IMA Journal of Numerical Analysis (2009) **29**, 208−234
doi:10.1093/imanum/drn005
Advance Access publication on March 14, 2008

Numerically stable LDL^{T}-factorization of $\mathscr{F}$-type saddle point matrices

ARIE C. DE NIET
*Research Institute of Mathematics and Computing Science,
University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands,
and Witteveen+Bos, Consulting Engineers, van Twickelostraat 2,
7411 SC Deventer, The Netherlands*

AND

FRED W. WUBS[†]
*Research Institute of Mathematics and Computing Science,
University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands*

[Received on 7 December 2006; revised on 4 December 2007]

We present a new algorithm that constructs a fill-reducing ordering for a special class of saddle point matrices: the $\mathscr{F}$-matrices. This class contains the matrix occurring after discretization of the Stokes equation on a C-grid. The commonly used approach is to construct a fill-reducing ordering for the whole matrix followed by an adaptation of the ordering such that it becomes feasible. We propose to compute first a fill-reducing ordering for an extension of the definite submatrix. This ordering can be easily extended to an ordering for the whole matrix. In this manner, the construction of the ordering is straightforward and it can be computed efficiently. We show that much of the structure of the matrix is preserved during Gaussian elimination. For an $\mathscr{F}$-matrix, the preserved structure allows us to prove that any feasible ordering obtained in this way is numerically stable. The growth factor of this factorization is much smaller than the one for general indefinite matrices and is bounded by a number that depends linearly on the number of indefinite nodes. The algorithm allows for generalization to saddle point problems that are not of $\mathscr{F}$-type and are nonsymmetric, e.g. the incompressible Navier–Stokes equations (with Coriolis force) on a C-grid. Numerical results for $\mathscr{F}$-matrices show that the algorithm is able to produce a factorization with low fill.

Keywords: saddle point problem; indefinite matrix; $\mathscr{F}$-matrix; factorization; numerical stability; growth factor; C-grid; (Navier–)Stokes equations; electrical networks.

1. Introduction

In this paper, we study the direct solution of the equation

$$Kx = b, \tag{1.1}$$

where $K \in \mathbb{R}^{(n+m)\times(n+m)}$ $(n \geqslant m)$ is a saddle point matrix that has the form

$$K = \begin{pmatrix} A & B \\ B^{\mathrm{T}} & 0 \end{pmatrix}, \tag{1.2}$$

[†]Email: wubs@math.rug.nl

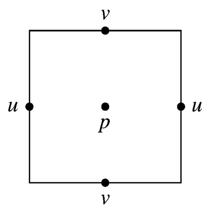

FIG. 1. Positioning of velocity (u, v) and pressure (p) variables in the C-grid.

with $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$. In this paper, we consider only symmetric positive definite A. This makes the matrix K itself symmetric indefinite. Although we assume A to be symmetric, many of the results in this paper can be easily generalized for nonsymmetric matrices A.

A survey of the occurrence of saddle point problems and their numerical solution can be found in Benzi *et al.* (2005). In many cases, saddle point problems can be solved efficiently via a Krylov subspace iteration (van der Vorst, 2003) combined with appropriate preconditioning (Benzi *et al.*, 2005; de Niet & Wubs, 2007; Elman *et al.*, 2002; Kay *et al.*, 2002). Nevertheless, in this paper we will focus on the direct solution of saddle point problems that occur in computational fluid dynamics. If the problem is 2D, direct solvers can compete in many cases with iterative methods. Also in general direct methods are more robust than iterative methods. The disadvantage is that they often require more memory.

In Duff *et al.* (1986) and Meurant (1999), one can find extended introductions to the field of sparse matrix factorizations. The basics are Gaussian elimination, matrix graphs, elimination trees, fill-reducing orderings, etc. We will introduce these notions only briefly where we need them.

1.1 $\mathscr{F}$*-matrices*

In a large part of this paper, we will pay attention to a special class of saddle point matrices, namely, the $\mathscr{F}$-matrices. We start off by defining the gradient matrix which is used to specify the $\mathscr{F}$-matrix.

DEFINITION 1.1 A gradient matrix has at most two entries per row. Moreover, if there are two entries, their sum is zero.

We have chosen the name 'gradient matrix' because this type of matrix typically results from the discretization of a pressure gradient in flow equations. It is important to note that the definition allows a gradient matrix to be nonsquare. Now, we can define the $\mathscr{F}$-matrices.

DEFINITION 1.2 An $\mathscr{F}$-matrix is a saddle point matrix (1.2) with A symmetric positive definite and B a gradient matrix.

The definition is originally due to Tůma (2002). $\mathscr{F}$-matrices occur in various fluid flow problems where Arakawa A-grids (collocated) or C-grids (staggered, see Fig. 1) are used. For example, in Arioli & Manzini (2003) the discretization of Darcy's equation in groundwater flow results in an $\mathscr{F}$-matrix. They occur as well in electrical networks (Vavasis, 1994).

In the example below, we introduce the $\mathscr{F}$-matrix that will be used throughout the paper to illustrate the theory.

EXAMPLE 1.3 Let K be a saddle point matrix as defined in (1.1) where the matrices A and B are, respectively,

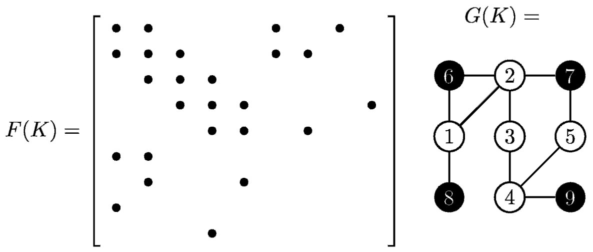

FIG. 2. The sparsity pattern $F(K)$ (left) and the adjacency graph $G(K)$ (right) of the matrix K as given in Example 1.3. In the adjacency graph, the numbers of the nodes correspond to the rows of $F(K)$. The P-nodes are coloured black; V-nodes are white.

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{1.3}$$

So K is a 9×9 symmetric saddle point matrix.

1.2 *Factorization of sparse indefinite matrices*

We define the ‘adjacency graph’ of the matrix K as follows:

$$G(K) = \{V \cup P, E\},$$

where the nodes or vertices are a union of two disjoint sets

$$V = \{i \mid K_{ii} \neq 0\} \quad \text{and} \quad P = \{i \mid K_{ii} = 0\}$$

and the edges are given by the off-diagonal nonzeros in K

$$E = \{\{i, j\} \mid i \neq j, K_{ij} = K_{ji} \neq 0\}.$$

The graph is undirected, so the edges $\{i, j\}$ and $\{j, i\}$ are considered to be the same. Note that our definition of an adjacency graph differs from the commonly used definition. We distinguish the vertices with zero diagonal entry in K from the ones with nonzero diagonal entry. From the definition, it follows immediately that the set V contains precisely the nodes that originate from the submatrix A. In fluid problems, these are the velocity nodes. The set P contains all the nodes that originate from the empty $(2, 2)$ block in K. In fluid terms, these are the pressure nodes. The nodes in V and P will be called V-nodes and P-nodes, respectively.

In Fig. 2, one finds the sparsity pattern and the adjacency graph of the matrix in Example 1.3. For this example, $V = \{1, 2, 3, 4, 5\}$ and $P = \{6, 7, 8, 9\}$. The P-nodes are coloured black in the adjacency graph.

The aim of this paper is to find a fill-reducing pre-ordering Q such that we can perform Gaussian elimination on the permuted saddle point matrix QKQ^{T}, resulting in a factorization of the form

$$QKQ^{\mathrm{T}} = LDL^{\mathrm{T}}, \tag{1.4}$$

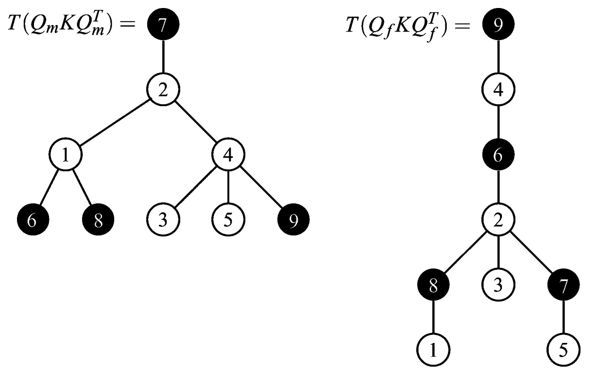

FIG. 3. Two elimination trees for the saddle point matrix in Example 1.3. The tree at the left belongs to the minimum degree ordering, the tree at the right to the ordering given by Algorithm 2.2.

where L is a unit lower triangular matrix and D is a block-diagonal matrix with blocks of size 1×1 or 2×2.

The 'elimination tree' of the reordered matrix QKQ^{T} is constructed in the following way. Each vertex is assigned precisely one parent

$$\mathrm{PARENT}[j] = \min_i \{i > j \mid l_{ij} \neq 0\},$$

where l_{ij} is the entry on the ith row and jth column of the lower triangular matrix L of (1.4). So to find the parent of j, we search the first nonzero entry below the diagonal in the jth column of L. One gets the elimination tree by drawing an edge between each vertex and its parent. For an example, see Fig. 3. Both the elimination tree and the adjacency graph play an important role in sparse matrix factorization. Operations on the matrix like reorderings and Gaussian elimination can be translated into operations on the adjacency graph and the elimination tree (see Liu, 1990).

The search for an appropriate ordering Q is far from trivial for large sparse matrices in general and especially in the case of symmetric indefinite saddle point matrices. There are three major issues that we have to address about the factorization (1.4):

(1) 'factorizability': given an ordering Q, does the factorization exist?
(2) 'sparsity': can we construct Q such that it reduces the fill in the L-factor?
(3) 'numerical stability': given an ordering Q and the corresponding LDL^{T}-factorization, can we prove that the errors in $x = L^{-\mathrm{T}}D^{-1}L^{-1}b$ are bounded?

We start with (1), the factorizability. Positive definite matrices like our submatrix A are 'strongly factorizable', i.e. all possible orderings give a factorizable matrix. This does not hold for saddle point matrices. We illustrate this with an example.

EXAMPLE 1.4 In Example 1.3, we can choose the Q such that it is the reordering matrix corresponding to the permutation $q = \{6, 7, 8, 9, 1, 2, 3, 4, 5\}$. This would give the following permuted saddle point matrix:

$$\begin{bmatrix} 0 & B^{\mathrm{T}} \\ B & A \end{bmatrix},$$

which is not factorizable, because if we start performing Gaussian elimination, we get in the first step a zero pivot. The minimum degree ordering for the matrix (computed with MATLAB) is $q_m = \{6, 8, 1, 3, 5, 9, 4, 2, 7\}$. This ordering has the same problem, as it wants to eliminate the black nodes 6 and 8 first. In fact, any ordering that starts with a P-node will suffer from this problem. In general, a node in P cannot be eliminated before one of its neighbours (which are all in V because the $(2, 2)$ block is zero) is eliminated. Or, equivalently, as formulated in Tůma (2002), a necessary condition for factorizability is that the elimination tree of the permuted matrix has no leaves in P. For our model problem, let Q_m be the permutation matrix corresponding to the permutation q_m; then the elimination tree for the matrix $Q_m K Q_m^{\mathrm{T}}$ has three leaves in P as is shown in Fig. 3 (left), so the reordered saddle point matrix is not factorizable.

