NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2000; **00**:1–6 *Prepared using* nlaauth.cls *[Version: 2002/09/18 v1.02]*

Preconditioner updates for solving sequences of linear systems in matrix-free environment[†]

Jurjen Duintjer Tebbens and Miroslav Tůma[*]

Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 18207 Praha 8, Czech Republic.

SUMMARY

We present two new ways of preconditioning sequences of nonsymmetric linear systems in the special case where the implementation is matrix-free. Both approaches are fully algebraic, they are based on the general updates of incomplete LU decompositions recently introduced in [1], and they may be directly embedded into nonlinear algebraic solvers. The first of the approaches uses a new model of partial matrix estimation to compute the updates. The second approach exploits separability of function components to apply the updated factorized preconditioner via function evaluations with the discretized operator. Experiments with matrix-free implementations of test problems show that both new techniques offer useful, robust and black-box solution strategies. In addition, they show that the new techniques are often more efficient in matrix-free environment than either recomputing the preconditioner from scratch for every linear system of the sequence or than freezing the preconditioner throughout the whole sequence.

KEY WORDS: preconditioned iterative methods, matrix-free environment, factorization updates, inexact Newton-Krylov methods, incomplete factorizations

1. Introduction

We consider the solution of sequences of linear systems

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

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 numerous industrial and scientific computations, for example, when a system of nonlinear equations is solved by a Newton or Broyden-type method [2], [3]. Krylov subspace methods are among the most successful approaches for solving the

[*]Correspondence to: Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 18207 Praha 8, Czech Republic. E-mail: {tebbens,tuma}@cs.cas.cz

[†]This work was supported by the project No. IAA100300802 of the Grant Agency of the Academy of Sciences of the Czech Republic. 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.

Received 2008
Revised

linear systems. These methods have the property that the system matrix is needed only in the form of matrix-vector products, and the explicit representation of the matrix is not necessary. If the system matrix is not represented explicitly, we often say that the method is *matrix-free.*

It is widely recognized that in most cases of practical interest, Krylov subspace methods must be preconditioned in order to be efficient and robust. However most of the strong preconditioners either require the system matrix explicitly, or their computation may be rather expensive. In order to reduce the costs of the computation of preconditioners, we may reuse a preconditioner over more systems of the given sequence of systems of linear equations. The quality of the reused preconditioner may be enhanced through updates containing information extracted from the sequence of matrices, or from previous application of the Krylov subspace method. Due to the costs that are related to the fact that the system matrix is not given explicitly, and which may be magnified in a parallel computing environment, avoiding frequent recomputations of the preconditioner from scratch seems to be very important in matrix-free environment.

In this paper we address the problem of solving a sequence of general nonsymmetric systems in matrix-free environment by Krylov subspace methods with preconditioners that are based on incomplete LU decompositions and that are updated with general rank-n approximate modifications. In the next paragraph, we will briefly summarize basic lines of previous research on matrix-free preconditioning and on solving sequences of systems of linear equations with preconditioner updates, that is, on the two subproblems which we face. As far as we know, the combination of these subproblems has never been solved in the past. We propose two ways to do this by overcoming both theoretical and practical obstacles. These ways differ by the necessary sizes of intermediate memory and they can be useful in different applications.

Let us first mention several *preconditioning strategies designed for or related to matrix-free environment.* First, the preconditioners can correspond to a discretized operator which is simpler than the discretized operator for evaluating the sparse system matrix, see, e.g., [4], [5], [6], [7], [8], [9]. Successful preconditioners derived directly from the advection-diffusion operators were also proposed for solving problems in applications which provide rather dense Jacobian matrices [10], [11]. All these physics-based preconditioners are often used in matrix-free environment. Second, a preconditioner can be algebraic. The lack of explicit availability of the system matrix then often implies that the preconditioner is rather simple and/or sparse. In some of such situations, the preconditioner is the matrix diagonal or its approximation. In other situations, preconditioning employs more complex stationary iterative methods, fast FFT-based solvers, ADI methods, inner-outer schemes etc., in order to be easily applied in matrix-free environment. An early important paper which explicitly targets preconditioning in matrix-free environment is [12] with results for a model nonlinear boundary value problem, see also the applications in CFD [13]. For details on the important class of Jacobian-free Newton-Krylov methods (JFNK) which combine a matrix-free approach with nonsymmetric Krylov subspace methods and for modifications of this class, see the overview in [8], but also [4].

