EFFICIENT PRECONDITIONING OF SEQUENCES OF NONSYMMETRIC LINEAR SYSTEMS

JURJEN DUINTJER TEBBENS AND MIROSLAV TŮMA*

Abstract. We present a new approach for approximate updates of factorized nonsymmetric preconditioners for solving sequences of linear algebraic systems. This approach is algebraic and it is theoretically motivated. It generalizes diagonal updates introduced by Benzi and Bertaccini [3, 9]. It is shown experimentally that this approach can be very beneficial. For example, it is successful in significantly decreasing the number of iterations of a preconditioned iterative method for solving subsequent systems of a sequence when compared with freezing the preconditioner from the first system of the sequence. In some cases, the updated preconditioners offer a rate of convergence similar to or even higher than the rate obtained when preconditioning with recomputed preconditioners. Since the updates are typically cheap and straightforward, their use is of practical interest. They can replace recomputing preconditioners, which is often expensive, especially in parallel and matrix-free environments.

Key words. preconditioned iterative methods, sparse matrices, sequences of linear algebraic systems, incomplete factorizations, factorization updates, Gauss-Jordan transformations, minimum spanning tree

AMS subject classifications. Primary 65F10, 65F50, 65N22, 65H10. Secondary 15A06.

1. Introduction.

We consider the solution of sequences of linear systems

$$A^{(i)}x = b^{(i)},\ i = 1, \ldots, \tag{1.1}$$

where $A^{(i)} \in \mathbb{R}^{n\times n}$ are general nonsingular sparse matrices and $b^{(i)} \in \mathbb{R}^n$ are corresponding right-hand sides. Such sequences arise in many applications like computational fluid dynamics, structural mechanics, numerical optimization as well as in solving non-PDE problems. For example, a system of nonlinear equations $F(x) = 0$ for $F : \mathbb{R}^n \to \mathbb{R}^n$ solved by a Newton or Broyden-type method leads to a sequence of problems

$$J(x_i)(x_{i+1} - x_i) = -F(x_i),\ i = 1, \ldots, \tag{1.2}$$

where $J(x_i)$ is the Jacobian evaluated in the current iteration x_i or its approximation [33], [34].

The solution of sequences of linear systems is the main bottleneck in many applications mentioned above. For instance, the solvers may need powerful preconditioners in order to be efficient and computing preconditioners $M^{(1)}, M^{(2)}, \ldots$ for individual systems separately can be very expensive. There is a strong need for reduction of costs by sharing some of the computational effort among the subsequent linear systems.

A way to reduce the overall costs for solving systems of the type (1.2) is to modify Newton's method by skipping some Jacobian evaluations as in the Shamanskii combination of Newton's method and the Newton-chord method [11], [54]. In this way we get a sequence of systems with identical matrices, and techniques for solving systems with more right-hand sides may be applied provided the right-hand sides are available a priori, see, e.g., [46], [25], [55], [60]. However, combinations of Newton's method

*Institute of Computer Science, Czech Academy of Sciences, Pod Vodárenskou věží 2, 18207 Praha 8, Czech Republic. This work is supported by the National Program of Research "Information Society" under project 1ET400300415. The work of the first author is also supported by project number KJB100300703 of the Grant Agency of the Academy of Sciences of the Czech Republic.

and the Newton-chord method have much weaker nonlinear convergence properties than the standard Newton method.

A different approach to reduce the overall costs, which is usually more efficient, is based on *freezing* the preconditioner (using the same preconditioner for a sequence of linear systems), but recomputing (approximate) Jacobians $A^{(i)}$ [12], [39], [40]. This approach is very natural in the context of a matrix-free environment, where the system matrices $A^{(i)}$ may be available only in the form of matrix-vector products (matvecs), see also the overview of matrix-free Newton-Krylov methods in [38].

Another way to avoid efficiency and/or memory related problems connected to algebraic preconditioning is to use conceptually simpler preconditioners derived from the physics of the problem. In some PDE problems the original operator can be replaced by a simpler one. Early results related to preconditioning by fast solvers can be found in [16], [24]. For instance, the simpler operator can be a scaled diffusion operator for a PDE with variable coefficients or a convection-diffusion operator [12], [36], [38]. In the algebraic setting, simple preconditioners derived from stationary iterative methods can be used. Preconditioning by the symmetric part of a nonsymmetric matrix was proposed in [17], [62], see also [14]. Another popular preconditioning technique for general convection-diffusion-reaction models is based on generalizations of ADI splitting from [49], see, e.g., [36]. Note that we restrict ourselves here to linear preconditioners; for nonlinear preconditioning techniques we refer, e.g., to [13] and the references therein. In order to make the preconditioning more efficient and to simplify the preconditioner setup even more, reformulations based on nested iterations were introduced, see, e.g., [59]. For instance, the flexible Krylov-subspace framework enables theoretically sound implementations of inner-outer Krylov-subspace methods [51], [56].

Freezing the preconditioner or using simple preconditioning techniques may not be enough for fast convergence in practice. Our contribution proposes new and efficient approximate updates of a preconditioner which is factorized as $LDU \approx A$. The updated preconditioners are then used for solving the subsequent members of the sequence. We do not assume any simple relation among the systems of the sequence. Note that straightforward approximate small rank preconditioner updates can be obtained in case of a sequence of linear systems from a quasi-Newton method, as shown in the SPD case in [45], [8]. It is well-known how to compute the *exact* updates of sparse decompositions [19], [20], [21]; the techniques for *dense* updates starting in early papers, e.g., [28], and having mostly the intent to be applied to the simplex method of linear programming and its extensions, are a classical part of numerical mathematics. Another algebraically-motivated strategy used in preconditioning sequences of systems is to use adaptive information generated by Krylov subspace methods [2]. Recent work on recycling explicit information from Krylov subspaces can be found in [42], [48].

In this paper we directly generalize the approximate diagonal updates which are useful for solving the parabolic PDEs proposed in [3], see also [9]. This generalization consists in modifying general offdiagonal entries. Our numerical experiments show that the generalizations are competitive with recomputing the factorized nonsymmetric preconditioners in terms of achieving similar convergence rates for subsequent systems. Moreover, forming the updates can be significantly cheaper than recomputing the preconditioner. As far as we know, there are no theoretical or experimental results in this direction. We give a couple of theoretical explanations for the good performance of the updates and discuss some unexpected effects which help to improve the convergence, and, as far as we know, have not been communicated before.

The strategy which we use forms the updated preconditioner from two separate layers: entries of the original factorized preconditioner and scaled entries of the matrix update. For the sake of quality and efficiency we typically need to exploit only a part of the update. This part may result from a Gauss-Seidel type of splitting, or it may be found in a more sophisticated way. In this paper we treat both cases.

The paper is organized as follows. In Section 2 we present a brief introduction into preconditioner updates and motivate the basic form of our updated factorizations. In Section 3 we describe the new techniques for approximate updating. The results of numerical experiments with the new algorithms are presented and discussed in Section 4. Directions for current and future research are given in Conclusions. Throughout the paper, $\|\cdot\|$ denotes an arbitrary matrix norm.

2. The ideal updated preconditioner. Some of the strategies to update preconditioners that we mentioned in the introduction are linked with specific classes of linear solvers (e.g. recycling Krylov subspaces) and nonlinear solvers (e.g. Broyden-type methods) or they were designed for symmetric matrices. In this paper we wish to consider sequences of general, nonsymmetric systems that are solved by preconditioned iterative methods. We address here the following problems: First, how can we update, in theory, a preconditioner in such a way that the updated preconditioner is likely to be as powerful as the original one? And second, how can we approximate, in practice, such an update in order to obtain a preconditioner that is inexpensive to apply and yet useful?

In order to simplify the notation, we consider two linear systems of dimension n denoted by $Ax = b$ and $A^+x^+ = b^+$. Denote the difference matrix $A - A^+$ by B and let M be a preconditioner approximating A. Some information about the quality of the preconditioner M can be taken from a norm of the matrix

$$A - M \tag{2.1}$$

or from some norm of the matrix

$$I - M^{-1}A \quad \text{or} \quad I - AM^{-1} \tag{2.2}$$

if we consider preconditioning from the left or right, respectively (see, e.g. [3]). If preconditioners are in factorized form, both (2.1) and (2.2) should be considered in practice since the preconditioners can suffer from two types of deteriorations. While the norm of the matrix (2.1) expresses *accuracy* of the preconditioner, the norms of the matrices (2.2) relate to its *stability* [15], see also [5]. We will define updated preconditioners M^+ for A^+ whose accuracy and stability are close to the accuracy and stability of M for A. For their derivation we concentrate on the norm of the matrix (2.1) because of its simplicity. Later in this section we present theoretical results demonstrating that both accuracy and stability of the derived updates are comparable to or even better than those of M for A.

We immediately obtain

$$\|A - M\| = \|A^+ - (M - B)\|.$$

Hence $M^+ \equiv M - B$ represents an updated preconditioner for A^+ of the same "level" of accuracy as M represents for A. We will call it the *ideal* updated preconditioner. Note that there may very well exist different preconditioners that are ideal with respect to a norm of $A^+ - M^+$. Just consider $M^+ = M - C$ for some matrix $C \neq B$ with

$$\|A - M\| = \|A^+ - M^+\| = \|A^+ - M + C\|.$$

Because B is often readily available, we will concentrate on $M^+ = M - B$.

If we want to use M^+ as a preconditioner, we need to multiply vectors with its inverse in every iteration of the linear solver. In some problems, the difference matrix B is such that $(M - B)^{-1}$ can be obtained from M^{-1} with low costs. For instance if B has small rank, M^+ can be easily inverted using the Sherman-Morrison formula, see e.g. [45, 8]. In general, however, the ideal updated preconditioner cannot be used since multiplication of vectors with $(M - B)^{-1}$ is expensive. Instead, we will consider cheap approximations of $(M - B)^{-1}$.