The example shows that the ordering for saddle point matrices has to be chosen carefully. An ordering that gives a factorizable permuted matrix is called ‘feasible’.

Most of the literature is about the factorization of sparse symmetric indefinite matrices in general, e.g. Duff *et al.* (1979, 1991), which is a much larger class than the saddle point problems that we consider. To produce a feasible ordering, algorithms for factorization of indefinite matrices often use the pivoting strategies of Bunch–Parlett or Bunch–Kaufman (Bunch & Parlett, 1971, Bunch & Kaufman, 1977, and Ashcraft *et al.*, 1999). These strategies are applied in the factorization phase and basically do the following: if a zero pivot is encountered, a search is started for a second node that is coupled to the first such that the two together form a stable and invertible 2×2 matrix. In the case of our example, the nodes $\{6, 2\}$ would form an acceptable pivot because

$$\begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$$

is invertible. Because of the Bunch–Parlett and Bunch–Kaufman pivoting strategies, the matrix D in (1.4) is allowed to have 2×2 blocks on the diagonal.

Recently, these pivoting strategies were fruitfully combined with a weighted matchings search algorithm in the package PARDISO (Röllin & Schenk, 2005; Hagemann & Schenk, 2006; Schenk & Gärtner, 2006). Similar techniques were used in the CMP-algorithm as described in Duff & Pralet (2007) and Pralet (2004). Both algorithms take the following steps: (i) group V- and P-nodes beforehand in 2×2 blocks by a weighted matchings algorithm, (ii) treat the 2×2 blocks as a supernode and construct the compressed graph, (iii) compute an ordering for the compressed graph, and (iv) expand the ordering to the original graph. Since these two algorithms are most competitive for sparse symmetric indefinite systems, we will compare the algorithm that we propose in this paper to PARDISO.

The second important issue is (2), i.e. the sparsity of the factors. To reduce both memory requirements and construction time, we would like the factor L to be as sparse as possible. The two most important algorithms to compute a fill-reducing ordering for a matrix are approximate minimum degree (Amestoy *et al.*, 1996) and nested dissection (George, 1973). Unfortunately, they apply only to matrices that have no zeros on the diagonal because they are based on the adjacency graph of the matrix, which assumes a positive diagonal entry. The algorithms do not see the difference between white (V) and black (P) nodes in Fig. 2. In our example, the approximate minimum degree ordering gives a non-feasible ordering because the black node 6 (see ordering q_m above) is selected first in the factorization phase.

If the approximate minimum degree algorithm is applied to a general saddle point matrix K, it is unlikely that the computed ordering is feasible. So the ordering has to be repaired either during elimination, by delaying the elimination of P-nodes, or by adapting the ordering beforehand. For repair during elimination, we have to check all pivots, which gives some increase in the work required during the factorization phase. In Tůma (2002), the ordering is repaired before elimination. After the computation of a fill-reducing ordering for K, the ordering is adapted on the basis of, elimination tree operations only. For $\mathscr{F}$-matrices, it can be proven that the final ordering is feasible. However, a delay in the elimination of indefinite nodes either during elimination or beforehand creates extra fill in the factor L. Especially, if we deal with saddle point problems where B^{T} has fewer entries per row than A—which is often the case for $\mathscr{F}$-matrices—a fill-reducing ordering like (approximate) minimum degree will choose the nodes in P to be eliminated first because they have the lowest degree. In Tůma (2002), it was observed that 80% of the leaves of the elimination tree belong to the set P. For all these leaves, elimination had to be delayed.

There is one more alternative, which is called the Schur complement approach. The elimination of the nodes from P is delayed until all V-nodes have been eliminated. Unfortunately, in many cases the Schur complement $-B^{\mathrm{T}}A^{-1}B$ is completely full. So this approach is not very practical and we will not give further attention to this approach.

Instead of eliminating the V-nodes first, one could also choose to eliminate all P-nodes, together with an equal number of nodes from V. Since the choice of the requisite V-nodes is not unique, the crux here is to find good combinations. The interesting point is that the Schur complement of this elimination will be sparse if we can split B^{T} into $[B_1^{\mathrm{T}}, B_2^{\mathrm{T}}]$ such that the inverse of B_1 is sparse. A different kind of view on this approach is that a basis for the null space of B^{T} is given by the columns of $[-B_2B_1^{-1}, I]^{\mathrm{T}}$, which are sparse. Hence, if the equation associated with the lower part of (1.2) reads $B^{\mathrm{T}}v = 0$, then the solution for the velocities is just a linear combination of the basis of the null space. By substitution in the equation for the upper part of (1.2) and just testing (premultiplying) with the null space, we find exactly the same sparse Schur complement system as before. Therefore, the approach is also called the null-space method. The problem is to find a good splitting of B such that a sparse representation of the kernel is possible. This is called the nice basis problem and is usually solved using graph theory; for more details see Arioli & Manzini (2003), Pinar *et al.* (2006) or Benzi *et al.* (2005). A related approach in electromagnetics is the tree/co-tree decomposition; which is used to transform an underdetermined problem into a uniquely solvable one (Webb, 1993).

The last issue is (3), i.e. numerical stability. The growth factor

$$\rho = \frac{\max_{i,j,l} |k_{ij}^{(l)}|}{\max_{i,j} |k_{ij}|} \tag{1.5}$$

is an important measure for stability. The growth factor is the largest entry that occurs in the Schur complements during Gaussian elimination divided by the largest entry in the matrix K. This growth factor can become very large, even if we have a feasible ordering for K. If we consider Bunch–Kaufman or Bunch–Parlett pivoting, the stability bound for the growth factor is very weak: $\rho \leqslant 2.57^{(n+m-1)}$ (Higham, 2002, Section 11.1). Although in many applications no problems were reported with numerical stability, there is a lack of better bounds for ρ.

Next to bounding the growth factor, it is important to bound the size of the entries in the matrix L because the error in the solution $x = L^{-\mathrm{T}}D^{-1}L^{-1}b$ is clearly related to the size of the entries in L (see, e.g. Ashcraft *et al.*, 1999). Finally, we note that the error in the solution is also determined by the condition number of K, which is a given fact as long as K is not modified.

One of the important results in this paper is that for $\mathscr{F}$-matrices we are able to give much better bounds for the growth factor and for the size of the entries in L.

In this paper, we will not pay much attention to the actual construction of the LDL^{T}-factorization. We will focus on the construction of a feasible ordering, numerical stability and the sparsity of the computed factors. Given a feasible, numerically stable ordering for K, there are several efficient codes available for construction of the corresponding LDL^{T}-factorization. We mention MA47 (Duff *et al.*, 1991) as an example.

The outline of the paper is as follows. In Section 2, we sketch the algorithm to compute a fill-reducing ordering for a saddle point matrix. In Section 3, we show properties that remain invariant under Gaussian elimination with this ordering. In Section 4, we give a proof for numerical stability of Gaussian elimination for $\mathscr{F}$-matrices using this ordering. Fast construction of the ordering is treated in Section 5. In Section 6, we show the numerical results for a Stokes equation in a driven cavity and for a set of $\mathscr{F}$-matrices that is used in Tůma (2002). We end with a discussion in Section 7.

2. Sketch of the algorithm

As we mentioned in Section 1.2, many of the current algorithms have in common that they compute a fill-reducing ordering for K and then somehow adapt it to make it feasible. The delay in elimination will give an increase of the fill in the factors. Unfortunately, the increase is hard to predict, but it certainly implies an increase in the construction time of the factors. To overcome this inefficiency, we propose a different approach. The idea is to compute an ordering for the velocity nodes V—based on a graph that contains information of the whole matrix—and then to insert the pressure nodes P appropriately. Assume that we have an elimination order on V; then we use the following simple rule to insert elements of P in the ordering for V.

RULE 2.1 If during Gaussian elimination with K the node $v \in V$ is to be eliminated and it is connected to a $p \in P$ then v and p are eliminated together using a 2×2 pivot.

Note that with this rule we get as many 2×2 pivots as there are nodes in P. Only if a node $v \in V$ becomes totally disconnected from P due to the elimination of previous nodes, can it be eliminated singly.

Because all P-nodes are eliminated together with a V-node in pivots of the form

$$\begin{pmatrix} \alpha & \beta \\ \beta & 0 \end{pmatrix},$$

there is no doubt about factorizability. We immediately get a feasible ordering, so we do not need any additional repairs.

If we apply this rule to an ordering of V that is constructed as a fill-reducing ordering for A, the resulting ordering for K will not be fill-reducing in general. To ensure that the final ordering *is* fill reducing, we have to use information about the whole matrix, so somehow the sparsity patterns of B and B^{T} have to be included. This is the case if the ordering for V is fill-reducing for the sparsity pattern $F(A) \cup F(BB^{\mathrm{T}})$, where $F(A)$ denotes the sparsity pattern of A. This matrix is an envelope for the fill that will be created by the elimination of the nodes in P. In many cases, this will be equal to $F(A + BB^{\mathrm{T}})$, but to avoid possible cancellation in the addition we will use the matrix $F(A) \cup F(BB^{\mathrm{T}})$. Summarizing, we get the following algorithm.

ALGORITHM 2.2 To compute a feasible fill-reducing ordering for the saddle point matrix K:

1. Compute a fill-reducing ordering for the V-nodes based on $F(A) \cup F(BB^{\mathrm{T}})$.
2. Insert the P-nodes into the ordering according to Rule 2.1.

The P-nodes (Step 2) can be inserted during Gaussian elimination, which means that we have to adapt the algorithm for Gaussian elimination. However, in case of $\mathscr{F}$-matrices, this can be done symbolically before elimination. A very efficient algorithm for that is described in Section 5.

$$F' = \begin{bmatrix} \bullet & \bullet & & & \\ \bullet & \bullet & \bullet & & \bullet \\ & \bullet & \bullet & \bullet & \\ & & \bullet & \bullet & \bullet \\ & \bullet & & \bullet & \bullet \end{bmatrix} \qquad G(F') =$$

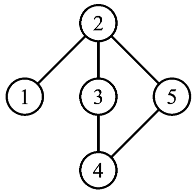

FIG. 4. The sparsity pattern $F' = F(A) \cup F(BB^{\mathrm{T}})$ (left) and the corresponding adjacency graph (right) that is used to compute an ordering for the V-nodes of Example 1.3.