Preconditioner updates used for *solving system sequences* are traditionally based on modifications by matrices of small rank. Early work which uses the Broyden formula to update the preconditioner was introduced in [14]. Other updates based on matrices of small rank were considered in [15], [16], [17] and in limited-memory variable metric methods [18] for smooth optimization, to name just of few. An important class of algebraically-motivated strategies to accelerate the convergence of preconditioned iterative methods is based on constructing or improving the preconditioner by adaptive spectral information obtained directly from the

Krylov subspace methods, see, e.g., [19, 20, 21, 22, 23, 24, 25, 26]. All of these techniques have also a significant potential to be applied in the form of preconditioner updates for solving sequences of systems [27], [28], [29]. These strategies are often problem specific, but they are in general compatible with matrix-free implementations.

Although it is possible to analyze the spectral properties of sequences of preconditioned matrices in some important cases, in other situations we know much less. Preconditioner updates of small rank are often restricted to specific classes of problems or nonlinear schemes as well. Therefore, cheap and generally rank-n preconditioner updates are strongly needed. Recently some new approaches to approximate preconditioner updates were introduced, see, e.g., [30]. The authors in [31] propose approximate diagonal updates to solve parabolic PDEs, see also [32]. Nonsymmetric updates of general incomplete LU decompositions were considered in [1, 33], see also some results in solving real-world problems in [34]. So far, neither of these approaches have addressed the challenges related to preconditioner updates in matrix-free environment. Some of the mentioned preconditioner updates may not even be compatible with matrix-free environment.

This paper deals with matrix-free algorithms to solve the sequences of linear systems with the general triangular preconditioner updates introduced in [1]. Its adaptation for matrix-free environment is not straightforward and we propose two new approaches to do this. The first of them is based on an efficient matrix estimation technique. More precisely, a new *partial estimation* procedure is proposed. The second matrix-free approach applies the updates via modified forward or backward solves with the preconditioner, inside the iterative method. It is shown that both approaches may be robust in matrix-free environment. The paper is organized as follows. In Section 2 the general algorithmic framework for the updates is briefly summarized and some preliminary terminology for the two basic approaches in matrix-free environment is introduced. The first new approach is presented in detail in Section 3 and the second strategy is described in Section 4. Section 5 discusses numerical experiments for both approaches. The paper is finalized by some concluding remarks.

2. Basic update technique and matrix-free computations

The triangular preconditioner updates for nonsymmetric sequences from [1] are defined with the help of the difference between the matrix from the first (*reference*) linear system of a sequence and the *current* system matrix. Let A be the system matrix of the reference system and let A^+ be the current system matrix. If LDU is an incomplete triangular decomposition of A and $B = A - A^+$ is the difference matrix, then the basic triangularly updated preconditioners for the current system are defined as

$$(LD - tril(B))\,U, \qquad \text{or} \qquad L\,(DU - triu(B))\,, \tag{2.2}$$

where *tril* and *triu* denote, respectively, lower and upper triangular part of a matrix. We assume that $(LD - tril(B))$ or $(DU - triu(B))$, respectively, is nonsingular. Table 2.1 compares the costs and memory for one step of a preconditioned iterative method when one recomputes the preconditioner for the linear system of a sequence, denoted as “Recomp”, with the costs and memory when using the first update in (2.2), denoted as “Update”. With the recomputation strategy, we denote the approximate LU factors of A^+ by L^+, D^+ and U^+. The table shows that applying the updates is only slightly more expensive and needs a little more memory than

Table 2.1. Cost comparison between recomputed and updated preconditioning

type	initialization	solve step	memory
Recomp	$A^+ \approx L^+D^+U^+$	solves with L^+D^+, U^+	A^+, L^+D^+, U^+
Update	—	solves with $LD, U, tril(B)$	$A^+, tril(A), LD, U$

recomputing, but of course it saves all factorization costs (initialization). The table provides only a rough comparison; in particular, the amount of overlap between the sparsity patterns of LD and $tril(B)$ may have an important influence on the storage and application costs. When the overlap is important it makes sense to merge LD and $tril(B)$, i.e. compute the difference $LD - tril(B)$ and perform forward solves with it. Otherwise, separate solves with LD and $tril(B)$ (where only the entries on the main diagonal are merged) are usually more convenient for implementation reasons. In some cases A and A^+ may have not only somewhat different sparsity patterns, but also completely different sizes (i.e. numbers of nonzero entries).