In this paper we will assume that M is given in the form of a triangular decomposition as $M = LDU \approx A$, where L and U have unit main diagonal. The approximate updates of factorized preconditioners which we will describe below typically assume that the matrices have a strong diagonal. Note that this assumption is very similar to theoretical assumptions which are generally required to get simple incomplete factorizations without a breakdown. For example, standard ILU(0) and AINV preconditioners are proved to be breakdown-free if the system matrix is an H-matrix [44], [6]. In order to extend the breakdown-free property to more general matrices we need to change the decomposition by modifications which make the diagonal stronger, e.g., by a preliminary shift [44], [41], see also [35], [1], or by global modification of the decomposition [57], [37], [4]. The transfer from diagonal dominance of the matrix to diagonal dominance of the factors is discussed, for example, in [7], cf. also [3], or in the practical reordering strategies based on strong transversals [47], [22], [23]. In the following we tacitly assume matrices are given in such form that the factors L and U more or less approximate the identity matrix.

If $M-B$ is invertible, we can approximate its inverse by a product of more factors which are easier to invert. For example, we can replace $(M - B)^{-1}$ by a product of inverses of triangular matrices and by an inverse of a difference of matrices where a diagonal matrix is used instead of M, as in

$$(M - B)^{-1} = U^{-1}(D - L^{-1}BU^{-1})^{-1}L^{-1} \approx U^{-1}(D - B)^{-1}L^{-1}, \tag{2.3}$$

provided $D - B$ is nonsingular. Now assume $\overline{D - B}$ is a nonsingular approximation of $D - B$ that can be inverted inexpensively. Then we can define a preconditioner M^+ via the last expression in (2.3) as

$$M^+ = L(\overline{D - B})U. \tag{2.4}$$

In the symmetric case, this preconditioner changes to $M^+ = L(\overline{D - B})L^T$, hence symmetry is preserved if we choose $\overline{D - B}$ appropriately. Here we are primarily interested in the nonsymmetric case, and in this case we can further simplify the update. For example, we can approximate as

$$(M - B)^{-1} = (DU - L^{-1}B)^{-1}L^{-1} \approx (DU - B)^{-1}L^{-1}, \tag{2.5}$$

if $DU - B$ is nonsingular. If $\overline{DU - B}$ denotes a nonsingular and easily invertible approximation of $DU - B$, then we define M^+ by

$$M^+ = L(\overline{DU - B}). \tag{2.6}$$

In comparison with (2.4), it seems to be much easier to deal only with two factors. An analogue of (2.5) is approximation through

$$(M - B)^{-1} = U^{-1}(LD - BU^{-1})^{-1} \approx U^{-1}(LD - B)^{-1}. \tag{2.7}$$

In our experiments we choose between approximation with (2.5) or (2.7) adaptively (we explain this later on). We describe our theoretical results for the case (2.5) only.

A first question is whether the update (2.6) has the potential to be more powerful than the frozen preconditioner $M = LDU$ for A^+. In the following simple lemma we express the relation of frozen and updated preconditioner quantitatively.

LEMMA 2.1. *Let* $||A - LDU|| = \varepsilon||A|| < ||B||$. *Then the preconditioner from (2.6) satisfies*

$$\begin{aligned}||A^+ - M^+|| &\leq \frac{||L(DU - \overline{DU - B}) - B|| + \varepsilon||A||}{||B|| - \varepsilon||A||} \cdot ||A^+ - LDU|| \\ &\leq \frac{\|L\|\,\|DU - B - \overline{DU - B}\| + ||L - I||\,\|B\| + \varepsilon||A||}{||B|| - \varepsilon||A||} \cdot ||A^+ - LDU||.\end{aligned}$$

Proof. We get directly

$$\begin{aligned}||A^+ - M^+|| &= ||A - B - L(\overline{DU - B})|| = ||(A - LDU) + L(DU - \overline{DU - B}) - B|| \\ &\leq \big(\varepsilon||A|| + ||L(DU - \overline{DU - B}) - B||\big)\,\frac{||B|| - \varepsilon||A||}{||B|| - \varepsilon||A||} \\ &\leq \big(\varepsilon||A|| + ||L(DU - \overline{DU - B}) - B||\big)\,\frac{||(A - LDU) - B||}{||B|| - \varepsilon||A||} \\ &\leq ||A^+ - LDU||\frac{||L(DU - \overline{DU - B}) - B|| + \varepsilon||A||}{||B|| - \varepsilon||A||} \\ &= ||A^+ - LDU||\frac{||L(DU - \overline{DU - B} - B) + (L - I)B|| + \varepsilon||A||}{||B|| - \varepsilon||A||} \\ &\leq ||A^+ - LDU||\frac{\|L\|\,\|DU - B - \overline{DU - B}\| + ||L - I||\,\|B\| + \varepsilon||A||}{||B|| - \varepsilon||A||}.\end{aligned}$$

□

The multipliers of $||A^+ - LDU||$ in Lemma 2.1 can be smaller than one if $\overline{DU - B}$ is close to $DU - B$ and if $||L - I||$ tends to be small. In practice, taking into account preconditioner modifications to improve diagonal dominance, this is often realistic. Note that the assumption $||A - LDU|| = \varepsilon||A|| < ||B||$ is satisfied as soon as we have a strong preconditioner $M = LDU$.

The lemma states, apart from showing a relation to the frozen preconditioner, that for $\varepsilon||A||$ small enough a good approximation to $\overline{DU - B}$ combined with a close to diagonal factor L yields an accurate preconditioner which *may* be as powerful as a recomputed preconditioner. If we have a recomputed preconditioner M^R with say $\|A^+ - M^R\| = \delta = \|A - M\|$, then based on (2.5) we expect $\|A^+ - M^+\| \geq \delta$. But the previous lemma shows $\|A^+ - M^+\| < \delta$ is not at all excluded. In Section 4 we will show experimentally that the update (2.6) in some cases gives a higher convergence rate than if the preconditioner is recomputed.

The following theorem shows in a different way that, under the given assumptions, the quality of the update may be better than that of recomputed preconditioners if the approximation $\overline{DU - B}$ is favorably chosen. Since Lemma 2.1 is related to the accuracy according to (2.1), the theorem considers its quality with respect to (2.2). The result is a straightforward generalization of a result from [9]. To simplify the description, the scaled updated approximate factor $D^{-1}(\overline{DU - B})$ will be denoted by $\overline{U - D^{-1}B}$, and $U^{-1}(\overline{U - D^{-1}B})$ will be denoted by $I - \overline{U^{-1}D^{-1}B}$.

THEOREM 2.2. *Assume that $LDU + E = A$ for some error matrix E and let $\|\overline{U^{-1}D^{-1}B}\|_2 \leq 1/c < 1$ where $\|\cdot\|_2$ denotes the Euclidean norm. Further assume that the singular values σ_i of*

$$(I - L)B + L\left(\overline{DU - B} - (DU + L^{-1}E - B)\right)$$

satisfy

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_t \geq \delta \geq \sigma_{t+1} \geq \ldots \geq \sigma_n.$$

for some integer t, $t \ll n$, and some small $\delta > 0$. Let $(\overline{DU - B})$ have nonzero main diagonal, and $D = \mathrm{diag}(d_1, \ldots, d_n)$. Then there exist matrices F and Δ such that

$$(\overline{DU - B})^{-1}L^{-1}A^+ = I + \Delta + F, \tag{2.8}$$

with rank$(\Delta) \leq t$ and

$$\|F\|_2 \leq \frac{c}{c-1} \max_i \frac{\delta}{|d_i|} \|L^{-1}\|_2 \|U^{-1}\|_2.$$

Proof. We have

$$\begin{aligned} L(\overline{DU - B}) - A^+ &= L(DU + L^{-1}E - B + \overline{DU - B} - (DU + L^{-1}E - B)) - A^+ \\ &= (I - L)B + L\left(\overline{DU - B} - (DU + L^{-1}E - B)\right). \end{aligned}$$

By assumption, the SVD of the latter matrix can be written as

$$(I - L)B + L\left(\overline{DU - B} - (DU + L^{-1}E - B)\right) = W\Sigma V^T =$$
$$W\mathrm{diag}(\sigma_1, \ldots, \sigma_t, 0, \ldots, 0)V^T + W\mathrm{diag}(0, \ldots, 0, \sigma_{t+1}, \ldots, \sigma_n)V^T \equiv \Delta_1 + F_1,$$

where $\mathrm{rank}(\Delta_1) \leq t$ and $\|F_1\|_2 \leq \delta$. Hence

$$L(\overline{DU - B}) - A^+ = \Delta_1 + F_1$$

and

$$(\overline{DU - B})^{-1}L^{-1}A^+ = I - (\overline{DU - B})^{-1}L^{-1}\Delta_1 - (\overline{DU - B})^{-1}L^{-1}F_1.$$

By setting

$$F \equiv -(\overline{DU - B})^{-1}L^{-1}F_1, \qquad \Delta \equiv -(\overline{DU - B})^{-1}L^{-1}\Delta_1,$$

we get (2.8), where $\mathrm{rank}(\Delta) \leq t$. The matrix F can be bounded by

$$\|F\|_2 \leq \|L^{-1}\|_2 \left\| \left(D(\overline{U - D^{-1}B}) \right)^{-1} \right\|_2 \delta,$$

hence

$$\begin{aligned} \|F\|_2 &\leq \max_i \frac{\delta}{|d_i|} \|L^{-1}\|_2 \|(\overline{U - D^{-1}B})^{-1}\|_2 \\ &\leq \max_i \frac{\delta}{|d_i|} \|L^{-1}\|_2 \|U^{-1}\|_2 \|(I - \overline{U^{-1}D^{-1}B})^{-1}\|_2. \end{aligned}$$

By assumption, $\|\overline{U^{-1}D^{-1}B}\|_2 \leq 1/c < 1$, and consequently

$$\begin{aligned}\|F\|_2 &\leq \max_i \frac{\delta}{|d_i|}\|L^{-1}\|_2\|U^{-1}\|_2 \left(1 - \|\overline{U^{-1}D^{-1}B}\|_2\right)^{-1} \\ &\leq \frac{c}{c-1}\max_i \frac{\delta}{|d_i|}\|L^{-1}\|_2\|U^{-1}\|_2.\end{aligned}$$

□

Note that if the matrix F in (2.8) is zero, then the preconditioned system is a rank t update of the identity and Krylov subspace methods converge, in exact arithmetics, in at most $t+1$ iterations.