EXAMPLE 2.3 To clarify the algorithm, we apply it to the matrix in Example 1.3. In the first step, we will compute an ordering for the V-nodes that is based on the sparsity pattern $F(A) \cup F(BB^{\mathrm{T}})$. This sparsity pattern and the adjacency graph can be found in Fig. 4. The minimum degree ordering for this matrix is $q_v = \{1, 3, 5, 2, 4\}$. The second step is the insertion of the P-nodes in this ordering. The first node in the ordering ($q_v(1) = 1$) is connected to the P-nodes 6 and 8 (see Fig. 2), so according to Rule 2.1 we have to choose one of these two nodes to eliminate together with node 1. For now this choice is arbitrary, but in Section 5 we will argue that we had better choose node 8 because it has the least degree (number of neighbours) of the two. If we remove nodes 1 and 8 from the graph and continue the elimination and insertion of P-nodes, we finally get the ordering $q_f = \{1, 8, 3, 5, 7, 2, 6, 4, 9\}$. With this ordering, the permuted saddle point matrix becomes (we leave out the zeros)

$$Q_f K Q_f^{\mathrm{T}} = \begin{bmatrix} 2 & 1 & & & & -1 & -1 & & \\ 1 & & & & & & & & \\ & & 2 & & & -1 & & -1 & \\ & & & 2 & 1 & & & -1 & \\ & & & 1 & & -1 & & & \\ -1 & & -1 & & -1 & 2 & 1 & & \\ -1 & & & & & 1 & & & \\ & & -1 & -1 & & & & 2 & -1 \\ & & & & & & & -1 & \end{bmatrix} \tag{2.1}$$

and its factors are

$$L = \begin{bmatrix} 1 & & & & & & & & \\ \frac{1}{2} & 1 & & & & & & & \\ & & 1 & & & & & & \\ & & & 1 & & & & & \\ & & & \frac{1}{2} & 1 & & & & \\ -\frac{1}{2} & -1 & -\frac{1}{2} & & 2 & 1 & & & \\ -\frac{1}{2} & -1 & & & & \frac{2}{7} & 1 & & \\ & & -\frac{1}{2} & -\frac{1}{2} & -1 & -\frac{3}{7} & -\frac{3}{2} & 1 & \\ & & & & & & & 1 & 1 \end{bmatrix} \quad \text{and}$$

$$D = \begin{bmatrix} 2 &&&&&&&& \\ & -\frac{1}{2} &&&&&&& \\ && 2 &&&&&& \\ &&& 2 &&&&& \\ &&&& -\frac{1}{2} &&&& \\ &&&&& \frac{7}{2} &&& \\ &&&&&& -\frac{2}{7} && \\ &&&&&&& \frac{3}{2} & \\ &&&&&&&& -\frac{2}{3} \end{bmatrix}.$$

We chose here to build the factorization such that D is a diagonal matrix with no 2×2 blocks on the diagonal. Consequently, D has negative entries on the diagonal. This factorization allows us to construct the elimination tree that is based on the L-factor. The tree is depicted in Fig. 3 (right).

REMARK 2.4 The motivation for this approach can be understood from a nested dissection point of view. In our type of applications, often B^{T} is a discrete divergence and B a discrete gradient operator. The rows of B^{T} then represent the continuity equation in each cell of the grid. If we deal with a C-type staggered grid (e.g. the pressure nodes are in cell centres, the velocity nodes on cell faces; see Fig. 1), it means that all velocity nodes occur in the continuity equation of two different cells. To compute the pressure gradient in a velocity node, we take the difference of the pressure values in the two neighbouring cells. The elimination of a velocity node forces the two neighbouring cells to merge. We need only one pressure node to fix the pressure in a cell, so we can eliminate one of the two, together with the velocity node. Or, equivalently, we can eliminate one of the two continuity equations because the pressure nodes in the two cells get connected by the elimination of the velocity node. In this view, we can consider the velocity nodes as separators of the pressure nodes, which need to be eliminated as soon as possible. That is precisely what is formulated in Rule 2.1 and is used in Algorithm 2.2.

REMARK 2.5 We make one last comment to clarify the relation with other approaches. The adjacency graph of $F(A) \cup F(BB^{\mathrm{T}})$ can be viewed as the compressed graph of a supernode that consists of both a node in V and one in P, where for the latter we leave open which of the ps, that are connected to v, will be taken. In the expanded elimination tree, we have to choose and the choice is made by Rule 2.1. It is advantageous to delay the choice since in the proof of Theorem 3.4 we will see that couplings to a P-node can be lost due to cancellation (see also Remark 3.5). During the insertion of the P-nodes, described in Section 5, we can track cancellation and hence no repairs need to be made during the actual factorization.

In the rest of the paper, $K^{(l)}$ denotes the matrix K after l steps of Gaussian elimination according to an ordering defined by the algorithm above. If this is not necessary we will not make a distinction between $K^{(l)}$ or any symmetric permutation of the matrix.

3. Properties invariant under Gaussian elimination

In this section, we will show that Algorithm 2.2 gives an ordering with the nice property that during elimination much of the structure of the original matrix is preserved. If we perform Gaussian elimination on the matrix K, we get a sequence of Schur complements $K^{(l)}$ for $l = 1, \ldots, n$. If we separate the V- and P-nodes, the structure of all these matrices is

$$K^{(l)} = \begin{pmatrix} A^{(l)} & B^{(l)} \\ B^{(l),\mathrm{T}} & C^{(l)} \end{pmatrix}.$$

The first important property is that Gaussian elimination subject to Rule 2.1 on a saddle point matrix (1.2) with $C = 0$ always gives a saddle point matrix with the very same structure as Schur complement. In other words, the nodes in P are not coupled in K and will never become coupled in $K^{(l)}$.

THEOREM 3.1 If $C^{(0)} = 0$ then for all l we have $C^{(l)} = 0$.

Proof. The proof is by induction. By assumption, we have $C^{(0)} = 0$. Now, suppose that $C^{(l)} = 0$. Let us compute $K^{(l+1)}$ by eliminating the first row in $K^{(l)}$. Now, we have to distinguish two cases that can occur.
Case 1. The first possibility is that the lth node is not connected to a node in P. We can make this more explicit by splitting $A^{(l)}$ and $B^{(l)}$:

$$K^{(l)} = \begin{pmatrix} \alpha & a & 0 \\ a^{\mathrm{T}} & \hat{A} & \hat{B} \\ 0 & \hat{B}^{\mathrm{T}} & 0 \end{pmatrix}.$$

According to Rule 2.1, this node can be eliminated singly. The Schur complement of α is equal to $K^{(l+1)}$ and easy to compute:

$$K^{(l+1)} = \begin{pmatrix} A^{(l+1)} & B^{(l+1)} \\ B^{(l+1),\mathrm{T}} & 0 \end{pmatrix} \tag{3.1}$$

with

$$A^{(l+1)} = \hat{A} - aa^{\mathrm{T}}/\alpha \tag{3.2}$$

and

$$B^{(l+1)} = \hat{B}. \tag{3.3}$$

Clearly, $C^{(l+1)} = 0$.
Case 2. The other possibility is that the lth node *is* connected to a node in P. Then, the structure is

$$K^{(l)} = \left(\begin{array}{cc|cc} \alpha & \beta & a & b \\ \beta & 0 & \hat{b} & 0 \\ \hline a^{\mathrm{T}} & \hat{b}^{\mathrm{T}} & \hat{A} & \hat{B} \\ b^{\mathrm{T}} & 0 & \hat{B}^{\mathrm{T}} & 0 \end{array}\right).$$

To satisfy Rule 2.1, the first two rows are eliminated together, using a 2×2 pivot. Elimination gives again a matrix with structure (3.1), but now we have

$$A^{(l+1)} = \hat{A} - a^{\mathrm{T}}(\hat{b}/\beta) - (\hat{b}/\beta)^{\mathrm{T}}a + \alpha(\hat{b}/\beta)^{\mathrm{T}}(\hat{b}/\beta) \tag{3.4}$$

and

$$B^{(l+1)} = \hat{B} - (b/\beta)^{\mathrm{T}}\hat{b}. \tag{3.5}$$

Also, in this case, we have $C^{(l+1)} = 0$. □

In the proof of the theorem, we see that in both cases $B^{(l+1)}$ is determined by the elements of $B^{(l)}$ only. No elements of A are involved.

COROLLARY 3.2 For all l the matrix $B^{(l)}$ is independent of the sparsity pattern and the size of the entries in A. It depends only on the ordering of the V-nodes.

An immediate consequence of this corollary is that we can compute the sequence of $B^{(l)}$s without using the size of the entries of A. We can exploit this during the insertion of the P-nodes into the ordering of the V-nodes in Section 5. Note that on the other hand the fill structure and the size of the entries of $A^{(l+1)}$ do depend on B, as is clear from (3.4).

The following theorem is important for stability.

THEOREM 3.3 If A is symmetric positive definite, $A^{(l)}$ is symmetric positive definite for all l.

Proof. This is easy to prove by induction. We have $A^{(0)} = A$ is symmetric positive definite by assumption. Now, assume that $A^{(l)}$ is symmetric positive definite. Elimination of the next node will give us $K^{(l+1)}$ and $A^{(l+1)}$. As in the proof of Theorem 3.1, we have to distinguish two cases.
Case 1. If the lth node is not coupled to a node in P, $A^{(l+1)}$ is given by (3.2). It is the Schur complement of a symmetric positive definite matrix; which is again symmetric positive definite.
Case 2. If the lth node is coupled to a node in P, $A^{(l+1)}$ is given by (3.4). We can rewrite it as

$$A^{(l+1)} = \hat{A} - aa^{\mathrm{T}}/\alpha + \alpha(a/\alpha - \hat{b}/\beta)^{\mathrm{T}}(a/\alpha - \hat{b}/\beta). \tag{3.6}$$

It is the Schur complement of a positive definite matrix $(\hat{A} - aa^{\mathrm{T}}/\alpha)$ plus the term $\alpha(a/\alpha - \hat{b}/\beta)^{\mathrm{T}}(a/\alpha - \hat{b}/\beta)$, which is obviously symmetric and positive definite if $\alpha > 0$. Because $A^{(l)}$ is symmetric positive definite, all its diagonal elements are positive, so we indeed have $\alpha > 0$. □

THEOREM 3.4 If K is an $\mathscr{F}$-matrix, all $K^{(l)}$ are $\mathscr{F}$-matrices.

Proof. By assumption, we know that $K^{(0)} = K$ is an $\mathscr{F}$-matrix. According to Definition 1.2, it holds that $C^{(0)} = 0$, $A^{(0)}$ is symmetric positive definite and $B^{(0)}$ is a gradient matrix. Theorems 3.1 and 3.3, respectively, ensure that $C^{(l)} = 0$ and $A^{(l)}$ is symmetric positive definite, so we have only to prove that $B^{(l)}$ is a gradient matrix.