The simple updates (2.2) can be expected to be efficient when the dominant information in the difference matrix B is contained in one triangular part, like for instance with upwind/downwind finite difference discretizations. However, we showed in [1, 33, 34] that the updates, possibly combined with improvements like specific reordering, Gauss-Jordan transforms or Gauss-Seidel-type extensions, are beneficial for a much broader spectrum of problems (including some CFD simulations discretized with finite volumes or elements). One remarkable feature is that the updates seem to yield powerful preconditioning not only, as one would expect from the definition (2.2), when the system matrices are changing slowly. In [34] the updates are most efficient in the transient phase of the simulation where turbulence causes large differences between system matrices. This may be explained by the fact that we take into account part of the large differences through the matrix B in (2.2). In addition, part of the structure of A and A^+ is contained in the updates.

The goal of this paper is not so much to show that preconditioner updates of the type (2.2) can yield a strong acceleration as compared to other preconditioning strategies. For examples where this is the case we refer to [1, 33, 34]. The aim of the paper is to introduce techniques that enable efficient usage of these updates in matrix-free environment and that can be applied as a black-box strategy. Because of the latter aim, our experiments often span whole nonlinear processes, even when we know that we could get a more profound effect by concentrating on transient phases of the nonlinear solvers. Without loss of generality, we will use the first type of update in (2.2) in our exposition. In practice, the type of update is chosen dynamically [33, 34].

The updates (2.2) are based on an incomplete reference factorization LDU. In many applications preconditioners which are simple, as those mentioned in the introduction, and thus naturally matrix-free, are not powerful enough due to linear or sub-linear convergence of the corresponding Krylov subspace method. Instead, strong and robust algebraic preconditioners like some type of incomplete decomposition must be used. They require, however, to be stored explicitly and, in order to be computed, they need the explicit entries of the matrix that they precondition. This means that the system matrix has to be estimated with the help of matrix-vector multiplications (matvecs). Let us shortly describe the standard matrix estimation strategy which is used very often in matrix-free nonlinear solvers or numerical optimization packages.

The matrix estimation problem is the problem to estimate a sparse matrix, given its sparsity structure, by a small number of well-chosen matvecs. Curtis, Powell and Reid [35] were the first to demonstrate that all nonzero entries of a sparse matrix can be estimated using a number of matvecs which is often much smaller than the matrix dimension. Estimation of the entries of a generally nonsymmetric matrix B can be formulated as the following problem.

Problem 2.1. *Given the sparsity pattern of B find vectors $d_1, \dots, d_p$ such that for each nonzero entry b_{ij} of B there is a vector $d_k, 1 \leq k \leq p$, satisfying $(Bd_k)_i = b_{ij}(d_k)_j$.*

In practice, we need to have p as small as possible so that the number of matvecs needed to obtain all nonzero entries is minimal. Coleman and Moré [36] demonstrated the relation of the matrix estimation problem 2.1 to the vertex *coloring* of a related graph G by a minimum number of colors. This minimum number is called the chromatic number of G. So-called direct methods for solving the matrix estimation problem for a matrix B described in Problem 2.1 use as G the intersection graph of B, that is the adjacency graph $G(B^T B)$ of $B^T B$. Note that for an (undirected) adjacency graph $G(C)$ of a square and symmetric matrix C we define its set of vertices as $V(G(C)) = \{1, \dots, n\}$ and its set of edges as $E(G(C)) = \{\{i, j\} \mid c_{ij} \text{ is nonzero}\}$. A vertex coloring of the intersection graph labels every vertex with a color in such a way that no two adjacent vertices have the same color. The number of groups of vertices of the related graph with the same color then corresponds to the number of matvecs needed to estimate all entries of the matrix. A recent survey of theoretical results and techniques in this field is [37] where one can find details on many standard matrix estimation strategies.