In the following section we propose approximations $\overline{DU-B}$ of $DU-B$ that can be efficiently computed and that lead to preconditioners that are inexpensive to apply. All techniques we present can be analogously formulated for updates of the form $(\overline{LD-B})U$ corresponding to (2.7).

3. Approximate preconditioner updates. We propose the following strategies to approximate $DU-B$ by an easily invertible matrix $\overline{(DU-B)}$. A first obvious but effective strategy is to set $\overline{DU-B} \equiv triu(DU-B)$, where $triu$ denotes the possibly sparsified upper triangular part (including the main diagonal). This results in the preconditioner

$$M^+ = L(DU - triu(B)), \tag{3.1}$$

which can be obtained entirely for free. The additional cost for applying this preconditioner is one triangular sweep with the triangular part of B, if we store B and U separately. We may also merge them; then the additional sweep can be virtually for free if the sparsity patterns of $triu(B)$ and U are close enough. We will call the update constructed by considering entries from one triangular part only the *structured update*. A trivial structured sparsification is given by

$$\overline{DU-B} \equiv \mathrm{diag}(DU-B),$$

which is a straightforward application of an approach from [3] to nonsymmetric problems.

As we show in the experiments, the simple update (3.1) and its analogue

$$M^+ = (LD - tril(B))U \tag{3.2}$$

seem to be powerful in many problems. One expects them to be particularly suited when one triangular part of B clearly dominates the other. The typical situation of that kind arises when matrices come from upwind/downwind discretization schemes. Nevertheless, as they take into account only one triangular part of the difference matrix B, there may be applications where important information is lost, leading to weak convergence. In the following we present a technique to replace $DU-B$ by an easily invertible matrix which is in general not triangular.

Denote the matrices $\mathrm{diag}(\overline{DU-B})$ by $\tilde{D}$, and $\tilde{D}^{-1}(\tilde{D} - \overline{DU-B})$ by $\tilde{B}$, respectively. Then $\tilde{B}$ has zero diagonal and we can write

$$\overline{DU-B} = \tilde{D}(I-\tilde{B}). \tag{3.3}$$

To motivate the scaling transformation in (3.3) consider for a moment the case when $\tilde{B} = \beta e_i e_j^T$, for some $1 \leq i,j \leq n, i \neq j$, and recall we assume $\overline{DU-B}$ is nonsingular,

hence so is $I - \tilde{B}$. Then we get, with the Sherman-Morisson formula,

$$(I - \tilde{B})^{-1} = I + \beta e_i e_j^T / (1 - \beta e_j^T e_i) = I + \beta e_i e_j^T = I + \tilde{B}. \tag{3.4}$$

The matrix in (3.4) is equal to the identity modified by an off-diagonal entry β at the position (i, j). That is, $(I - \tilde{B})$ is a special Gauss-Jordan transformation [29], it is inverted without costs and it has a fill-in free inverse.

Based on this well-known fact, in the following we will try to find *unstructured* approximations $\overline{DU - B}$ of $DU - B$ such that the scaled matrix $I - \tilde{B}$ can be written as a product of Gauss-Jordan transformations

$$(I - e_{i_1}\tilde{b}_{i_1 *})(I - e_{i_2}\tilde{b}_{i_2 *}) \dots (I - e_{i_K}\tilde{b}_{i_K *}), \qquad K \leq n - 1, \tag{3.5}$$

where $\tilde{B} = (\tilde{b})_{ij}$. Denote the sparsity structure of a row i of $\tilde{B}$ (with zero diagonal) by $row(i)$, that is, $row(i) = \{k | i \neq k \wedge \tilde{b}_{ik} \neq 0\}$. The multiplication $(I - \tilde{B})^{-1} v$ for a given vector v is very cheap, as stated in Observation 3.1.

Observation 3.1. *The number of operations for multiplying a vector by a matrix of the form (3.5) or its inverse is at most* $2 \sum_{j=1}^{K} |row(i_j)|$.

It is well known that any unit upper triangular matrix $I - \tilde{B}$ from (3.3) can be trivially written as the product $R_{n-1} \dots R_1$ of $n - 1$ elementary triangular matrices $R_i = I - e_i \tilde{b}_{i *}$ for $i = 1, \dots, n - 1$. Hence using (3.1) may be considered a special case of (3.5). The following theorem shows a necessary and sufficient condition for the existence of a decomposition of $I - \tilde{B}$ of the form (3.5).

Theorem 3.1. *Let* $I - \tilde{B} = I - \sum_{j_l : l = 1, \dots, K} e_{j_l} \tilde{b}_{j_l *}$. *Then*

$$I - \tilde{B} = (I - e_{i_1}\tilde{b}_{i_1 *})(I - e_{i_2}\tilde{b}_{i_2 *}) \dots (I - e_{i_K}\tilde{b}_{i_K *}) \tag{3.6}$$

if and only if

$$i_l \notin \bigcup_{k=1}^{l-1} row(i_k) \text{ for } 2 \leq l \leq K \tag{3.7}$$

for all $i_1, \dots, i_K$ *such that* $\{j_1, \dots, j_K\} = \{i_1, \dots, i_K\}$.

Proof. The equivalence of (3.6) and (3.7) follows from the orthogonality of the unit vector e_{i_l} with respect to all $\tilde{b}_{i_k *}$ for $k < l$, $1 \leq l \leq K$. □

Based on Theorem 3.1 we first propose a greedy procedure to find a suitable approximation $\overline{DU - B}$ with $I - \tilde{B}$ satisfying (3.6). Consider a sequential choice of indices $i_1, \dots, i_K$, where $K \leq n - 1$ are determined by the algorithm. In each step we keep and update a set of *candidate rows* $\mathcal{R}$ initialized by $\{1, \dots, n\}$. After choosing a row i we remove from $\mathcal{R}$ all the rows $j \in \mathcal{R}$ for which $\tilde{b}_{ij} \neq 0$.

Algorithm 3.1. *Algorithm to approximate* $DU - B$ *by a matrix which, scaled by its diagonal, can be written in the form (3.6).*

(1) *set* $\mathcal{R} = \{1, \dots, n\}$, $K = 0$
(2) *for* $k = 1, \dots, n$ *do*
(3) set $row(k) = \{i | i \neq k \wedge |(DU - B)_{ki}| \neq 0\}$
(4) set $p_k = \sum_{j \in row(k)} |(DU - B)_{kj}|$
(5) *end for*
(6) *while* $\mathcal{R} \neq 0$ *do*

(7) *choose a row* $i \in \mathcal{R}$ *maximizing* $p_i - \sum_{j \in \mathcal{R} \cap row(i)} p_j$
(8) *set* $K = K + 1$, $i_K = i$, $\mathcal{R} = \mathcal{R} \backslash \{row(i_K) \cup i\}$
(9) *end while*

The row indices $i_1, \ldots, i_K$ provided by Algorithm 3.1 then determine the approximation in (3.3) with $I - \tilde{B}$ equal to the product (3.5). The heuristic criterion in step (7) aims on the one hand to choose the row of $DU - B$ with largest entries. On the other hand it stimulates the choice of a row which results, based on condition (3.7), in removal of candidate rows with small entries. To balance between the two heuristics one may want to introduce a weighting parameter ω and use

$(7')$ *choose a row* $i \in \mathcal{R}$ *maximizing* $p_i - \omega \cdot \sum_{j \in \mathcal{R} \cap row(i)} p_j$.

Clearly, the algorithm may find more factors of (3.6) if there are less nonzero entries in the searched rows. Therefore it may be reasonable to perform some dropping strategy on-the-fly when running the algorithm by substituting step *(3)* with

$(3')$ *set* $row(k) = \{i | i \neq k \wedge |(DU - B)_{ki}| > tol\}$

for a predefined drop-tolerance *tol*. Apart from tolerance-based dropping, sparsification based on the given mask may enhance the effectiveness of our strategy. Note that sparsification not only helps in covering as much rows as possible by Gauss-Jordan transformations, but it also leads to less expensive matvecs with the inverse of (3.5).

A more elegant and systematic way to get an unstructured update based on Gauss-Jordan transformations can be described by the following bipartite graph model. Let us define the bipartite graph of $(DU - B)$ as $G(DU - B) = (R, C, E)$, where $R = \{1, \ldots, n\}$, $C = \{1', \ldots, n'\}$ and $E = \{(i, j') | (DU - B)_{ij} \neq 0\}$. Then we have the following result.

THEOREM 3.2. *Consider a spanning forest* $T = (V_T, E_T)$ *of* $G(DU - B)$ *such that* $\{(i, i') | 1 \leq i \leq n\} \subseteq E_T$. *Then the matrix* $\overline{DU - B} \in \mathbb{R}^{n \times n}$ *with the entries defined by*

$$\overline{(DU - B)}_{ij} = \begin{cases} (DU - B)_{ij} & \text{if } (i, j') \in E_T \\ 0 & \text{otherwise} \end{cases},$$

scaled by its diagonal entries as in (3.3), can be expressed as a product of the form (3.5).

Proof. First consider the case when the spanning forest T is not connected. Components of T induce a block diagonal splitting of $\overline{DU - B}$, and matrices corresponding to individual blocks can be mutually multiplied in any order without causing any fill-in. Consequently, we can assume without loss of generality that T is connected and that T is a spanning tree. In the following we will show how to form the sequence of Gauss-Jordan transformations from the left to the right.