Once more we use induction. So let us assume that $B^{(l)}$ is a gradient matrix. Again, we distinguish the two cases that occur during Gaussian elimination.
Case 1. If the lth node is not coupled to a pressure node, $B^{(l+1)}$ is given by (3.3). The only difference with $B^{(l)}$ is the omission of the first (empty) row. So properties like the number of entries per row and row sum are certainly preserved and $B^{(l+1)}$ is a gradient matrix.
Case 2. If the lth node is coupled to a pressure node, $B^{(l+1)}$ is given by (3.5). The gradient matrix $B^{(l)}$ has at most two entries per row; when a row has precisely two entries, they have zero sum. The first row of $B^{(l)}$ is $(\beta \ \ b)$. Because β is nonzero, the row vector b has either one entry with value $-\beta$ or no entries at all. Hence, $-b/\beta$ is either a vector with one entry with value 1 or a zero vector. In (3.5), $\hat{b}^{\mathrm{T}}$ (i.e. the first column of $B^{(l)}$) is multiplied with this vector and added to $\hat{B}$. This leads to the following simple procedure to construct $B^{(l+1)}$. We get $B^{(l+1)}$ from $B^{(l)}$ by removing its first row and first column ($\hat{b}^{\mathrm{T}}$)—which gives $\hat{B}$—and add $\hat{b}^{\mathrm{T}}$ to the column that had an entry on the first row (b). Obviously, the rows with no entry in the first column do not change at all, so the number of entries and the row sum are preserved. Of interest are the rows that have an entry in $\hat{b}^{\mathrm{T}}$. Now, there are two possibilities. If $b = 0$, the only change is that the number of entries on the row decreases by one and the row sum does not matter anymore because there is either one or no entry left. In that case, $B^{(l+1)}$ is a gradient matrix. Otherwise (if b has precisely one entry), the row sum is preserved because we multiply the first

column by 1 and add it to one of the other columns of the matrix $\hat{B}$. So now $B^{(l+1)}$ is a gradient matrix as well. □

REMARK 3.5 Exact cancellation can occur in $B^{(l+1)}$ during Gaussian elimination. Fortunately, we know precisely when this happens, namely, if there is a row (not the first one) in $B^{(l)}$ that is a multiple of the first row. As shown in the last proof, the entries on that row are summed up in $B^{(l+1)}$ and cancel each other out because their sum is zero. It is advantageous to know when cancellation happens because this allows us to insert the nodes beforehand in the ordering in the symbolic factoring phase, as we will see in Section 5.

REMARK 3.6 For $B^{(l)}$, the size of the entries does not change. Each row in $B^{(l)}$ can be traced back to a row in $B = B^{(0)}$. If the row in $B^{(l)}$ is not empty, the size of the entries (at most two) on both rows is exactly the same. This is an immediate consequence of the proof of the last theorem. Entries on a row can move and possibly coincide, in which case they annihilate, but the size never changes. So we have $\max_{ij} |b_{ij}^{(l)}| \leqslant \max_{ij} |b_{ij}^{(0)}|$. This simplifies the study of the numerical stability of the elimination.

Summarizing, we can say that with Rule 2.1 much of the structure of the original saddle point matrix is preserved during Gaussian elimination. In particular, all Schur complements of $\mathscr{F}$-matrices are again $\mathscr{F}$-matrices. This is advantageous for the study of the stability of the method, which we consider next.

4. Numerical stability

For the numerical stability of the factorization, we need to show that both the growth factor ρ defined in (1.5) and the elements of L are bounded (see Higham, 2002, Chapter 9; Duff *et al.*, 1986, Chapters 4 and 5; and Ashcraft *et al.*, 1999, for more details).

The pivoting strategies of Bunch–Parlett and Bunch–Kaufman guarantee the bound $\rho \leqslant (2.57)^{(n+m-1)}$. This bound is very weak because it grows exponentially with the problem size. Fortunately, in many cases, the growth factor stays far away from the bound, but it appears to be hard to find a substantially smaller upper bound. We found such a smaller bound for $\mathscr{F}$-matrices. In this section, we present a bound on ρ that depends only linearly on m, i.e. the number of P-nodes. A similar bound will be given for the elements of L.

In this section, we assume that all entries in B have a value in $\{-1, 0, 1\}$. This is not a restriction because if B does not satisfy this property it can be forced to do so by a simple row scaling. Note, however, that such a scaling influences the condition number of K, for better or worse.

Before we give the theorem, we introduce the number $\chi(A)$ that denotes the maximum number of entries per row in the matrix A. Moreover, we introduce a number for the largest entry in the matrix A, the formal definition being

$$\mu(A) = \max_{ij} |a_{ij}|.$$

This can be used to define the growth factor ρ_A of A during the elimination

$$\rho_A = \max_{l} \mu(A^{(l)})/\mu(A).$$

THEOREM 4.1 Let K be an $\mathscr{F}$-matrix with an ordering given by Algorithm 2.2 and let m be the number of columns in B; then for the LDL^{T}-factorization of K it holds that: (i) the growth factor is bounded by

$$\rho \leqslant \frac{\max(\rho_A \mu(A), 1)}{\max(\mu(A), 1)}, \quad \text{with } \rho_A \leqslant (2+m)\chi(A) - 1,$$

and (ii) the elements in L are bounded by

$$\mu(L) \leqslant \max([(1+m)\chi(A) - 1]\mu(A), 1).$$

Hence, the factorization is numerically stable.

The bound on the growth factor is much smaller than the general one for indefinite matrices. It grows, in the worst case that $\mu(A) \approx 1$, only linearly with m instead of exponentially.

REMARK 4.2 Another intriguing issue of the theorem is that in the case where $\max[\rho_A, (1+m)\chi(A)-1]$ $\mu(A) \leqslant 1$ both the growth factor of K and the elements of L are bounded by 1. Indeed, $\mu(A)$ can be made arbitrarily small by scaling K, but as mentioned before, this influences its condition number. However, if we do so the bounds tell us that the scaling should be more severe if the computational grid for the problem, in the case of a Stokes problem, is refined. We would rather like to have found bounds independent of the grid resolution. This is precisely what we saw for an example that is presented at the end of this section. However, we were not able to prove it. Hence, we conjecture that there is a class of problems, the Stokes problem, for which (if combined with an appropriate ordering) the bounds are independent of the grid resolution and hence a bounded scaling exists such that the growth factor of K is bounded by 1. This is similar to the Cholesky factorization of a symmetric positive definite matrix, of which the growth factor is also bounded by 1.

We give the proof of Theorem 4.1 later in this section, after introducing a few useful lemmas on gradient matrices. For these general lemmas, A and B are arbitrary, as are n, m and l.

LEMMA 4.3 Let G be an invertible gradient matrix with entries in $\{-1, 0, 1\}$. Then all entries of G^{-1} are in $\{-1, 0, 1\}$ as well.

Proof. The matrix G is a so-called totally unimodular matrix (see Theorem 6.27 from Cook *et al.*, 1997). A totally unimodular matrix is a matrix with elements in $\{-1, 0, 1\}$ for which the determinant of every square submatrix has also a value in $\{-1, 0, 1\}$. Hence, if the gradient matrix is nonsingular, then its determinant has magnitude 1. The inverse of a nonsingular totally unimodular, matrix is also totally unimodular, since one can express every subdeterminant of the inverse in a subdeterminant of the original, divided by the determinant of the whole matrix times a factor of unit magnitude (see Section 0.8.4 in Horn & Johnson, 1985). This completes the proof. □

LEMMA 4.4 Let G be an invertible gradient matrix with values in $\{-1, 0, 1\}$. There exist permutation matrices Q_1 and Q_2 and a diagonal matrix D with entries ± 1 such that $U = DQ_1GQ_2$ is a unit upper triangular gradient matrix. Moreover, the matrix U is an M-matrix.

Proof. The most important concern is to get it in upper triangular form. We will prove that by induction. First, we show that it is possible to bring a general nonsingular gradient matrix G of order n to the form

$$\begin{pmatrix} G' & u \\ 0 & d \end{pmatrix},$$

where d is a scalar. For that, note that G must contain at least one row with a single element, otherwise the sum of all columns would be zero and hence the matrix would be singular. By column and row permutations, this element can be brought to the last position, giving a matrix of the above form.

Now, assume we have the above form where d is a square upper triangular matrix of order k. Since the determinant of the whole matrix is the product of the determinant of d and that of G', the latter must be nonsingular. Hence, G' is also a nonsingular gradient matrix and we can repeat the process described above. After that, d is extended to an upper triangular matrix of order $k+1$ and G' is shrunk to a matrix of order $n-k-1$. Hence, by induction, we see that the matrix G can be brought to upper triangular form by permuting rows and columns.

This gives us an upper triangular matrix $\tilde{U} = Q_1 G Q_2$ with entries ± 1 on the diagonal. Choose D equal to the diagonal of $\tilde{U}$ and $U = D\tilde{U} = DQ_1GQ_2$ is a unit upper triangular gradient matrix.

According to Example 6.5b in Axelsson (1994), all upper and lower triangular matrices with positive diagonal elements and nonpositive off-diagonal elements are M-matrices; U is of this form. □

By definition, M-matrices have non-negative inverse; hence, together with Lemma 4.3 we immediately have the following.

COROLLARY 4.5 The elements of the inverse of U defined in Lemma 4.4 are in $\{0, 1\}$.

In the following lemmas, we derive some properties of μ.

LEMMA 4.6 Let $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{m\times l}$; then

$$\mu(AB) \leqslant \chi(A)\mu(A)\mu(B),$$
$$\mu(AB) \leqslant \chi(B^{\mathrm{T}})\mu(A)\mu(B),$$
$$\mu(AB) \leqslant m\mu(A)\mu(B).$$

Proof. The second bound follows from the first if we note that $\mu(AB) = \mu((AB)^{\mathrm{T}}) = \mu(B^{\mathrm{T}}A^{\mathrm{T}})$. Because $\chi(A) \leqslant m$ and $\chi(B^{\mathrm{T}}) \leqslant m$, the third bound is easily obtained as well, so we have only to prove the first bound. This follows from

$$\mu(AB) = \max_{ij}\left|\sum_{k=1}^{m} a_{ik}b_{kj}\right| \leqslant \max_{ij}\sum_{k=1}^{m}|a_{ik}||b_{kj}| \leqslant \mu(B)\max_{i}\sum_{k=1}^{m}|a_{ik}| \leqslant \mu(B)\chi(A)\mu(A). \tag{4.1}$$

□

If we deal with a diagonally dominant matrix, we have a sharper bound.

LEMMA 4.7 Let $A \in \mathbb{R}^{n\times m}$ be diagonally dominant and $B \in \mathbb{R}^{m\times l}$; then

$$\mu(AB) \leqslant 2\mu(A)\mu(B).$$

Proof. The matrix A is diagonally dominant, so $\sum_{k=1}^{m}|a_{ik}| \leqslant 2|a_{ii}| \leqslant 2\mu(A)$. If we use this in the last step in (4.1) in the proof of Lemma 4.6, we get the desired expression. □

If at least one of the two matrices in a matrix product is a gradient matrix, we can say more about the size of the elements of the product. In the following lemma, we use $A \leqslant 0$ (or $A \geqslant 0$) to express that all elements of A are nonpositive (or non-negative).