In matrix-free environment the factorization LDU in (2.2) has in general been obtained through estimating the reference matrix A, and it is stored explicitly. The update needs in addition *a part* of the difference matrix B, which is *not* given explicitly (only A has been estimated). Since the straightforward estimation of the difference matrix by applying Problem 2.1 to A^+ may be expensive, one possible strategy which we propose is based on modified matrix estimation that is reasonably cheap. In this case we use a new enhanced *partial* and *approximate* matrix estimation. This approach is described in Section 3. As in any matrix-free implementation that uses matrix estimation, we will assume that the sparsity pattern of A and A^+ is given. Note that very often the sparsity pattern can be obtained from the subroutine that performs the matvec: The variables involved in the definition of the kth entry of the output vector yield the sparsity pattern of the kth row of the system matrix.

The idea of estimating an implicitly given system matrix which may or may not be easily available, is frequently used in practice. Straightforward reasons to do so are, for instance, the need to avoid analytical computation of the system matrix, saving the storage costs for the system matrix or simply the fact that the preconditioners which do not need the matrix explicitly, may be weak. However, estimation based on Problem 2.1 needs some intermediate memory, and memory issues are often crucial in matrix-free environment and storage costs related to incomplete factorizations should be kept as low as possible. In Section 4 we will describe a strategy to use the triangular updates (2.2) in matrix-free environment without running any matrix estimation procedure other than for the reference matrix. Then the difference matrix B does not need to be stored. The costs of the technique depend on the degree of *separability* of the function components of the function which performs the matvec. Let us explain in the subsequent paragraph what we mean by separability in our case (cf. the concept of partial separability in optimization, e.g. in [38]).

Consider a Krylov subspace method where the product of the system matrix A with a vector v is replaced by the value of a function $\mathcal{F}$ evaluated at v. We say that $\mathcal{F}$ is separable if the evaluation of $\mathcal{F}$ can be easily separated in the evaluation of its function components. That is, the function $\mathcal{F}$ is well separable if the components of the function $\mathcal{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ can be written as $\mathcal{F}_i : \mathbb{R}^n \rightarrow \mathbb{R}$, where $e_i^T \mathcal{F}(v) = \mathcal{F}_i(v)$, and computing $\mathcal{F}_i(v)$ costs about an n-th part of the full function evaluation $\mathcal{F}(v)$. Note that in some cases, as they arise in complicated computations based on finite volumes or finite elements, the contributions for each volume or element are computed simultaneously, and in this case, the evaluation of a single function component costs more.

The next section presents our first technique, which is based on solving new matrix estimation problems, to apply the updates in matrix-free environment. Section 4 describes the second technique which fully avoids matrix estimation and assumes separability of the function components.

3. Matrix-free triangular updates via partial matrix estimation

This section describes our first proposal for computing and applying the triangular updates for a sequence of linear systems in matrix-free environment. As mentioned above, in general, it is possible to get a system matrix by solving the *matrix estimation problem*, i.e. Problem 2.1. We will see that we can efficiently estimate only a part of a given matrix, that is, we will solve a *partial matrix estimation problem* [37], [39].

Using the notation from above, consider matrices A and A^+ from a sequence. If we need to compute a preconditioner directly from A^+, then a straightforward strategy is to estimate A^+ entirely. When the sparsity patterns of A^+ and A are the same, we can use the same graph G to find, let us say, p color groups for both matrices (note that we typically use only approximate algorithms for graph coloring since the related decision problem is NP-complete [40]), and the graph coloring algorithm does not need to be rerun to estimate A^+ if we have estimated A. In this way, we need p matvecs for each estimation. If the matrix patterns in the sequence differ too much, we may need to run the graph coloring algorithm for A^+ as well, but its running time is typically smaller than the time needed for the matvecs. It was demonstrated in [39] that for A^+ we can use the results of the graph coloring algorithm for a matrix with a “slightly different” sparsity pattern.