Our assumption implies that T contains at most $n - 1$ edges (i, j') with $i \neq j$. There exists a free row vertex $i \in R$ in T which is in T incident only to the edge (i, i') such that there is an edge $(k, i') \in E_T$ for some k. Set $i_1 = i$. Then remove from T the vertices $i \in R$, $i' \in C$ and all edges incident to them. Clearly, the updated tree T contains a free row vertex again. By repeating the choice of free row vertices and updates T in this way we get the sequence $i_1, \ldots, i_{n-1}$. If we rewrite as $I - \tilde{B}$ the matrix $\overline{DU - B}$ scaled by its diagonal we have $I - \tilde{B} = (I - e_{i_1} \tilde{b}_{i_1 *})(I - e_{i_2} \tilde{b}_{i_2 *}) \ldots (I - e_{i_{n-1}} \tilde{b}_{i_{n-1} *})$ which proves the theorem. □

Theorem 3.2 implies the following algorithmic strategy to find a matrix $\overline{DU-B}$ which would approximate $DU-B$ and could be expressed as a product of Gauss-Jordan transformations.

ALGORITHM 3.2. *Algorithm to find $\overline{DU-B}$ such that (3.6) is satisfied based on a bipartite graph of $DU-B$.*

(1) Find a spanning forest $T=(V_T,E_T)$ of $G(DU-B)$ of maximum weight with edge weights $w_{ij}=|(DU-B)_{ij}|$ for $(i,j')\in E_T$ such that $\{(i,i')|1\le i\le n\}\subseteq E_T$.

(2) Find the entries of $\tilde{B}$ (and corresponding entries of $\overline{DU-B}$) as well as a feasible ordering of Gauss-Jordan factors for $i_1,\dots,i_{n-1}$ in (3.5) with Theorem 3.2.

(3) For each $k=2,\dots,n$ add to $\overline{DU-B}$ all entries $(DU-B)_{i_kl}$ of $DU-B$ such that $l\in\{i_1,\dots,i_{k-1}\}$.

Note that in the last step of Algorithm 3.2 we possibly put into $\overline{DU-B}$ much more nonzero entries than the $2n-1$ entries provided by the weighted spanning forest. This is possible because of Theorem 3.1. The complexity of the weighted minimum spanning forest (here we need, in fact, a weighted maximum forest) is $O(m\log m)$ for the Kruskal algorithm [31] and $O(n+m\log m)$ for the Prim algorithm [50], where m is the number of edges in the graph G. Note, in addition, that we start with the partial spanning tree with the set of edges $\{(i,i')|1\le i\le n\}$. While in some cases the algorithms may seem time consuming, this procedure can provide useful updates. As in Algorithm 3.1, we can sparsify $DU-B$ by discarding entries smaller than a certain drop tolerance tol, which reduces the value of m and therefore also computational complexity.

From Lemma 2.1 it is clear that the quality of the approximation of $DU-B$ may play a decisive role in the power of the preconditioner $M^+=L(\overline{DU-B})$. In practice, the way that the original incomplete decomposition is constructed (scaling L during the construction, pivoting) can strongly support the quality of $\overline{DU-B}$. In order to use the most powerful type of update, in our experiments we switch adaptively between (3.1) and (3.2) based on the weighting of both triangular parts of B and use an unstructured update based on Algorithm 3.1 or 3.2 if its weighting is the most important. More precisely, we compute sums of magnitudes of entries in the triangular parts of the matrices and simulate runs of Algorithm 3.1 and Algorithm 3.2 to get the sum of magnitudes of entries covered by the unstructured update. We then use the strategy which corresponds to the maximum value among these sums.

It can happen and it often happens that in spite of the fact that the updated preconditioner loses some information about the system matrix it yields a better convergence rate than if the preconditioner would be recomputed from the scratch. There are several possible explanations for this phenomenon. First, note that we showed theoretically in Lemma 2.1 and Theorem 2.2 that our updated preconditioners have the potential to be stronger than recomputed factorizations. In practice, it frequently happens that by updating the preconditioner we relate it to a previous decomposition which is more diagonally dominant than a recomputed decomposition. A part of the stable triangular factors is inherited and the update may even stabilize less stable factors of the initial factorization. Note that a modified old decomposition might be useful in general, but, e.g. in the related strategy [44], the size of the modification should be typically rather small to get a useful preconditioned iterative method. This is exactly what happens when modifying with entries of difference matrices B that are typically small compared to those of $A^{(i)}$. In addition, updates appear to perform better also in cases where there is no instability. We presume this is so because the preconditioner may be favorably modified by the additional structural information

given by the update. To our knowledge, this conjecture is stated for the first time. An overlooked fact is that the most powerful dual-threshold incomplete decompositions and inverse decompositions can be very memory-efficient, but they may discard the structure of the problem. Our updates can add to a memory-efficient decomposition cheap and useful information about the structure as seems to be clear from our experiments. We believe that such a strategy might be used to improve constructing general preconditioners in some cases. We might consider the update as a simple and efficient way to modify off-diagonal entries of the preconditioner, getting thus a generalization of diagonal modifications from [44] or forced diagonal modifications introduced in [35]. It is not unusual that level-based incomplete decompositions are much better than their sophisticated counterparts. Such a behavior has been observed on some VENKAT matrices from the Harwell-Boeing collection, where powerful and compact ILUT [52] preconditioners are less efficient than often very dense but reasonably structured ILU preconditioners using the concept of levels [61], [30].

The next section is devoted to numerical experiments with the most promising updates introduced in the paper.

4. Numerical experiments. In this section we present results of numerical experiments with preconditioned Krylov subspace methods for solving sequences of systems of linear algebraic equations, where updated preconditioners are compared with recomputed and frozen preconditioners. We consider the sequences in three application problems. The first and second problem were generated with the optimization software UFO [43]. The last application is based on [10]. Software for the problem was kindly provided by Philipp Birken. We present results with several kinds of incomplete LU-preconditioners to show that the introduced techniques are quite general. In order to show a larger spectrum of various results, some of the computations were done in Matlab using its incomplete ILU decomposition script. We used Matlab version 7.0. Most of the tests, in particular for larger problems, were performed with preconditioners and the updates written in Fortran 90, and compiled by Compaq Visual Fortran 6.6a. The codes were run on a computer with Intel Pentium 4, 3GHz processor, 1GB RAM memory, 512k L2 cache.

As an accelerator, the BiCGSTAB [58] iterative method with right preconditioning was used. We also performed some experiments with the restarted GMRES [53] method and the transpose-free QMR [26] method. The results were similar and we do not report on them here. Iterations were stopped when the Euclidean norm of the residual was decreased by seven orders of magnitude. Nevertheless, in our experiments we observed close to linear behavior of convergence curves of the preconditioned iterative method. Therefore, we expect qualitatively the same results for weaker or nonuniform stopping criteria used in nonlinear solvers.

Our first test problem is a two-dimensional nonlinear convection-diffusion model problem which we use to illustrate various aspects of the proposed strategies (general behavior of the strategies, choice of parameters, values of the bounds in Lemma 2.1). It has the form (see, e.g. [33])

$$-\Delta u + Ru\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) = 2000x(1-x)y(1-y), \tag{4.1}$$

on the unit square, discretized by 5-point finite differences on a uniform 70×70 grid. The initial approximation is the discretization of $u_0(x, y) = 0$. We choose the modest Reynolds number $R = 50$ in order to avoid potential discretization problems which

may ask for adding stabilization terms. We obtain a small sequence of 7 matrices with 24220 nonzeros each (in the tables we denote the number of nonzeros by nnz).

Our update techniques are particularly beneficial when recomputing preconditioners is expensive. We start with a typical example given by the so-called ILU(0) incomplete decomposition which has the same sparsity pattern as the matrix it preconditions. This has the obvious advantage that it enables straightforward a priori allocation but its computation may be time-consuming. In Table 1 we display the total time to solve the whole sequence and the numbers of BiCGSTAB iterations needed to solve the individual linear systems for several preconditioning strategies. In the first of them, denoted by 'Recomp', the ILU(0) preconditioner was computed for each matrix separately. The strategy 'Freeze' used a fixed preconditioner. The strategy denoted by 'Str' used structured updates, 'Unstr. GJ' stays for unstructured updates based on Gauss-Jordan transformations obtained from Algorithm 3.1 and 'Unstr. Kr.' for those obtained from Algorithm 3.2, where the spanning tree is computed with Kruskal algorithm. We see that the recomputed ILU(0)-decompositions yield powerful preconditioners for our problem but they are rather slowly computed in Matlab. Freezing the initial ILU(0)-decomposition avoids these slow computations and although it yields much higher numbers of BiCGSTAB iterations, the overall time to solve the sequence is shorter. Excellent behavior of the structured updates is demonstrated by this table. Here the triangular parts were chosen adaptively based on the magnitudes of their entries. While iteration numbers are nearly as low as with recomputation, significant time savings are achieved by avoiding the recomputation of preconditioners. The iteration counts for unstructured updates from Algorithm 3.1 are a little higher than for structured updates but they are clearly lower than with the frozen preconditioner. Unstructured updates from Algorithm 3.2 yield iteration numbers comparable to those of structured updates.

Of course, running Algorithm 3.1 or Algorithm 3.2 to compute the unstructured updates adds a time penalty. However, the timings displayed in Table 1 are pessimistic because they include solution with non-triangular factors of the form (3.5), which cannot compete with the highly optimized implementation of back- and forward solves in Matlab. The complexity of Algorithm 3.1 or Algorithm 3.2 alone is not very high for sparse matrices since it is linear in the number of matrix nonzeros. In this context, note that using a drop tolerance in Algorithm 3.1 and Algorithm 3.2 has an influence on the number of nonzeros and hence also on computational time. We computed the unstructured updates with $tol = 0.3$ in Algorithm 3.1 and Algorithm 3.2. In practice this parameter should be chosen according to the following considerations for the individual algorithms.