LEMMA 4.8 Let $G \in \mathbb{R}^{n\times m}$ be a gradient matrix with entries in $\{-1, 0, 1\}$ and let $A \in \mathbb{R}^{m\times l}$. If $A \leqslant 0$ or $A \geqslant 0$ then

$$\mu(GA) \leqslant \mu(A).$$

Proof. This inequality follows immediately from the properties of a gradient matrix as given in Definition 1.1. An entry of GA is either equal to an entry of A or equal to the difference of two entries of A of equal sign, so $\mu(GA) \leqslant \mu(A)$. □

LEMMA 4.9 Let $G_1 \in \mathbb{R}^{n\times m}$ and $G_2 \in \mathbb{R}^{m\times m}$ be gradient matrices with entries in ± 1. Let G_2 be invertible; then

$$\mu(G_1 G_2^{-1}) \leqslant 1.$$

Proof. The matrix G_2 is an invertible gradient matrix with entries in ± 1, so we can apply Lemma 4.3 and obtain a factorization $G_2^{-1} = Q_2 U^{-1} D Q_1$, where Q_1 and Q_2 are permutations, D is a diagonal matrix with ± 1 on the diagonal and U is a unit upper triangular gradient matrix. If we insert the expression for G_2^{-1} in μ, we get

$$\mu(G_1 G_2^{-1}) = \mu(G_1 Q_2 U^{-1} D Q_1) = \mu(G_1 Q_2 U^{-1}),$$

where the last equality holds because μ is independent of permutations and diagonal scaling with ± 1. (This follows from the definition, but can be derived from Lemma 4.6 as well.) Note that $G_1 Q_2$ is a gradient matrix and that $0 \leqslant U^{-1}$ because of Corollary 4.5. This allows us to apply Lemma 4.8 and we obtain

$$\mu(G_1 G_2^{-1}) = \mu(G_1 Q_2 U^{-1}) \leqslant \mu(U^{-1}) = 1.$$

□

LEMMA 4.10 Let $G_1 \in \mathbb{R}^{m\times m}$ and $G_2 \in \mathbb{R}^{m\times n}$ be gradient matrices with entries in ± 1 such that the matrix $[G_1 G_2]$ is a gradient matrix as well. Let G_1 be invertible; then

$$\mu(G_1^{-1} G_2) \leqslant 1.$$

Proof. We extend the composite matrix $[G_1 G_2]$ with a zero block and a minus identity matrix such that we get the invertible gradient matrix

$$G = \begin{pmatrix} G_1 & G_2 \\ 0 & -I \end{pmatrix} \quad \text{with inverse } G^{-1} = \begin{pmatrix} G_1^{-1} & G_1^{-1} G_2 \\ 0 & -I \end{pmatrix}.$$

Lemma 4.3 applies to G, so we immediately see that the matrix $G_1^{-1} G_2$, which is a submatrix of G^{-1}, has entries in ± 1. □

We now have all the tools to prove the theorem.

Proof of Theorem 4.1. For part (i), the goal is to obtain a bound on the size of the entries in $K^{(l)}$ in terms of the largest entry of $K^{(0)}$. We start with the following considerations.

(1) The size of the entries in $K^{(l)}$ is not influenced by a change of ordering within the first l V-nodes and the corresponding P-nodes.

(2) $K^{(0)}$ is an $\mathscr{F}$-matrix, so Remark 3.6 implies that $\mu(B^{(l)}) \leqslant \mu(B^{(0)}) \leqslant 1$ for all l. The critical part is the growth of the entries in $A^{(l)}$, so we will try to bound the growth factor ρ_A of A during the elimination.

(3) Theorem 3.3 states that for all l the matrix $A^{(l)}$ is positive definite. Gaussian elimination on symmetric positive definite matrices is unconditionally stable with growth factor $\rho < 1$ (see Problem 10.4 in Higham, 2002). Hence, we can ignore the elimination of single V-nodes in the computation of the growth factor. Consequently, we can focus on the effect of the elimination of the coupled V- and P-nodes.

Suppose that at a certain stage of Gaussian elimination we used k 2×2 pivots. Then, we can reorder and split the original matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

such that the k 2×2 pivots are in the first blocks A_{11} and B_{11}. Note that with this choice B_{11} is square. If we insert the splitting of the matrices A and B in K, we get, after a little rearrangement,

$$\left(\begin{array}{cc|cc} A_{11} & B_{11} & A_{12} & B_{12} \\ B_{11}^{\mathrm{T}} & 0 & B_{21}^{\mathrm{T}} & 0 \\ \hline A_{21} & B_{21} & A_{22} & B_{22} \\ B_{12}^{\mathrm{T}} & 0 & B_{22}^{\mathrm{T}} & 0 \end{array}\right). \tag{4.2}$$

The matrix B_{11} is square and invertible, so

$$\begin{pmatrix} A_{11} & B_{11} \\ B_{11}^{\mathrm{T}} & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & B_{11}^{-\mathrm{T}} \\ B_{11}^{-1} & -B_{11}^{-1} A_{11} B_{11}^{-\mathrm{T}} \end{pmatrix}. \tag{4.3}$$

Hence, the Schur complement of the first two block rows becomes

$$\begin{pmatrix} S_{22} & B_{22} - B_{21} B_{11}^{-1} B_{12} \\ B_{22}^{\mathrm{T}} - B_{12}^{\mathrm{T}} B_{11}^{-\mathrm{T}} B_{21}^{\mathrm{T}} & 0 \end{pmatrix}$$

with

$$S_{22} = A_{22} - A_{21} B_{11}^{-\mathrm{T}} B_{21}^{T} - B_{21} B_{11}^{-1} A_{12} + B_{21} B_{11}^{-1} A_{11} B_{11}^{-\mathrm{T}} B_{21}^{T}.$$

Because of Theorem 3.4, we know that $B_{22} - B_{21} B_{11}^{-1} B_{12}$ is a gradient matrix (with a bound of 1); hence, to compute the growth factor we have to focus on the block S_{22}:

$$\mu(S_{22}) \leqslant \mu(A_{22}) + \mu(A_{21} B_{11}^{-\mathrm{T}} B_{21}^{T}) + \mu(B_{21} B_{11}^{-1} A_{12}) + \mu(B_{21} B_{11}^{-1} A_{11} B_{11}^{-\mathrm{T}} B_{21}^{T}). \tag{4.4}$$

We now apply Lemma 4.6 in order to get

$$\mu(S_{22}) \leqslant \mu(A_{22}) + 2\chi(A_{12})\mu(A_{12})\mu(B_{21} B_{11}^{-1}) + k\chi(A_{11})\mu(A_{11})\mu(B_{21} B_{11}^{-1})^2. \tag{4.5}$$

The matrices B_{21} and B_{11} are gradient matrices and B_{11} is invertible, so we can apply Lemma 4.9 and that gives

$$\mu(B_{21} B_{11}^{-1}) \leqslant 1. \tag{4.6}$$

This simplifies the bound to

$$\mu(S_{22}) \leqslant \mu(A_{22}) + 2\chi(A_{12})\mu(A_{12}) + k\chi(A_{11})\mu(A_{11}). \tag{4.7}$$

It is obvious that for all four subblocks of A, we have $\mu(A_{ij}) \leqslant \mu(A)$. Furthermore, A is positive definite, so it has strictly positive diagonal entries and the off-diagonal block A_{12} satisfies $\chi(A_{12}) \leqslant \chi(A) - 1$. Using $\chi(A_{11}) \leqslant \chi(A)$ and $k \leqslant m$, we finally get

$$\mu(S_{22}) \leqslant (1 + 2(\chi(A) - 1) + m\chi(A))\mu(A).$$

Division by $\mu(A)$ gives the bound for the growth factor ρ_A of A. Next, we have to combine this result with the fact that there is no growth in the elements of $B^{(l)}$ during the elimination (see Remark 3.6) in order to obtain a growth factor for K. The maximum of K is just the maximum of $\mu(A)$ and 1, and the maximum over all $K^{(l)}$ is just the maximum over all $A^{(l)}$ and 1, where the maximum over all $A^{(l)}$ is precisely $\rho_A \mu(A)$, which leads to the bound given in part (i).

For part (ii), the bound on the size of the elements in L, it is sufficient to look at a general lower diagonal block because for each entry of L we can define a block that includes that entry. If we use the splitting in (4.2) and the inverse as in (4.3), a lower diagonal block of L is given by

$$\begin{pmatrix} A_{21} & B_{21} \\ B_{12}^{\mathrm{T}} & 0 \end{pmatrix}\begin{pmatrix} A_{11} & B_{11} \\ B_{11}^{\mathrm{T}} & 0 \end{pmatrix}^{-1} = \begin{pmatrix} B_{21}B_{11}^{-1} & A_{21}B_{11}^{-\mathrm{T}} - B_{21}B_{11}^{-1}A_{11}B_{11}^{-\mathrm{T}} \\ 0 & B_{12}^{\mathrm{T}}B_{11}^{-\mathrm{T}} \end{pmatrix}. \tag{4.8}$$

Now, we have to find the bounds for the subblocks. With (4.6), we already have a bound for the heading block. For the (2, 2) block, we can use Lemma 4.10, which gives

$$\mu(B_{12}^{\mathrm{T}}B_{11}^{-\mathrm{T}}) = \mu(B_{11}^{-1}B_{12}) \leqslant 1. \tag{4.9}$$

The right upper block is a bit more difficult, but once more using (4.6) and Lemma 4.6, we get

$$\begin{aligned} \mu(A_{21}B_{11}^{-\mathrm{T}} - B_{21}B_{11}^{-1}A_{11}B_{11}^{-\mathrm{T}}) &\leqslant \chi(A_{21})\mu(A_{21})\mu(B_{11}^{-\mathrm{T}}) \\ &\quad + k\mu(B_{21}B_{11}^{-1})\chi(A_{11})\mu(A_{11})\mu(B_{11}^{-\mathrm{T}}) \\ &\leqslant (\chi(A)-1)\mu(A) + k\chi(A)\mu(A) \\ &\leqslant ((m+1)\chi(A)-1)\mu(A). \end{aligned} \tag{4.10}$$

Combining the bounds finishes the proof. □

Compared to the general bound $\rho \leqslant (2.57)^{n+m-1}$, this bound is very sharp. In practice, even this bound for $\mathscr{F}$-matrices is hardly attained, first of all because the bounds in Lemmas 4.6 and 4.7 are quite pessimistic. Especially, if we choose fill-reducing orderings for $F(A) \cup F(BB^{\mathrm{T}})$, the matrix $B_{21}B_{11}^{-1}$ will not be a matrix full of 1s, but sparse, and therefore $B_{21}B_{11}^{-1}A_{11}B_{11}^{-\mathrm{T}}B_{21}^{T}$ will have a maximum element much smaller than $k\chi(A)\mu(A)$. Even if $B_{21}B_{11}^{-1}$ would be a matrix full of 1s, it is likely that the product will have a smaller maximum value because in general summation of entries on a row in A_{11} will be much smaller than $\chi(A)\mu(A)$.