In order to use the triangular updates described above we only have to estimate, in addition to A which was estimated earlier, the upper or the lower triangular part of A^+. This leads to a special partial matrix estimation problem. Without loss of generality, consider estimation of the lower triangular part $tril(A^+)$ of A^+. We will formulate this problem as a standard graph coloring problem (called 1-distance graph coloring problem; the problem can be formulated also differently using a different vertex coloring paradigm) for a graph which is different from the intersection graph of A^+. The following theorem describes this graph.

Theorem 3.1. *Consider the graph*

$$G_T(B) = G(tril(B)^T tril(B)) \cup G_K,$$

where $G(tril(B)^T tril(B)) = (V, E)$ *is the intersection graph of the lower triangular part of the matrix* B *and* G_K *is defined as*

$$G_K = \cup_{i=1}^n G_i, \qquad G_i = (V_i, E_i) = (V, \{\{k, j\} |\ b_{ik} \neq 0 \wedge b_{ij} \neq 0 \wedge k \leq i < j\}).$$

If the graph $G_T(B)$ can be colored by p colors, then the entries of the lower triangular part $tril(B)$ of B can be computed by p matvecs of B with vectors $d_1, \dots d_p$ such that for each nonzero entry b_{ij} of $tril(B)$ there is a vector $d_k, 1 \le k \le p$, satisfying $(Bd_k)_i = b_{ij}(d_k)_j$.

Proof: First note that the theorem gives necessary conditions to solve a modified Problem 2.1 in which we have to estimate only the entries of $tril(B)$ via matvecs with B. The intersection graph $G(tril(B)^T tril(B))$ prohibits the vectors $d_1, \dots d_p$ to contain in any component a sum of two or more nonzero entries from $tril(B)$. The graph G_K then prohibits to have in any of these vectors a component which would add a nonzero entry of $tril(B)$ with one or more nonzero entries of the strict upper triangular part of the adjacency graph $G(B^T B)$ of B. Note that the role of the index i in the definition of G_K restricts the adjacency of the entries b_{ik} and b_{ij} just to the cases when the former is from $tril(B)$ and the latter from the strict upper triangular part of B.

Assume now that $G_T(B)$ was colored by p colors. Define the vectors as usual in matrix estimation problems, that is $d_k, 1 \le k \le p$, such that

$$(d_k)_j = \begin{cases} 1 & \text{if the vertex } j \text{ has the color } k, \\ 0 & \text{otherwise.} \end{cases}$$

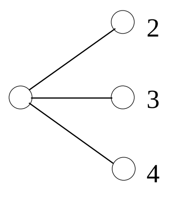
Consider a nonzero entry b_{ij} of $tril(B)$. There are edges $\{i, l\}$ in $G_T(B)$ for each $1 \le l \le n$ such that b_{il} is nonzero and their existence is a sufficient condition for separate computability of the entries of $tril(B)$. Note that this would not necessarily hold for a nonzero entry of B outside $tril(B)$. It also need not to be the case if G_K would be missing as we explained in this proof above. Then we have $(Bd_k)_i = b_{ij}$ for some k and we have the result. □

Note that the graph $G_T(B)$ contains only a subset of edges of the adjacency graph $G(B^T B)$ which should be considered to solve Problem 2.1. Consequently, in order to estimate only a triangular part of A^+ we may need a *smaller* number of matvecs than in the case of estimation of the whole A^+ (and subsequent extraction of the desired triangular part). We will show this in the following example.

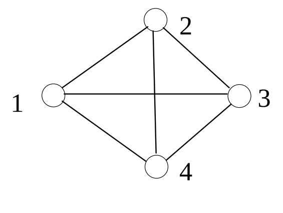
Example 3.1.

$$B^{(1)} = \begin{pmatrix} * & * & * & * \\ * & * & & \\ * & & * & \\ * & & & * \end{pmatrix},$$

$G_T(B^{(1)})$:



$G((B^{(1)})^T B^{(1)})$:



The graph $G_T(B^{(1)})$ for estimation of the lower triangular part can be colored with 2 colors but the graph $G((B^{(1)})^T B^{(1)})$ for estimating the whole matrix needs all 4 colors. Hence in general the lower triangular part of an n-dimensional matrix with the sparsity pattern of $B^{(1)}$ can always be estimated with 2 colors whereas estimating the whole matrix will require n colors.