In Algorithm 3.2 we first construct a maximum spanning forrest of at most $2n-1$ entries. Hence we need a value of tol selecting the $2n-1$ largest entries and as few other entries as necessary to be able to build the spanning forrest. We could have optimized the choice of tol according to this rationale, leading to $tol = 0.35$ and an overall time of 10.5 seconds. For Algorithm 3.1 the situation is quite different. Here, an interesting fact is that if we significantly overestimate the parameter then the unstructured update may be very sparse since a smaller number of nonzeros can be covered by Gauss-Jordan transforms. If we underestimate it then the update may be very sparse as well since we get only a small number of factors in the unstructured update of the form (3.1). In our case we did not optimize its choice but a value $tol = 0.1 - 0.4$ for a reasonably scaled system matrix in order to keep only a few, say up to k nonzeros in a row, and thus to cover by the unstructured update approximately

TABLE 1
Nonlinear convection-diffusion model problem with R=50, n=4900, nnz=24220, ILU(0).

ILU(0), psize ≈ 24000					
Matrix	Recomp	Freeze	Str.	Unstr. GJ	Unstr. Kr.
$A^{(0)}$	40	40	40	40	40
$A^{(1)}$	29	36	32	39	30
$A^{(2)}$	21	39	27	34	30
$A^{(3)}$	20	48	26	33	24
$A^{(4)}$	17	55	26	31	26
$A^{(5)}$	16	58	29	29	30
$A^{(6)}$	15	50	22	24	26
$A^{(7)}$	15	62	26	28	29
$A^{(8)}$	17	68	28	30	31
$A^{(9)}$	15	71	27	28	28
$A^{(10)}$	15	51	24	29	28
overall time	11 s	7.5 s	5 s	8.5 s	12.5 s

$k \cdot n$ offdiagonal entries, is fine. This type of behavior is different from what we can sometimes observe in the field of algebraic preconditioners. As for the choice of ω in Algorithm 3.1, its value does not seem to have a crucial influence on the performance of the update either. In Figure 4.1 we display the total number of BiCGSTAB iterations needed to solve the whole sequence for different values of ω. Only for values smaller than 0.5, criterion $(7')$ of Algorithm 3.1 starts to overemphasize the weight of the chosen candidate row, resulting in bad approximations of $DU - B$. In the other experiments presented here, we always used the choice $\omega = 1$.

In Table 2 the accuracies $\|A^{(i)} - M^+\|$ (in the Frobenius norm) of the preconditioners M^+ for the individual strategies are displayed. For this sequence, where stability of the preconditioners is not an issue, the accuracies nicely correspond to the numbers of BiCGSTAB iterations. We also present some information about the quality of the approximations $\overline{DU - B}$. Table 3 contains the values of the approximations $\|DU - B - \overline{DU - B}\|$ in the Frobenius norm for the considered update techniques. Also these values correspond to the numbers of BiCGSTAB iterations.

In Table 4 we take a closer look at the various update techniques we introduced. Whereas Table 1 suggests that structured updates provide more efficient preconditioners than unstructured updates, this is not apparent from Table 4. Here we use as initial preconditioner the ILU-factorization implemented in Matlab with drop tolerance 0.01. The tolerance in Algorithms 3.1 and 3.2 for unstructured updates is 0.3. Clearly, unstructured updates are more powerful than structured updates with this kind of initial factorization. This is caused by the fact that the approximations $\overline{DU - B}$ in (2.6) cover more large entries when we use unstructured updates. In the following we quantify this property for a difference matrix B from the middle of the sequence, $B = A^{(0)} - A^{(4)}$. For other difference matrices from the sequence we would obtain similar numbers. With $B = A^{(0)} - A^{(4)}$, nonzero entries in $DU - B$ are quite evenly distributed over both triangular parts. We have $\|striu(DU - B)\| \approx 80$ and $\|stril(DU - B)\| \approx 38$ in the Frobenius norm. Here $stril(\cdot)$ and $striu(\cdot)$ denotes the strict lower and upper triangular matrix part, respectively. Hence the upper triangular part is dominating, but important entries may be found in the lower part too and they

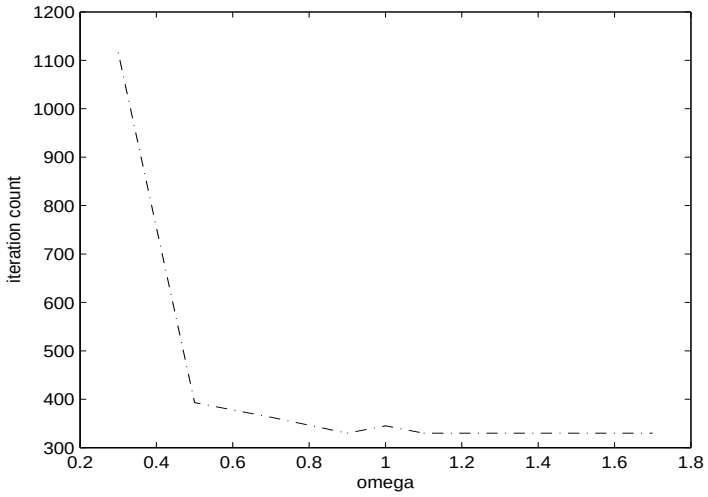

FIG. 4.1. *Nonlinear convection-diffusion model problem: Iteration counts for Unstr. GJ in dependency of* ω.

TABLE 2
Nonlinear convection-diffusion model problem, accuracies $\|A^{(i)} - M^+\|$

ILU(0), psize $\approx$ 24000					
Matrix	Recomp	Freeze	Str.	Unstr. GJ	Unstr. Kr.
$A^{(0)}$	28.5	28.5	28.5	28.5	28.5
$A^{(1)}$	27.8	34.6	29.2	50.2	37.3
$A^{(2)}$	26.8	42.3	41.7	51.0	42.1
$A^{(3)}$	25.5	51.0	48.5	55.8	48.9
$A^{(4)}$	24.1	60.4	55.8	64.0	56.5
$A^{(5)}$	23.6	63.5	58.3	63.9	59.1
$A^{(6)}$	23.1	66.6	60.6	64.9	61.6
$A^{(7)}$	23.1	66.6	60.6	64.9	61.5
$A^{(8)}$	23.1	66.5	60.6	64.9	61.5
$A^{(9)}$	23.1	66.5	60.6	64.9	61.5
$A^{(10)}$	23.1	66.5	60.6	64.9	61.5

are lost with structured updates. The unstructured updates take into account both triangular parts. This is reflected by the Frobenius norms $\|striu(\overline{DU - B})\| \approx 70$ and $\|stril(\overline{DU - B})\| \approx 16$ for the approximation $\overline{DU - B}$ from Algorithm 3.1. With Algorithm 3.2 we obtain $\|striu(\overline{DU - B})\| \approx 58$ and $\|stril(\overline{DU - B})\| \approx 32$. Note that Algorithm 3.2 yields more nonzeros, which is explained by step (3) of the algorithm.

TABLE 3
Nonlinear convection-diffusion model problem, approximation qualities $\|DU - B - \overline{DU - B}\|$.

ILU(0), psize ≈ 632000			
Matrix	Str.	Unstr. GJ	Unstr. Kr.
$A^{(1)}$	13.89	37.01	18.19
$A^{(2)}$	22.1	36	22.7
$A^{(3)}$	29.78	40.1	30.34
$A^{(4)}$	37.46	48.32	38.47
$A^{(5)}$	39.92	47.39	41.2
$A^{(6)}$	42.29	47.91	43.69
$A^{(7)}$	42.27	47.9	43.66
$A^{(8)}$	42.23	47.86	43.62
$A^{(9)}$	42.23	47.86	43.63
$A^{(10)}$	42.23	47.86	43.63

TABLE 4
Nonlinear convection-diffusion model problem with R=50, n=4900, nnz=24220, $ILU(0.01)$.

ILU(10^{-2}), psize ≈ 52000				
Matrix	Freeze	Unstr GJ	Struct	Unstr. Kr.
$A^{(0)}$	17	17	17	17
$A^{(1)}$	34	57	21	48
$A^{(2)}$	49	43	24	36
$A^{(3)}$	77	39	34	33
$A^{(4)}$	102	36	54	29
$A^{(5)}$	140	37	69	28
$A^{(6)}$	142	30	76	25
$A^{(7)}$	154	35	77	28
$A^{(8)}$	144	36	91	33
$A^{(9)}$	152	35	91	29
$A^{(10)}$	123	31	90	28
overall time	14.5 s	7.5 s	9 s	9 s

In this context, also note that the number of nonzeros of the initial factorization and structured updates is about 52000 whereas unstructured updates have smaller numbers of nonzeros, about 39000–46000, which makes application of the unstructured updated preconditioner less expensive. This is one of the reasons why the unstructured updates are competitive even with respect to timing with structured ones, in spite of the time penalty to run Algorithm 3.1 or 3.2. The other reason is, of course, lower BiCGSTAB iteration numbers.

In situations as in Table 1, recomputing preconditioners is outperformed by our updates because of the high expenses of recomputing. When on the other hand, recomputation is straightforward, updates need not be more effective. An example is given in Table 5 with as initial preconditioner the dual-threshold ILUT(0.1,5) decomposition, implemented in Fortran 90. The number of nonzeros in the incomplete LU decomposition is about 38000 (slightly differing for different matrices). Here the time spent for recomputation is very small due to the simple discretization stencil

TABLE 5
Nonlinear convection-diffusion model problem with R=50, n=4900, nnz=24220, ILUT$(0.1, 5)$.