For one special case, we can give slightly better bounds, i.e. when A is diagonally dominant.

THEOREM 4.11 Let K be an $\mathscr{F}$-matrix with an ordering given by Algorithm 2.2 and let m be the number of columns in B. Furthermore, let A be a diagonally dominant; then for the LDL^{T}-factorization of K it holds that: (i) the growth factor is bounded by

$$\rho \leqslant \frac{\max(\mu(A)\rho_A, 1)}{\max(\mu(A), 1)}, \quad \text{with } \rho_A \leqslant 2m+3,$$

and (ii) the elements in L are bounded by

$$\mu(L) \leqslant \max\{(2m+1)\mu(A), 1\}.$$

Hence, the factorization is numerically stable.

Proof. We can give sharper bounds on the terms in (4.4). Because A is symmetric and diagonally dominant, we have

$$\sum_{j=1, j\neq i}^{n} |a_{ij}| \leqslant |a_{ii}| \leqslant \mu(A).$$

It immediately follows that the off-diagonal blocks A_{12} and A_{21} have row sums smaller than $\mu(A)$. If we use that in (4.1), we can derive a sharper bound for $\mu(A_{21}B_{11}^{-\mathrm{T}}B_{21}^{T})$, i.e.

$$\mu(A_{21}B_{11}^{-\mathrm{T}}B_{21}^{T}) = \mu(B_{21}B_{11}^{-1}A_{12}) \leqslant \mu(B_{21}B_{11}^{-1})\mu(A) \leqslant \mu(A).$$

We can apply Lemma 4.6 in combination with Lemma 4.7 to the last term in (4.4):

$$\mu(B_{21}B_{11}^{-1}A_{11}B_{11}^{-\mathrm{T}}B_{21}^{T}) \leqslant k\mu(B_{21}B_{11}^{-1})\mu(B_{21}B_{11}^{-1}A_{11}) \leqslant 2k\mu(B_{21}B_{11}^{-1})^2\mu(A_{11}) \leqslant 2m\mu(A),$$

where we again used the fact that $\mu(B_{21}B_{11}^{-1}) \leqslant 1$, $\mu(A_{ij}) \leqslant \mu(A)$ and $k \leqslant m$. Combining all estimates gives

$$\mu(S_{22}) \leqslant (2m+3)\mu(A).$$

Division by $\mu(A)$ provides the bound for the growth factor ρ_A.

With respect to the bound on the elements of L, if A is diagonally dominant, we have instead of (4.10) the bound

$$\mu(A_{21}B_{11}^{-\mathrm{T}} - B_{21}B_{11}^{-1}A_{11}B_{11}^{-\mathrm{T}}) \leqslant \mu(A) + 2k\mu(A) \leqslant (2m+1)\mu(A). \qquad \square$$

REMARK 4.12 The proofs of Theorems 4.1 and 4.11 can be extended to a wider class of saddle point matrices. The only relevance of $\mathscr{F}$-matrices lies in the bounds $\mu(B_{21}B_{11}^{-1}) \leqslant 1$ and $\mu(B_{11}^{-1}B_{12}) \leqslant 1$. If B is not a gradient matrix, it might have other properties such that the entries of both matrices are bounded. That would be enough to compute much better bounds for the growth factor and for the size of the elements in L than the general ones for indefinite matrices.

EXAMPLE 4.13 At the end of this section, we show as an example the growth factor for $\mathscr{F}$-matrices from a Stokes problem on a square staggered grid. We will describe the problem in more detail in Section 6. We compute an ordering for the matrix with Algorithm 2.2 based on three different initial orderings for $F(A) \cup F(BB^{\mathrm{T}})$ [natural, reverse Cuthill–McKee (George & Liu, 1981) and approximate minimum degree (Amestoy *et al.*, 1996, 2004)]. We perform Gaussian elimination on the reordered matrix, meanwhile monitoring the growth of the entries in the Schur complement. The results can be found in Fig. 5 and Table 1.

In Fig. 5, we plot $\rho_A^{(l)} = \mu(A^{(l)})/\mu(A^{(0)})$ for each Schur complement l for a square grid of size 9×9. The actual growth factor is the maximum of all $\rho_A^{(l)}$s. For all three orderings, the growth factor is rather small. The theoretical bound can be computed using Theorem 4.11 because the matrix is diagonally dominant. Here, the bound is $\rho_A \leqslant 2m+3 = 163$, which is still quite far from the computed values of ρ_A. It is of course much better than the general bound $2.57^{223} \approx 2.60 \times 10^{91}$. Of interest is the effect of the different ordering algorithms on the growth of the elements. In the case of the natural ordering, the maximum growth is realized almost immediately after elimination of the first row in the grid. After that, the growth remains constant except for a tiny peak halfway. The best results in terms of a small

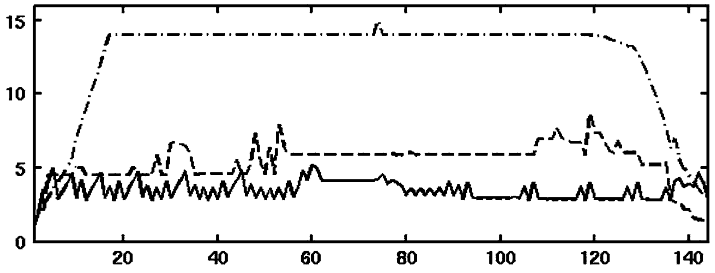

FIG. 5. $\mu(A^{(l)})/\mu(A^{(0)})$ during Gaussian elimination for the natural (dash-dot), approximate minimum degree (dash) and reverse Cuthill–McKee (solid) for the Stokes problem on a square grid of 9×9 cells, $n = 144$, $m = 80$.

TABLE 1 *Growth factor ρ_A for several Stokes matrices for the natural ordering (*nat*), reverse Cuthill–McKee (*rcm*), approximate minimum degree (*amd*) and the theoretical upper bound given by Theorem* 4.11 *(*Bound*). The column n contains the number of V-nodes and m the number of P-nodes*

Matrix	n	m	nat	rcm	amd	Bound
Stokes 3×3	12	8	6.0	5.6	6.5	19
Stokes 5×5	40	24	9.0	5.0	7.5	51
Stokes 9×9	144	80	15.0	5.0	8.5	163
Stokes 17×17	544	288	27.0	5.0	14.0	579
Stokes 33×33	2,112	1,088	51.0	5.0	15.0	2,179
Stokes 63×63	8,320	4,224	99.0	5.0	15.0	8,451

growth factor are obtained for the reverse Cuthill–McKee ordering. This is in agreement with a theorem in Bohte (1975). Apparently, a small bandwidth is a good guarantee for a small growth factor.

To illustrate how the growth factor ρ_A of A depends on the size of the grid and the number of P-nodes we show Table 1. As one can see, the growth factor remains very small in the case of reverse Cuthill–McKee. It seems to be more or less independent of the grid size. This is in contrast to the natural ordering, which gives a growth factor that increases with the grid size, by approximately a factor of 1.5 if the number of nodes is doubled in two directions; more precisely the data are fitted exactly by $3(1 + \sqrt{m+1})/2$. Note that it still grows less than linearly with m. The growth factor of approximate minimum degree is somewhere in between those two and appears to flatten for bigger problems. By inspection of the factorization, we saw that also the elements of L are bounded for reverse Cuthill–McKee and approximate minimum degree. In all cases, the bound ρ_A provided by Theorem 4.11 is quite pessimistic. In the experiments, it was easy to find a scaling that bounded the elements of L by one for all resolutions, independent of the mesh size. These two observations led to the conjecture posted in Remark 4.2.

We can conclude that for Gaussian elimination on $\mathscr{F}$-matrices both the growth factor ρ and the size of the elements in L are bounded by a number that grows linearly with m, i.e. the number of indefinite nodes. So Gaussian elimination is numerically stable for $\mathscr{F}$-matrices and iterative refinement is hardly needed to compute accurate solutions.

5. Fast insertion of *P*-nodes

The application of Algorithm 2.2 requires the insertion of the P-nodes into the ordering for the V-nodes. In this section, we show that this can be done very efficiently.

In Corollary 3.2, we expressed that the sequence of $B^{(l)}$s is independent of A; it only depends on the ordering of the V-nodes. We can exploit this for the insertion of P-nodes in the ordering for the V-nodes. The insertion can be done before the actual elimination based on the entries of B only. Especially, if we deal with an $\mathscr{F}$-matrix, an efficient algorithm exists. We only need to know the sparsity pattern of B.

The P-nodes are inserted by Algorithm 5.1. The basic idea is that we track the elimination of P-nodes in an ancestor array π. At the beginning, all nodes are their own ancestors, so $\pi[i] = i$ for all $i = 1, \ldots, m$. We add one extra element to this vector: $\pi[m+1] = m+1$. If a node v in V is coupled to two nodes j and k in P, it has to be eliminated together with one of them. Say we eliminate v and j together. As we saw in the proof of Theorem 3.4, all V-nodes that were coupled to j get coupled to k instead. We reflect this by a change in the ancestor array: $\pi[j] = k$; k becomes the ancestor of j. In the ancestor array, we can easily check what happened to the P-nodes by iteration on the array until a fixed point is found. All fixed points are not yet eliminated P-nodes. There is freedom to choose j or k. It is beneficial and in agreement with the minimum degree idea to pick the one with the least fill in the corresponding column of $B^{(l)}$. We will use an array nnz to store an estimate of the number of nonzeros in the columns of $B^{(l)}$. The array is used to make the decision as to whether j or k is to be eliminated. We will explain this in more detail in Remark 5.5.

ALGORITHM 5.1 Insert the P-nodes in the ordering for V.
Assume that we have a permutation $q_v = \{q_v(1), q_v(2), \ldots, q_v(n)\}$ for the nodes in V.

```
1. Initialize the ancestor array $\pi[i] = i$ for $i = 1, \ldots, n$.
2. Initialize $nnz[i]$—the number of nonzeros on the $i$-th column in $B$—for $i = 1, \ldots, n$.
3. Initialize final ordering $q_f = \emptyset$.
4. For $i = 1$ to $n$
   (a) Find $j$ and $k$, the row numbers of the entries on the $q_v(i)$-th row of $B$.
   (b) If there is only one entry assign $k := (m+1)$.
   (c) If the row is empty assign $j := (m+1)$ and $k := (m+1)$.
   (d) Trace back ancestors of $j$ and $k$ until fixed points are found:
       - Repeat $j := \pi[j]$ until $j == \pi[j]$
       - Repeat $k := \pi[k]$ until $k == \pi[k]$
   (e) If $j == k$
       - Expand the ordering $q_f := \{q_f, q_v(i)\}$.
   (f) Else
       - If $(j == (m+1))$ or $(nnz[k] < nnz[j])$, swap $j$ and $k$,
       - Expand the ordering $q_f := \big\{q_f, q_v(i), j+n\big\}$,
       - Change $\pi$ to reflect elimination: $\pi[j] := k$.
       - Estimate $nnz$ after elimination: $nnz[k] := nnz[j] + nnz[k] - 2$.
   (g) Endif
5. Return $q_f$.
```

We add a few remarks on this algorithm.

