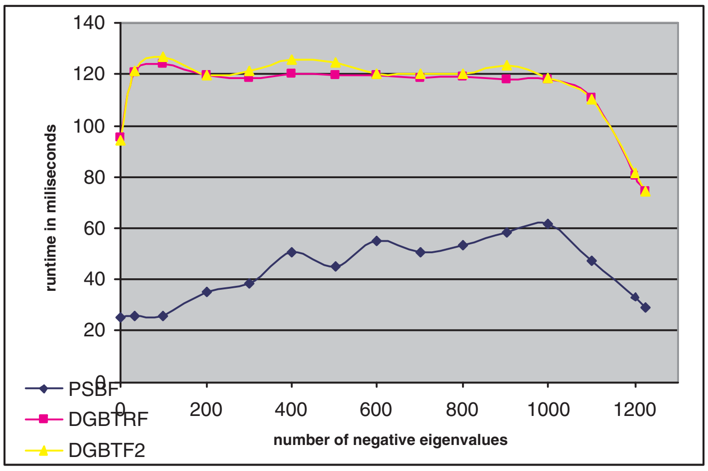

FIGURE 6.10. Comparison of runtime as a function of the number of negative eigenvalues. Test matrix is bcsstm27. Our algorithm outperforms both LAPACK routines.

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.

We also proved that the algorithm has an upper bound on its growth, a fact that suggests that the algorithm is backward stable.

The algorithm was implemented, tested and compared to GEPP. The tests included both artificial random banded matrices and some real-world matrices. The algorithm outperformed GEPP on most of the artificial tests and on all the real-world tests. The performance of the algorithm seems to be related to the number of positive eigenvalues to number of negative

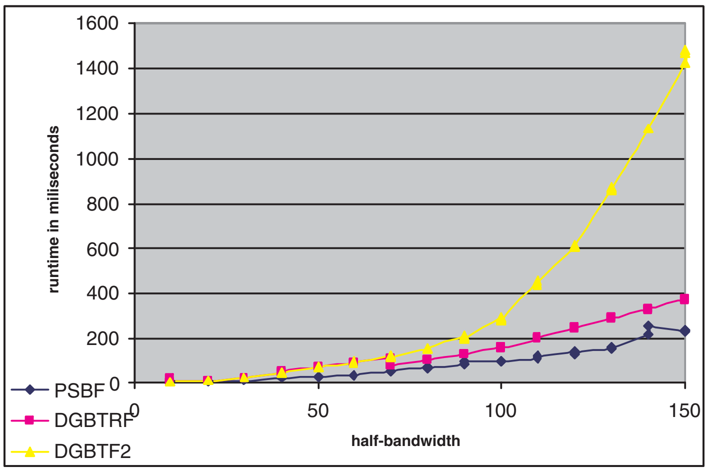

Figure 6.11. Comparison of runtime as a function of the bandwidth. Test matrix are random 1000×1000 with 50 negative eigenvalues. Our algorithm outperforms both LAPACK versions.