ILUT(0.1,5), timep$\approx$ 0.01, psize $\approx$ 38000			
Matrix	Recomp	Freeze	Struct. update
$A^{(0)}$	25	25	25
$A^{(1)}$	25	33	26
$A^{(2)}$	23	47	27
$A^{(3)}$	19	58	27
$A^{(4)}$	18	83	27
$A^{(5)}$	17	88	28
$A^{(6)}$	16	119	28
$A^{(7)}$	16	114	27
$A^{(8)}$	17	107	27
$A^{(9)}$	17	111	28
$A^{(10)}$	17	123	27
overall time	0.20 s	0.78 s	0.25 s

and by far the most time is spent while solving with BiCGSTAB. Still, concerning iteration counts, the (adaptively chosen) structured updates perform only little worse than recomputation. Note that there is a strong overlap between the location of the nonzeros in B and in the preconditioner, but as above, we did not merge the triangular parts of the updated preconditioner. Table 6 shows similar behavior for a much larger problem with ILUT(0.1,3) as initial decomposition. Here we discretized (4.1) on a 282×282 grid, the matrices have dimension 79524. While evaluating Tables 5 and 6, it is important to realize that the timings may provide here only partial information. In case of matrix-free implementation we typically need to estimate the matrices first using, for example, graph coloring techniques [18], [27]. Our matrices have five diagonals and this implies that they can be estimated by at most 7 matvecs. Namely, the number of matvecs corresponds to the number of colors needed to color the undirected graph of $A^T A$, the so-called intersection graph. Computing some of the standard preconditioners both *directly and efficiently* based on matvecs is a state-of-the-art challenging problem and can be very time-consuming. When using updates in matrix-free environment, only part of the difference matrix needs to be estimated. In our cases the needed part of the difference matrix was always available from at most three matvecs, because the intersection graph of the (possibly permuted) triangular part of the matrix could be colored by only three colors.

In addition to the experiments presented here we performed also some experiments where the nonlinear problems were discretized by upwind schemes, leading to triangular difference matrices. As one can guess from the pattern, the results for solving the linear problems were rather good, but we typically needed more nonlinear iterations. Consequently, discretization by central differences was preferable.

Our second test problem is a smaller but rather difficult problem of dimension 2500. It consists of the two dimensional driven cavity problem of the form

$$\Delta\Delta u + R\left(\frac{\partial u}{\partial y}\frac{\partial \Delta u}{\partial x} - \frac{\partial u}{\partial x}\frac{\partial \Delta u}{\partial y}\right) = 0,$$

on the unit square, discretized by 13-point finite differences on a shifted uniform grid with 50×50 inner nodes [32]. The boundary conditions are given by $u = 0$ on $\partial\Omega$ and

TABLE 6
Nonlinear convection-diffusion model problem with R=50, n=79524, nnz=615997, ILUT$(0.1, 3)$.

ILUT$(0.1, 3)$, timep≈ 0.05, psize ≈ 632000			
Matrix	Recomp	Freeze	Struct. update
$A^{(0)}$	82	82	82
$A^{(1)}$	86	85	82
$A^{(2)}$	73	97	82
$A^{(3)}$	72	91	76
$A^{(4)}$	66	97	73
$A^{(5)}$	68	113	77
$A^{(6)}$	71	140	75
$A^{(7)}$	68	139	70
$A^{(8)}$	70	137	76
$A^{(9)}$	69	136	83
$A^{(10)}$	65	217	72
overall time	17.4 s	31.0 s	19.4 s

TABLE 7
Driven cavity problem with R=50, n=2500, nnz=31504, ILU(0.01).

ILU(0.01), psize ≈ 47000					
Matrix	Recomp	Freeze	Str.	Unstr. GJ	Unstr. Kr.
$A^{(0)}$	93	93	93	93	93
$A^{(1)}$	269	93	88	337	81
$A^{(2)}$	> 500	> 500	156	324	58
$A^{(3)}$	> 500	164	179	265	60
$A^{(4)}$	> 500	288	298	206	74
$A^{(5)}$	> 500	> 500	144	184	71
$A^{(6)}$	> 500	> 500	132	190	70
overall time	∞	∞	8 s	17 s	6.5 s

$\partial u(0, y)/\partial x = 0$, $\partial u(1, y)/\partial x = 0$, $\partial u(x, 0)/\partial x = 0$ and $\partial u(x, 1)/\partial x = 1$. The initial approximation is the discretization of $u_0(x, y) = 0$.

For the same reason as before, we choose modest Reynolds numbers. Even with modest Reynolds numbers we obtain sequences of linear systems that are hard to solve for the BiCGSTAB accelerator. As system matrices have 31504 nonzeros, we needed a relatively dense initial ILU-preconditioner with 47000 nonzeros and with drop tolerance 0.01 from Matlab to be able to solve the linear systems at all. Sparser preconditioners caused BiCGSTAB to stagnate for the initial linear system. In Tables 7 and 9 we show experiments executed in Matlab with the initial ILU(0.01) preconditioner for $R = 50$ and $R = 10$ respectively. As before, with 'overall time' we mean the time needed to solve the whole sequence, including preconditioner computations. In the columns 'Unstr.' we display the performance of unstructured updates computed with Algorithm 3.1 ($tol = 0.05$ for $R = 50$ and $tol = 0.02$ for $R = 10$) and Algorithm 3.2 ($tol = 0.7$ for $R = 50$ and $tol = 0.02$ for $R = 10$).

This problem represents the case where recomputing should be avoided for stability reasons. For instance with $R = 50$, the recomputation of the incomplete factoriza-

TABLE 8
Driven cavity problem with R=50, estimated Euclidean norms of inverses of first factor.

ILU(0.01), psize ≈ 47000					
Matrix	Recomp	Freeze	Str.	Unstr. GJ	Unstr. Kr.
$A^{(0)}$	264	264	264	264	264
$A^{(1)}$	$2 \cdot 10^3$	264	203	1069	185
$A^{(2)}$	$9 \cdot 10^5$	264	227	99	101
$A^{(3)}$	$8 \cdot 10^4$	264	326	291	130
$A^{(4)}$	$3 \cdot 10^5$	264	327	290	131
$A^{(5)}$	$2 \cdot 10^5$	264	327	290	131
$A^{(6)}$	$4 \cdot 10^5$	264	327	290	131

TABLE 9
Driven cavity problem with R=10, n=2500, nnz=31504, ILU(0.01).

ILU(0.01), psize ≈ 47000					
Matrix	Recomp	Freeze	Str.	Unstr. GJ	Unstr. Kr.
$A^{(0)}$	84	84	84	84	84
$A^{(1)}$	84	87	95	91	91
$A^{(2)}$	312	183	119	95	113
$A^{(3)}$	261	198	119	103	134
$A^{(4)}$	352	> 500	190	149	164
$A^{(5)}$	259	> 500	163	204	164
$A^{(6)}$	291	183	150	217	144
overall time	12.5 s	∞	7 s	12 s	11 s

tion failed for the last 5 linear systems (giving the Matlab warning 'Incomplete upper triangular factor had 1 zero diagonal replaced by local drop tolerance'). In order to quantify instability we computed estimates of the 2-norms of the inverses of the factors of the used factorizations. For the initial decomposition we have $\|U^{-1}\|_2 \approx 41$ and $\|(LD)^{-1}\|_2 \approx 264$, but these norms grow rapidly for subsequent recomputed factorizations. In the second column of Table 8 the norms for $(LD)^{-1}$ are displayed, norms for U^{-1} grow similarly. Clearly, forward and backward substitution have become unstable. In the columns corresponding to updated factorizations we estimated $\|(\overline{LD - B})^{-1}\|_2$. We see that higher estimates correspond in the majority of cases to higher iteration numbers. In the frozen preconditioner strategy, however, instability is not the cause of stagnation. We guess the frozen preconditioner fails to provide the structural information contained in updated factorizations. The results for $R = 10$ reflect similar phenomena in a weaker form. Structured and unstructured updates from Algorithm 3.2 yield the best results. In the case $R = 50$ the optimal choice $tol = 0.7$ results in particular good performance of Algorithm 3.2, both with respect to time and iteration count.

We conclude this section with an application which leads to very large sequences of linear systems. They arise from numerical computation of steady vertical air flow through a level tunnel at a low Mach number subject to the gravitational force. The domain is a two dimensional longitudinal section of the tunnel with the pressure and density varying only in the horizontal direction such that the gravitational term is balanced out by the pressure gradient. Neumann boundary conditions and Lax-Friedrichs

fluxes were used. The gravitation term and the Euler equations were separated by a first-order operator splitting. For the discretization, the implicit Euler method combined with the first order finite volume discretization in space were used. In every time step, one Newton step is performed in the flow solver only. More details can be found in [10], in particular in Section 6.2. Our results were very similar for more variations of the problem.

Table 10 contains the results for two sequences from the linear systems for the described problem with a relatively coarse discretization grid. We used the dual-threshold ILUT(0.001,5) preconditioner, where the parameters were chosen in order to have a preconditioner size (that is, number of nonzeros) close to the size of the original matrix and such that the total number of matvecs (two in each iteration) to solve the initial system is reasonably small.

TABLE 10
Air flow in a tunnel, n=4800, nnz=138024, ILUT$(0.001, 5)$.