REMARK 5.2 One might fear cycles in the ancestor array π such that the algorithm never terminates Step 4(d). Fortunately, the only cycles that can appear in π are the fixed points. This can be simply explained as follows. Initially, all points are fixed points. The vector is changed only at Step 4(f) where one fixed point of the ancestor array is replaced by a reference to another fixed point. So during the algorithm all paths in π will always end in a fixed point; hence, the algorithm will certainly terminate.

REMARK 5.3 The length of the iterations at Step 4(d) is m in the worst-case scenario. The number of iterations is related to the depth of the elimination tree with respect to the nodes in P. If we use a fill-reducing ordering for a 2D problem, it will hardly ever be bigger than $\sqrt{m}$. In any case, the time used by the algorithm was always a fraction of the time used to compute the ordering for $F(A) \cup F(BB^{\mathrm{T}})$.

REMARK 5.4 If we trace back the ancestors of two nodes j and k in Step 4(d), the fixed point ancestors that we finally get might very well be equal. If they are, the two entries have the same magnitude but opposite sign at the same position, so summing up gives zero: they annihilate. Hence, in that case, the node v is no longer coupled to a pressure node, so we cannot insert any. This conforms with Step 4(e). Only if we have an $\mathscr{F}$-matrix, do we know exactly when this exact cancellation happens.

REMARK 5.5 If there are two entries in a row, say at positions j and k, we have a choice in Step 4(f) as to which of the two we will eliminate and insert in the ordering. Remember that the goal is to reduce the fill in the factorization. In (3.1), we see that the amount of (new) fill in the Schur complement $A^{(l+1)}$ is determined by the number of nonzeros in the vector $(\hat{b}/\beta)^{\mathrm{T}}$, i.e. the column in $B^{(l)}$ that belongs to the P-node that is eliminated. So to reduce the number of nonzeros in $A^{(l+1)}$, we should eliminate the node with the fewest entries in its column. However, it is too expensive to compute the number of nonzeros in the columns of $B^{(l)}$ precisely; therefore, we estimate this number in the array nnz. At Step 2, the number $nnz[i]$ is computed as the number of nonzeros on the ith column in B. At that step it is still exact. However, if at Step 4(f) j is eliminated we estimate that the number of nonzeros in the column of k in $B^{(l+1)}$ is equal to the sum of the number of entries in the two columns minus 2. We subtract 2 because we delete the first row in $B^{(l)}$. This number is an overestimate because it does not take account of exact cancellation. Nevertheless, the approximation is good enough to make the right decision in Step 4(f). If the decision leads to an elimination of k instead of j, we swap both entries in order to simplify the notation in the algorithm.

If we do not have an $\mathscr{F}$-matrix, a similar algorithm can be used. Corollary 3.2 holds for general saddle point matrices, so still only B and an ordering for V are needed to insert the P-nodes. Nevertheless, it will be less efficient because we cannot benefit from a simple structure of B. Furthermore, we need to monitor the size of the entries in $B^{(l)}$ as well and it will be more difficult to detect exact cancellation.

6. Numerical results

In this section, we will show the results of our algorithm for two sets of matrices. We implemented Algorithms 2.2 and 5.1 in MATLAB 7.1.0.183 (R14) Service Pack 3. The first ordering is computed with MATLAB's symmetric approximate minimum degree ordering SYMAMD (Amestoy *et al.*, 1996, 2004).

We compare the results of Algorithm 2.2 to those of PARDISO version 3.1 (serial) (Schenk & Gärtner, 2004, 2006; Schenk *et al.*, 2000), which is able to factorize indefinite symmetric matrices. It uses either AMD (approximate minimum degree) or METIS (nested dissection, Karypis & Kumar, 1998) as the basic ordering. We use the standard parameter settings except that we follow the advice of the manual to use scaling (IPARM(11) = 1) and weighted matchings (IPARM(13) = 1) in the case of highly indefinite matrices like saddle point problems. If we choose to switch off weighted matchings, PARDISO is still able to build a factorization, but the error in the solution without using iterative refinement rises from $\mathscr{O}(10^{-13})$ to $\mathscr{O}(10^{-6})$. So we really need weighted matchings to compute an accurate factorization.

The package PARDISO offers the possibility of ignoring its ordering algorithms and instead performs factorization based on an ordering provided by the user. We use this facility with the ordering of Algorithm 2.2 as input. In the tables in this section, we will use 'PARDISO(amd/metis/uo)' to denote the PARDISO factorization based on amd, metis and the user ordering, respectively. The MATLAB factorization is called 'LDL^{T}(amd)'.

All numerical experiments were done on a PC with two 2.4-GHz AMD Opteron processors and 7.6-GB memory.

6.1 *Stokes flow in a driven cavity*

The first problem is a 2D Stokes equation in a driven cavity. Here, the following set of equations has to be solved on the unit square Ω:

$$\left.\begin{aligned} -\nu \Delta u + \nabla p &= 0 \\ \nabla \cdot u &= 0 \end{aligned}\right\}, \tag{6.1}$$

where $u(x, y)$ is the velocity field and $p(x, y)$ the pressure field; the parameter ν controls the amount of viscosity. The nontrivial solution is determined by the boundary conditions, which are zero on three sides of the unit square. At the upper boundary ($y = 1$), we prescribe a horizontal velocity $u(x, 1) = 1$.

We can get rid of the parameter ν by defining a new pressure variable $\bar{p} = p/\nu$. If the first equation is divided by ν, we can substitute p by $\bar{p}$ and the parameter ν is gone. So we may assume that $\nu = 1$.

If the equations are discretized on a uniform staggered grid (a C-grid) with mesh size h, we get an $\mathscr{F}$-matrix. It is singular because the pressure field is determined up to a constant. We get rid of this problem by fixing one pressure node in the domain. So the number of pressure nodes is reduced by one. Table 2 shows the size and the number of nonzeros in the upper triangular part of the Stokes matrices.

The results of factorization of these matrices with different ordering algorithms can be found in Table 3. Obviously, Algorithm 2.2 is able to produce an ordering that gives a factorization with low fill. For all grid sizes, it results in a factorization that is sparser than the factorizations of PARDISO. The large gap between the numbers of nonzeros for PARDISO(uo) and LDL^{T}(amd) is remarkable since

TABLE 2 *The set of Stokes matrices. n is the number of V-nodes, m the number of P-nodes and* nnz *the number of nonzeros in the upper triangular part of the matrix*

Matrix	n	m	$n+m$	nnz
Stokes 3 × 3	12	8	20	48
Stokes 5 × 5	40	24	64	180
Stokes 9 × 9	144	80	224	684
Stokes 17 × 17	544	288	832	2,652
Stokes 33 × 33	2,112	1,088	3,200	10,428
Stokes 65 × 65	8,320	4,224	12,544	41,340
Stokes 129 × 129	33,024	16,640	49,664	164,604
Stokes 257 × 257	131,584	66,048	197,733	656,892
Stokes 513 × 513	525,312	263,168	788,480	2,624,508

TABLE 3 *Number of nonzeros for various factorizations of the Stokes matrices*

Matrix	PARDISO(amd)	PARDISO(nd)	PARDISO(uo)	LDL^{T}(amd)	Estimate
Stokes 3 × 3	146	129	114	82	49
Stokes 5 × 5	646	743	555	403	279
Stokes 9 × 9	3,203	3,668	2,829	2,134	1,705
Stokes 17 × 17	18,060	20,731	15,296	11,415	9,700
Stokes 33 × 33	98,936	114,307	79,131	63,304	56,926
Stokes 65 × 65	529,368	594,132	448,320	365,311	345,310
Stokes 129 × 129	2,761,774	3,014,234	2,444,302	2,039,458	1,995,350
Stokes 257 × 257	13,999,517	14,615,960	12,682,925	10,877,966	10,838,918
Stokes 513 × 513	69,350,006	69,354,497	63,766,827	55,900,331	56,276,468

the same orderings are used. The only explanation we have is that probably the L-factor of PARDISO contains a lot of entries close to machine precision because it cannot detect exact cancellation, as we could in our algorithm.

The last column in Table 3 contains the number of nonzeros in the LDL^{T}-factorization of the reordered fill matrix $F(A) \cup F(BB^{\mathrm{T}})$. We used MATLAB's symbfact function to compute this number. If we compare the last two columns, we can conclude that the number of nonzeros in the symbolic factorization of $F(A) \cup F(BB^{\mathrm{T}})$ based on the first ordering q_V provides a reasonable estimate for the number of nonzeros that we get in the factorization of the $\mathscr{F}$-matrix K after extension of the ordering to the whole matrix with Algorithm 2.2. The estimate becomes even better for larger matrices.

6.2 *Matrices of Tůma*

The second set of test matrices consists of seven matrices provided by Miroslav Tůma, of which four can be found in Tůma (2002). The matrices arise in mixed-hybrid finite-element discretizations of 3D potential fluid flow problems (Maryška *et al.*, 2000). In Table 4, we show the size and number of nonzeros of the matrices. In Tůma (2002), the matrices have a slightly larger number of nonzeros because the sparse matrix format contains some 'zero nonzeros' that we removed *a priori*. We have to remark that the Tůma matrices do not satisfy the assumption that the number of nonzeros in a row in B is smaller than the number of nonzeros in A. All As in the Tůma set have at most three nonzeros per row, while at least a quarter of the rows in B have more than three nonzeros. Maybe Rule 2.1 is not the best in that case. Possibly, a weakening of the rule could provide a further improvement of the performance.

Table 5 contains the number of nonzeros of the factors with different fill-reducing ordering algorithms. We compare the results of Algorithm 2.2 with the results of Tůma and PARDISO. The ordering of Tůma is better than the two orderings of PARDISO. However, in all cases our ordering gives the sparsest factors. The big difference between the approximate minimum degree and nested dissection ordering in PARDISO is quite remarkable. It must be due to the structure of the Tůma matrices because for the Stokes matrices the results for both orderings are similar. The qualitative difference between the two sets of matrices is noticed as well in the effect of the choice we made in Remark 5.5. If we do not eliminate the node that has the least nonzeros in its column of B^{T}, but simply pick the first node, the results of LDL^{T}(amd) in Table 3 hardly change, whereas the results in Table 5 seriously deteriorate.

Also, here we added in the last column the estimate of the number of nonzeros of LDL^{T}(amd), which is in this case slightly worse than that for the Stokes matrices.