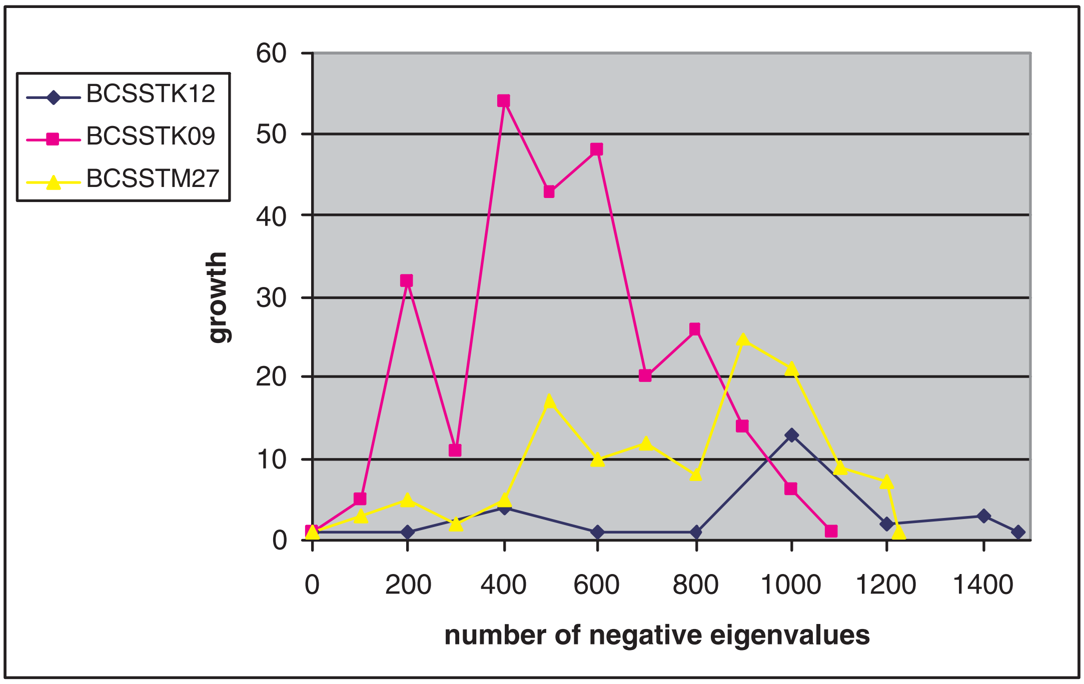

Figure 6.12. The growth for each one of the tests conducted on the collection matrices. Growth is much lower then for the random matrices.

eigenvalues ratio. 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.

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 growth 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. 11:233–242, 1971.

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

[3] Christian H. Bischof, Bruno Lang, and Xiaobai Sun. A framework for symmetric band reduction. *ACM Transactions on Mathematical Software*, 26(4):581–601, 2000.

[4] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. 31:162–179, 1977.

[5] T. Davis. University of florida sparse matrix collection. *http://www.cise.ufl.edu/research/sparse*, 2000.

[6] I. S. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Technical Report TR/PA/97/36, CERFACS, Toulouse, France, 1997. Also Technical Report RAL-TR-97-031 from Rutherford Appleton Laboratory and Technical Report ISSTECH-97-017 from Boeing Information & Support Services.

[7] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9(3):302–325, September 1983.

[8] Iain S. Duff and J. K. Reid. MA27 – A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Technical Report AERE R10533, AERE Harwell Laboratory.

[9] Iain S. Duff and John K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Technical Report RAL-95-001, Didcot, Oxon, UK, 1995.

[10] L. Fox, H. D. Huskey, and J. H. Wilkinson. Notes on the solution of algebraic linear simultaneous equations. 1:149–173, 1948.

[11] Nicholas J. Higham. How accurate is Gaussian elimination? In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1989, Proceedings of the 13th Dundee Conference*, volume 228, pages 137–154, Essex, UK, 1990. Longman Scientific and Technical.

[12] Nicholas J. Higham. Stability of aasen's method. *Manuscript*, 1997.

[13] Nicholas J. Higham. Stability of the diagonal pivoting method with partial pivoting. *J-SIMAX*, 18(1):52–65, Jan 1997.

[14] Mark T. Jones and Merrell L. Patrick. Bunch-kaufman factorization for real symmetric indefinite banded matrices. *SIAM J. Matrix Anal. Appl.*, 14:553–559, 1993.

[15] Bruno Lang. A parallel algorithm for reducing symmetric banded matrices to tridiagonal form. *SIAM J. Sci. Comput.*, 14(6):1320–1338, November 1993.

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

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

[18] H. Rutishauser. On jacobi rotation patterns. In *Proc. Symp. Appl. Math. 15 (Amer. Math. Soc.)*, pages 219–239, 1963.

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

[20] G.W. Stewart. The economical storage of plane rotations. *Numer. Math.*, 25(2):137–139, 1976.

[21] J. H. Wilkinson. Error analysis of direct methods of matrix inversion. *Journal of the ACM (JACM)*, 8(3):281–330, 1961.

[22] J. H. Wilkinson. *Rounding Errors in Algebraic Processes.* Notes on Applied Science, No. 32. 1963.

School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel
E-mail address: irony@tau.ac.il, stoledo@tau.ac.il