ILUT$(0.001, 5)$, timep$\approx$0.05, psize $\approx$ 135798						
Matrix	Recomp		Freeze		Update	
	Its	Time	Its	Time	Its	Time
$A^{(5)}$	29	0.57	19	0.33	19	0.34
$A^{(10)}$	30	0.55	17	0.27	17	0.27
$A^{(15)}$	33	0.64	21	0.39	19	0.34
$A^{(20)}$	32	0.64	19	0.34	17	0.31
$A^{(25)}$	33	0.56	20	0.33	19	0.33
$A^{(30)}$	34	0.66	24	0.44	21	0.34
$A^{(35)}$	33	0.66	23	0.42	19	0.36
$A^{(40)}$	39	0.72	31	0.52	24	0.39
$A^{(45)}$	44	0.78	33	0.55	27	0.45
$A^{(50)}$	40	0.75	39	0.63	24	0.44
$A^{(55)}$	40	0.74	47	0.78	25	0.42
$A^{(60)}$	47	0.85	80	1.41	31	0.56
$A^{(65)}$	47	0.80	107	1.64	27	0.42
$A^{(70)}$	38	0.75	72	1.28	28	0.51
$A^{(75)}$	114	2.03	230	4.06	105	1.96
$A^{(80)}$	63	1.19	87	1.51	80	1.42
$A^{(35)}$	33	0.66	36	0.63	35	0.67
$A^{(40)}$	39	0.72	37	0.64	35	0.59
$A^{(45)}$	44	0.78	42	0.67	35	0.59
$A^{(50)}$	40	0.75	43	0.67	29	0.45
$A^{(55)}$	40	0.74	57	0.95	31	0.53
$A^{(60)}$	47	0.85	84	1.37	33	0.54
$A^{(65)}$	47	0.80	102	1.55	34	0.52
$A^{(70)}$	38	0.75	87	1.47	34	0.58
$A^{(75)}$	114	2.03	163	2.65	147	2.45
$A^{(80)}$	63	1.19	81	1.38	93	1.64

Here we show results only for some linear systems from the beginning of the sequences (as given by the superscripts); the whole sequence has more than 1000 linear

systems. Three preconditioning strategies were tested: Recomputation, freezing and updating. Updates were always related to the first matrix of the sequence. In the first sequence of Table 10, the preconditioner that is being frozen or updated was computed for the matrix $A^{(0)}$, and in the second sequence it was taken from the 30th linear system. The update strategy was implemented as a black-box routine which decides which of the updates (unstructured update from Algorithm 3.1 or 3.2, structured update based on the upper triangular part of the difference matrix, structured update based on the lower triangular part of the difference matrix) is used, based on the sum of magnitudes of strong matrix entries. The structured updates store the update separately although merging with the decomposition could provide even better timings. The results are characterized by the number of iterations of the BiCGSTAB method, and by the timings of the preconditioned iterative method to solve the individual linear systems, including the time to compute the preconditioner. The average time to compute the preconditioner is denoted by *timep* and its average number of nonzeros is denoted by *psize*. These last two characteristics slightly differ in individual computations of a sequence of problems. Note that preconditioning this problem was necessary, the unpreconditioned method worked rather poorly.

From Table 10 we can see once more that freezing the preconditioner may not be enough for getting efficiently preconditioned iterative methods for all the systems. Freezing with updating is typically better in terms of the number of matvecs. The additional solve with the update may add a time penalty but its influence seems to be limited. Clearly, by changing the matrix more and more the gap between the efficiency of freezing and updating gets larger up to some point where, of course, also the update is not sufficient anymore. We included this point in our table but in practice this would be the moment to recompute a factorization. As in the previous problem, it seems that the update is even more powerful than the recomputed preconditioners in the sense of giving the smallest number of iterations among all the three preconditioning strategies. This must be mainly caused by the fact that recomputation becomes less stable as the sequence proceeds, as can be seen from the iteration numbers around the 75th linear system. However, the role of additional structural information provided by updates should not be underestimated. In Table 12 we will consider a sequence without instability regions where updates are still more powerful than recomputed factorizations.

The following Table 11 presents qualitatively the same results for a larger matrix. As above, a powerful ILUT preconditioner was chosen in order to provide small iteration counts, and to have the number of nonzeros of the preconditioner similar to the number of nonzeros of the original matrix. Note that for most of the more difficult problems, the time needed to solve the linear system is the best for our updates. While, as above, there is a similar behavior of the iteration counts we also show results for more matrices around the point where the original frozen preconditioner stops to be useful. Note that for some matrices the updated preconditioner behaves *much* better than the other strategies.

In Table 12 we consider discretization leading to matrices of a dimension about sixty thousand. Most of the remarks on the previous two tables can be made here too, though note that there are no instability regions anymore. As before, updates achieve an acceleration compared to recomputing of up to 90%. The relation to the freezing strategy is the same as for the corresponding problems of smaller dimension. A noteworthy difference with smaller dimensions is that the ratio of the average time to recompute the preconditioner ('timep') and the time to solve the systems,

TABLE 11
Air flow in a tunnel, n=9600, nnz=277224, ILUT$(10^{-7}, 30)$.

ILUT$(10^{-7}, 30)$, timep ≈ 0.1, psize ≈ 283751						
Matrix	Recomp		Freeze		Update	
	Its	Time	Its	Time	Its	Time
$A^{(0)}$	3	0.13	3	0.13	3	0.13
$A^{(5)}$	3	0.13	3	0.03	3	0.03
$A^{(10)}$	4	0.15	4	0.05	5	0.05
$A^{(15)}$	4	0.15	5	0.06	6	0.06
$A^{(20)}$	5	0.15	6	0.06	7	0.09
$A^{(30)}$	7	0.18	7	0.08	8	0.11
$A^{(40)}$	8	0.23	14	0.16	14	0.17
$A^{(45)}$	9	0.23	18	0.17	20	0.23
$A^{(46)}$	11	0.24	22	0.23	16	0.18
$A^{(47)}$	11	0.23	18	0.19	16	0.18
$A^{(48)}$	15	0.29	23	0.25	22	0.26
$A^{(49)}$	15	0.30	23	0.25	22	0.29
$A^{(50)}$	16	0.33	24	0.23	19	0.23
$A^{(51)}$	27	0.48	31	0.38	25	0.33
$A^{(52)}$	47	0.69	33	0.34	27	0.31
$A^{(53)}$	44	0.73	33	0.39	23	0.29
$A^{(54)}$	67	1.12	54	0.61	32	0.43
$A^{(55)}$	92	1.49	196	2.23	56	0.84
$A^{(56)}$	76	1.21	131	1.48	40	0.54
$A^{(57)}$	79	1.33	81	1.05	51	0.80
$A^{(58)}$	52	0.91	45	0.59	34	0.51
$A^{(59)}$	50	1.02	40	0.63	38	0.65
$A^{(60)}$	32	0.74	961	15.3	440	7.98

is much larger. Hence avoiding recomputation becomes more important with larger dimensions. To conclude, let us mention the problem of recomputing related to a different preconditioner. This large air flow problem with the standard AINV(0.1) preconditioner with a number of nonzeros close to the number of nonzeros in the first matrix of the sequence converges in 12 iterations in average, the time to compute the preconditioner is 1.67 s and time for the BiCGSTAB-iterations is 0.25 s! We may assume that the role of avoiding frequent recomputations will be in this case significantly increased but we did not follow this line.

5. Conclusions. In this paper we proposed a couple of algebraic procedures which may be useful for solving sequences of systems of linear equations. The numerical experiments show that our updated preconditioners can be rather successful in practice, and the updates can often replace recomputation of preconditioners. In many cases, one would like to make the overall number of operations smaller with simple updates, and our experiments confirm that this is possible. In particular, the preconditioner update seems to be more advantageous than the other approaches if one of the following cases applies: if preconditioner computation is not cheap, if its recomputation is unstable or if the update is structurally dominant, that is, if it covers

TABLE 12
Air flow in a tunnel, n=59392, nnz=1127211, ILUT$(10^{-8}, 8)$.

ILUT$(10^{-8}, 8)$, timep ≈ 0.45, psize $\approx$ 1307000-1490000						
Matrix	Recomp		Freeze		Update	
	Its	Time	Its	Time	Its	Time
$A^{(0)}$	24	1.25	24	1.25	24	1.25
$A^{(2)}$	21	1.13	27	0.95	23	0.88
$A^{(4)}$	22	1.15	27	0.90	23	0.89
$A^{(6)}$	21	1.15	27	0.90	23	0.90
$A^{(8)}$	21	1.14	26	0.93	23	0.89
$A^{(10)}$	22	1.15	26	0.91	23	0.91
$A^{(12)}$	24	1.23	27	0.97	23	0.88
$A^{(14)}$	23	1.20	27	1.01	23	0.90
$A^{(16)}$	24	1.23	27	0.95	22	0.89
$A^{(18)}$	24	1.27	27	0.92	22	0.89
$A^{(20)}$	25	1.23	28	0.90	21	0.83
$A^{(22)}$	25	1.24	28	0.92	22	0.86
$A^{(24)}$	26	1.29	28	0.98	22	0.84
$A^{(26)}$	29	1.60	28	1.00	22	0.85
$A^{(28)}$	30	1.43	29	0.95	22	0.84
$A^{(30)}$	28	1.37	28	0.97	23	0.89
$A^{(32)}$	31	1.53	33	1.06	22	0.81
$A^{(34)}$	28	1.42	28	0.95	23	0.89
$A^{(36)}$	31	1.51	30	1.02	22	0.91
$A^{(38)}$	30	1.51	29	1.01	23	0.95

a significant part of the difference matrices from subsequent problems. Nevertheless, there can be also different, and sometimes very strong, reasons for avoiding preconditioner recomputations. In matrix-free and/or parallel environments, which are nowadays quite common, any recomputation of a preconditioner may be expensive. This is especially true for strong algebraic preconditioners which are used for solving difficult problems. We used intentionally structured updates based on one triangular part only. Part of our motivation was that we concentrated on finding methods for problems where the non-symmetry is apparent. In addition, we are interested in the structured update since we expect possible cheap estimation of sparsified triangular matrices. This may be important in matrix-free environment. Note that our unstructured updates are very close to permuted (and sparsified) triangular updates. We intend to present fully matrix-free results in the near future. Another issue which we currently investigate is combination of approximate factorizations with various Gauss-Seidel type preconditioners to define updates.