TABLE 4 *The set of Tůma's matrices. n is the number of V-nodes, m the number of P-nodes and nnz the number of nonzeros in the upper triangular part of the matrix*

Matrix	n	m	$n+m$	nnz
S3P	270	207	477	1,008
M3P	2,160	1,584	3,744	8,136
DORT2	7,515	5,477	12,992	28,440
DORT	13,360	9,607	22,967	50,560
L3P	17,280	12,384	29,664	65,376
dan2	63,750	46,661	110,411	318,304
novak	152,645	114,113	266,758	744,912

TABLE 5 *Number of nonzeros in various factorizations of the Tůma matrices. The column Tůma contains results from Tůma (2002)*

Matrix	PARDISO(amd)	PARDISO(nd)	PARDISO(uo)	LDL^{T}(amd)	Tůma	Estimate
S3P	3,154	3,290	2,754	2,195	2,957	1,729
M3P	60,305	61,144	45,248	35,553	44,002	31,469
DORT2	558,522	276,376	243,088	208,393	231,312	188,697
DORT	1,278,505	641,788	598,130	527,602	551,215	491,736
L3P	1,445,900	1,000,789	785,689	690,932	—	643,277
dan2	4,465,051	3,040,497	2,686,839	2,240,202	—	1,973,721
novak	15,453,022	9,785,185	9,351,503	8,217,979	—	7,577,723

7. Discussion

In this paper, we proposed a new algorithm to compute a stable fill-reducing ordering for $\mathscr{F}$-matrices, a special class of symmetric saddle point matrices. The algorithm is based on a simple idea: compute an ordering for the V-nodes first and then add the P-nodes. The ordering for the V-nodes is a fill-reducing ordering for $F(A) \cup F(BB^{\mathrm{T}})$. The P-nodes are added under the rule 'eliminate P-nodes as soon as possible'. The final ordering is guaranteed to be feasible and in the case of $\mathscr{F}$-matrices it can be computed very fast.

In Section 3, we showed that the ordering guarantees that much of the structure of the saddle point problem is preserved in the Schur complements during Gaussian elimination; in particular the zero block remains empty and $B^{(l)}$ is independent of $A^{(l)}$. Positive definiteness of A and the $\mathscr{F}$-matrix properties are also preserved. From an engineering point of view, this is a very attractive property. Moreover, it eases the proof of the numerical stability of the method.

One of the important results is Theorem 4.1, where we give a bound for the growth factor in Gaussian elimination on $\mathscr{F}$-matrices as well as a bound for the size of the elements in the factor L of the LDL^{T}-factorization. The general bound for the growth factor grows exponentially fast, but it is too pessimistic. At least in the case of $\mathscr{F}$-matrices with the ordering of Algorithm 2.2, the growth factor is bounded by a number that grows linearly with the dimension. It is likely that this holds for a larger class of saddle point matrices because it seems possible to weaken the assumptions of the proof. If B is not a gradient matrix, it might have other properties such that $B_{21}B_{11}^{-1}$ and $B_{11}^{-1}B_{12}$ are bounded, as we have argued in Remark 4.12. We have to emphasize the word 'might' in the previous sentence, as clearly there exist examples of Bs such that entries in $B_{21}B_{11}^{-1}$ become very large.

The numerical experiments on the Stokes and Tůma matrices show that the algorithm is able to produce a good factorization with a sparser structure than the factorizations of other methods.

Another nice property is that the very same algorithm can be used for more general $\mathscr{F}$-matrices like the 2D Navier–Stokes equations (with Coriolis force) on a C-grid.

There is a lot more to study. In some cases, Rule 2.1 should be weakened to get a sparser factorization. There might be better ways to compute the first ordering because we lose information in the construction of $F(A) \cup F(BB^{\mathrm{T}})$. In general, we overestimate the fill with this matrix. For example, the matrix will even be dense if B contains a single full row. One more important question: how do we generalize the algorithm such that it is able to handle a saddle point matrix that is not an $\mathscr{F}$-matrix? Clearly, there are several ways to do this, but which one is the best, and can we keep the nice properties that we showed in Section 3?

Another interesting subject is the generalization to higher-order discretizations of the gradient in flows. In many cases, the new gradient matrix can be written as the product of a smoothing matrix and a gradient matrix as defined in this paper. This knowledge should be used to derive a variant of the current algorithm and to prove the stability of the algorithm. For other generalizations, we would start over again from a nested dissection (see Remark 2.4) of the problem. The separators surrounding a subdomain should be such that the problem on that subdomain is well conditioned or, in partial differential equation terminology, it should be well posed.

Finally, in this paper we showed that the simple ideas behind the algorithm make sense. The construction of the ordering is straightforward and fast. It keeps nice properties of the matrix during Gaussian elimination and in the experiments it appears to be powerful enough to result in a factorization with significantly lower fill than the factorizations of existing methods.

Acknowledgements

We thank Miroslav Tůma for the permission to use his set of $\mathscr{F}$-matrices from Tůma (2002). We also thank the referees for their constructive remarks, which helped to improve the paper.

Funding

Technology Foundation STW; Applied Science Division of NWO; Technology Programme of the Ministry of Economic Affairs GWI (5798).

References

AMESTOY, P. R., DAVIS, T. A. & DUFF, I. S. (1996) An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, **17**, 886–905.

AMESTOY, P. R., DAVIS, T. A. & DUFF, I. S. (2004) Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, **30**, 381–388.

ARIOLI, M. & MANZINI, G. (2003) Null space algorithm and spanning trees in solving Darcy's equation. *BIT*, **43**, 839–848.

ASHCRAFT, C., GRIMES, R. G. & LEWIS, J. G. (1999) Accurate symmetric indefinite linear equation solvers. *SIAM J. Matrix Anal. Appl.*, **20**, 513–561.

AXELSSON, O. (1994) *Iterative Solution Methods*. Cambridge: Cambridge University Press.

BENZI, M., GOLUB, G. H. & LIESEN, J. (2005) Numerical solution of saddle point problems. *Acta Numer.*, **14**, 1–137.

BOHTE, Z. (1975) Bounds for rounding errors in the Gaussian elimination for band systems. *J. Inst. Math. Appl.*, **16**, 133–142.

BUNCH, J. R. & KAUFMAN, L. (1977) Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comput.*, **31**, 163–179.

BUNCH, J. R. & PARLETT, B. N. (1971) Direct methods for solving symmetric indefinite systems of linear equations. *SIAM J. Numer. Anal.*, **8**, 639–655.

COOK, W. J., CUNNINGHAM, W. H., PULLEYBLANK, W. R. & SCHRIJVER, A. (1997) *Combinatorial Optimization*. New York: Wiley.

DE NIET, A. C. & WUBS, F. W. (2007) Two saddle point preconditioners for fluid flows. *Int. J. Numer. Methods Fluids*, **54**, 355–377.

DUFF, I. S., ERISMAN, A. M. & REID, J. K. (1986) *Direct Methods for Sparse Matrices*, Monographs on Numerical Analysis. Oxford: Clarendon Press.

DUFF, I. S., GOULD, N. I. M., REID, J. K., SCOTT, J. A. & TURNER, K. (1991) The factorization of sparse symmetric indefinite matrices. *IMA J. Numer. Anal.*, **11**, 181–204.

DUFF, I. S. & PRALET, S. (2007) Towards a stable static pivoting strategy for the sequential and parallel solution of sparse symmetric indefinite systems. *SIAM J. Matrix Anal. Appl.*, **29**, 1007–1024.

DUFF, I. S., REID, J. K., MUNKSGAARD, N. & NIELSEN, H. B. (1979) Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite. *J. Inst. Math. Appl.*, **23**, 235–250.

ELMAN, H. C., SILVESTER, D. J. & WATHEN, A. J. (2002) Performance and analysis of saddle point preconditioners for the discrete steady-state Navier–Stokes equations. *Numer. Math.*, **90**, 665–688.

GEORGE, A. (1973) Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, **10**, 345–363.

GEORGE, A. & LIU, J. W. H. (1981) *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall Series in Computational Mathematics. Englewood Cliffs: Prentice-Hall.

HAGEMANN, M. & SCHENK, O. (2006) Weighted matchings for the preconditioning of symmetric indefinite linear systems. *SIAM J. Sci. Comput.*, **28**, 403–420.

HIGHAM, N. J. (2002) *Accuracy and Stability of Numerical Algorithms*, 2nd edn. Philadelphia: SIAM.

HORN, R. A. & JOHNSON, C. R. (1985) *Matrix Analysis*. Cambridge: Cambridge University Press.

KARYPIS, G. & KUMAR, V. (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, **20**, 359–392.

KAY, D., LOGHIN, D. & WATHEN, A. J. (2002) A preconditioner for the steady-state Navier–Stokes equations. *SIAM J. Sci. Comput.*, **24**, 237–256.

LIU, J. W. H. (1990) The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, **11**, 134–172.

MARYŠKA, J., ROZLOŽNÍK, M. & TŮMA, M. (2000) Schur complement systems in the mixed-hybrid finite element approximation of the potential fluid flow problem. *SIAM J. Sci. Comput.*, **22**, 704–723.

MEURANT, G. (1999) *Computer Solution of Large Linear Systems*. Amsterdam: North-Holland.

PINAR, A., CHOW, E. & POTHEN, A. (2006) Combinatorial algorithms for computing column space bases that have sparse inverses. *Electron. Trans. Numer. Anal.*, **22**, 122–145.

PRALET, S. (2004) Constrained orderings and scheduling for parallel sparse linear algebra. *Ph.D. Thesis*, September 2004, CERFACS report TH/PA/04/105.

RÖLLIN, S. & SCHENK, O. (2005) *Maximum-Weighted Matching Strategies and the Application to Symmetric Indefinite Systems*. Lecture Notes in Computer Science, vol. 3732, Berlin: Springer pp. 808–817.

SCHENK, O. & GÄRTNER, K. (2004) Solving unsymmetric sparse systems of linear equations with PARDISO. *J. Future Gener. Comput. Syst.*, **20**, 475–487.

SCHENK, O. & GÄRTNER, K. (2006) On fast factorization pivoting methods for symmetric indefinite systems. *Electron. Trans. Numer. Anal.*, **23**, 158–179.

SCHENK, O., GÄRTNER, K. & FICHTNER, W. (2000) Efficient sparse *LU* factorization with left-right looking strategy on shared memory multiprocessors. *BIT*, **40**, 158–176.

TŮMA, M. (2002) A note on the LDL^T decomposition of matrices from saddle-point problems. *SIAM J. Matrix Anal. Appl.*, **23**, 903–915.

VAN DER VORST, H. A. (2003) *Iterative Krylov Methods for Large Linear Systems*. Cambridge Monographs on Applied and Computational Mathematics, vol. 13. Cambridge: Cambridge University Press.

VAVASIS, S. A. (1994) Stable numerical algorithms for equilibrium systems. *SIAM J. Matrix Anal. Appl.*, **15**, 1108–1131.

WEBB, J. (1993) Edge elements and what they can do for you. *IEEE Trans. Magnetics*, **29**, 1460–1465.