An interesting problem which we would like to consider in the future is to find first a nonsymmetric permutation which transforms the system matrices into a form more suitable for one particular structured or unstructured update. In particular, this permutation may make one triangular part of the matrices more heavy (in the sense of the sum of magnitudes of its entries) than the other triangular part. This may have a connection to the combinatorial method in Algorithm 3.2 to find an unstructured update. The use of a weighted spanning tree strongly reminds the popular strat-

egy of matchings-based nonsymmetric permutations which has significantly improved algebraic preconditioning in recent years [23], [5].

Acknowledgments. The authors thank Philipp Birken and Ladislav Lukšan for providing the software for solving the nonlinear problems and for useful instructions to work with it. They thank Andreas Meister for initiating application of the proposed techniques to the tunnel problem.

REFERENCES

[1] M.A. Ajiz and A. Jennings. A robust incomplete Choleski-conjugate gradient algorithm. *Int. J. Numer. Methods Engrg.*, 20:949–966, 1984.
[2] J. Baglama, D. Calvetti, G.H. Golub, and L. Reichel. Adaptively preconditioned GMRES algorithms. *SIAM J. Sci. Comput.*, 20:243–269, 1998.
[3] M. Benzi and D. Bertaccini. Approximate inverse preconditioning for shifted linear systems. *BIT Numer. Math.*, 43:231–244, 2003.
[4] M. Benzi, J. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM J. Sci. Comput.*, 22:1318–1332, 2000.
[5] M. Benzi, J.C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Sci. Comput.*, 21:1333–1353, 2000.
[6] M. Benzi, C.D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17:1135–1149, 1996.
[7] M. Benzi and M. Tůma. Orderings for factorized sparse approximate inverse preconditioners. *SIAM J. Sci. Comput.*, 21:1851–1868, 2000.
[8] L. Bergamaschi, R. Bru, A. Martínez, and M. Putti. Quasi-Newton preconditioners for the inexact Newton method. *ETNA*, 23:76–87, 2006.
[9] D. Bertaccini. Efficient preconditioning for sequences of parametric complex symmetric linear systems. *Electronic Transactions on Numerical Mathematics*, 18:49–64, 2004.
[10] P. Birken. *Numerical Simulation of Flows at Low Mach Numbers with Heat Sources.* PhD thesis, University of Kassel, Kassel, Germany, 2005.
[11] R. P. Brent. Some efficient algorithms for solving systems of nonlinear equations. *SIAM J. Numer. Anal.*, 10:327–344, 1973.
[12] P.N. Brown and Y. Saad. Hybrid Krylov methods for solving systems of nonlinear equations. *SIAM J. Sci. Stat. Comput.*, 11:450–481, 1990.
[13] X.-C. Cai and D.E. Keyes. Nonlinearly preconditioned inexact Newton algorithms. *SIAM J. Sci. Comput.*, 24:183–200, 2002.
[14] K. Chen. *Matrix Preconditioning Techniqes and Applications.* Cambridge Monographs in Applied and Computational Mathematics, No. 19, Cambridge University Press, Cambridge, 2005.
[15] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *J. Comput. Appl. Math.*, 86:387–414, 1997.
[16] P. Concus and G.H. Golub. Use of fast direct methods for the efficient numerical solution of nonseparable elliptic equations. *SIAM J. Numer. Anal.*, 10:1103–1120, 1973.
[17] P. Concus and G.H. Golub. A generalized conjugate gradient method for nonsymmetric systems of linear equations. In *Computer Methods in Applied Sciences and Engineering, Lecture Notes in Economics and Mathematical Systems, vol. 134.* Springer-Verlag, Berlin, New York, 1976.
[18] A.R. Curtis, M.J.D. Powell, and J.K. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119, 1974.
[19] T.A. Davis and W.W. Hager. Modifying a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 20:606–627, 1999.
[20] T.A. Davis and W.W. Hager. Multiple-rank modifications of a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 22:997–1013, 2001.
[21] T.A. Davis and W.W. Hager. Row modification of a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 26:621–639, 2005.
[22] I.S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20:889–901, 1999.
[23] I.S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.*, 22:973–996, 2001.

[24] H. Elman and M. Schultz. Preconditioning by fast direct methods for non-self adjoint nonseparable elliptic equations. *SIAM J. Numer. Anal.*, 23:44–57, 1986.
[25] P.F. Fischer. Projection techniques for iterative solution of $Ax = b$ with successive right-hand sides. *Comp. Meth. Appl. Mech. Engng.*, 163:193–204, 1998.
[26] R. Freund. A transpose-free quasi-minimal residual algorithm for non-hermitian linear systems. *SIAM J. Sci. Comput.*, 14:470–482, 1993.
[27] A.H. Gebremedhin, F. Manne, and A. Pothen. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Review*, 47:629–705, 2005.
[28] P.E. Gill, W. Murray, and M.A. Saunders. Methods for computing and modifying the LDV factors of a matrix. *Math. Comput.*, 29(132), 1975.
[29] G.H. Golub and C.F. Van Loan. *Matrix Computations. 3rd ed.* The Johns Hopkins University Press, Baltimore and London, 1996.
[30] D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM J. Sci. Comput.*, 22:2194–2215, 2001.
[31] J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. of the AMS*, 2:48–50, 1956.
[32] I.E. Kaporin and O. Axelsson. On a class of nonlinear equation solvers based on the residual norm reduction over a sequence of affine subspaces. *SIAM J. Sci. Comput.*, 16:228–249, 1995.
[33] C.T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, Philadelphia, 1995.
[34] C.T. Kelley. *Solving Nonlinear Equations with Newton's Method*. SIAM, Philadelphia, 2003.
[35] D.S. Kershaw. The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *J. Comp. Phys.*, 26:43–65, 1978.
[36] D. Keyes. Terascale implicit methods for partial differential equations. In *Contemporary mathematics*, volume 306, pages 29–84. AMS, Providence, 2001.
[37] S.A. Kharchenko, L.Yu. Kolotilina, A.A. Nikishin, and A.Yu. Yeremin. A reliable AINV-type preconditioning method for constructing sparse approximate inverse preconditioners in factored form. *Numer. Linear Algebra Appl.*, 8:165–179, 2001.
[38] D.A. Knoll and D.E. Keyes. Jacobian-free Newton-Krylov methods: A survey of approaches and applications. *J. Comp. Phys.*, 193:357–397, 2004.
[39] D.A. Knoll and P.R. McHugh. Newton-Krylov methods applied to a system of convection-reaction-diffusion equations. *Comput. Phys. Commun.*, 88:141–160, 1995.
[40] D.A. Knoll, P.R. McHugh, and D.E. Keyes. Newton-Krylov methods for low Mach number compresible combustion. *AIAA Journal*, 34:961–967, 1996.
[41] I. Lee, P. Raghavan, and E.G. Ng. Efective preconditioning through ordering interleaved with incomplete factorization. *SIAM J. Matrix Anal. Appl.*, 27:1069–1088, 2006.
[42] D. Loghin, D. Ruiz, and A. Touhami. Adaptive preconditioners for nonlinear systems of equations. *J. Comput. Appl. Math.*, 189:326–374, 2006.
[43] L. Lukšan, M. Tůma, J. Vlček, N. Ramešová, M. Šiška, J. Hartman, and C. Matonoha. UFO 2004 - interactive system for universal functional optimization. Technical Report V-923, ICS AS CR, 2004.
[44] T.A. Manteuffel. An incomplete factorization technique for positive definite linear system. *Math. Comput.*, 34:473–497, 1980.
[45] J.L. Morales and J. Nocedal. Automatic preconditioning by limited-memory quasi-Newton updates. *SIAM J. Opt.*, 10:1079–1096, 2000.
[46] D.P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra Appl.*, 29:293–322, 1980.
[47] M. Olschowka and A. Neumaier. A new pivoting strategy for Gaussian elimination. *Linear Algebra Appl.*, 240:131–151, 1996.
[48] M.L. Parks, E. de Sturler, G. Mackey, D.D. Johnson, and S. Maiti. Recycling Krylov subspaces for sequences of linear systems. *SISC, to appear*, 2006.
[49] D.W. Peaceman and H.H. Rachford. The numerical solution of parabolic and elliptic differential equations. *SIAM J. Appl. Math.*, 3:28–41, 1955.
[50] R.C. Prim. Shortest connection networks and some generalizations. *Bell Systems Tech. Journal*, 36:1389–1401, 1957.
[51] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Stat. Comput.*, 14:461–469, 1993.
[52] Y. Saad. ILUT: a dual threshold incomplete ILU decomposition. *Num. Lin. Alg. Appl.*, 1:387–402, 1994.
[53] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 7:856–869, 1986.
[54] V.E. Shamanskii. A modification of Newton's method. *Ukrain. Mat. Z.*, 19:1333–1338, 1967.

[55] V. Simoncini and E. Gallopoulos. An iterative method for nonsymmeric systems with multiple right-hand sides. *SIAM J. Sci. Comput.*, 16:917–933, 1995.

[56] V. Simoncini and D. Szyld. Flexible inner-outer krylov subspace methods. *SIAM J. Numer. Anal.*, 40:2219–2239, 2003.

[57] M. Tismenetsky. A new preconditioning technique for solving large sparse linear systems. *Lin. Alg. Appl.*, 154–156:331–353, 1991.

[58] H.A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[59] P. Vassilevski. Preconditioning nonsymmetric and indefinite finite element matrices. *Numer. Linear Algebra Appl.*, 1:59–76, 1992.

[60] B. Vital. *Etude de quelques méthodes de résolution de problèmes linéaires de grande taille sur multiprocesseur*. PhD thesis, Université de Rennes I., Rennes, 1990.

[61] J.W. Watts-III. A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation. *Society of Petroleum Engineers J.*, 21:345–353, 1981.

[62] O. Widlund. A Lanczos method for a class of nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 15:801–812, 1978.