This extreme situation will not arise often in practice, but nevertheless one can save an important amount of matvecs by restricting estimation to one triangular part. In order to enhance this effect, we also propose to perform a prefiltration that decreases the size of G_T. The prefiltration is based on the sparsity pattern of the reference matrix. We summarize the two crucial points of the approach we propose, i.e. partial estimation and prefiltration, in the algorithm below. The final matrix-free preconditioned iterative method with the updates to solve a sequence of linear systems needs to estimate the reference matrix as well as the triangular parts of the remaining matrices, so that they could be used in the updates. The following algorithm describes how these two tasks can be performed.

Algorithm 3.1. PARTIAL MATRIX ESTIMATION FOR TRIANGULAR PRECONDITIONER UPDATES. ***Input:*** *Matrix sequence* $A^{(0)}, A^{(1)}, \dots, A^{(n)}$ *and the sparsity pattern* $\mathcal{S}(A^{(0)})$.

```
1. Estimation. Estimate $A^{(0)}$ using $\mathcal{S}(A^{(0)})$.
2. Initial factorization. Factorize $A^{(0)}$ such that $A^{(0)} \approx LDU$.
3. Sparsification. Filtrate $A^{(0)}$ to get $\overline{A^{(0)}}$ and its sparsity pattern $\mathcal{S}(\overline{A^{(0)}})$.
4. for $i = 0, \dots, n$
       Estimate the lower triangular part $tril(A^{(i)})$ of $A^{(i)}$ using the coloring of $G_T(\overline{A^{(0)}})$
       and matrix-vector products with $A^{(i)}$. Then for $i \geq 1$, the lower triangular part of
       the difference matrix, $tril(A^{(0)}) - tril(A^{(i)})$, is used for the updates of the form (2.2).
   end for
```

Note that the lower triangular part $tril(A^{(0)})$ of $A^{(0)}$ is estimated twice. First, it is estimated as part of the whole matrix with the original sparsity pattern. Second, in step 4 of Algorithm 3.1, the filtrated pattern $\mathcal{S}(\overline{A^{(0)}})$ is used. Based on our experiments, the updates use just the latter quantity, that is, the updates use the lower triangular parts $tril(B^{(i)}) = tril(A^{(0)}) - tril(A^{(i)})$, for $i = 1, \dots, n$, which were all computed with the *filtrated and approximate* sparsity pattern $\mathcal{S}(\overline{A^{(0)}})$. Since the estimation adds some error to the computed matrix entries, it is important to distribute this error in all the approximate matrices in the same way. This explains our choice and the fact that the loop in Step 4 starts from 0. As for the sequential graph coloring heuristic, it tries to balance the error among the groups of columns of the same color as proposed in [39], see also [41]. Table 3.2 displays the costs and memory for one step of an iterative method preconditioned through recomputation and update, respectively, in matrix-free environment where updating is based on the approach from this section. We denote matrix estimations by $est(.)$. We see that in the initialization we save not only all factorization costs, but also estimation costs by solving a partial estimation problem.

An interesting aspect of the estimation of a triangular part of a matrix is that the number of colors depends on the matrix reordering since the graph construction depends on it. This

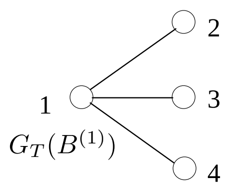
Table 3.2. Cost comparison with matrix-free updates based on Algorithm 3.1

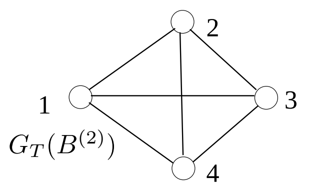
type	initialization	solve step	memory
Recomp	$est(A^+)$, $A^+ \approx L^+D^+U^+$	solves with L^+D^+, U^+	L^+D^+, U^+
Update	$est(tril(A^+))$	solves with $LD, U, tril(B)$	$tril(A^+), tril(A), LD, U$

is in contrast with the standard matrix estimation for the whole matrix explained in the previous section. It is easy to see this from Example 3.2, where we show the arrow matrix $B^{(1)}$ from Example 3.1 and its reordering $B^{(2)}$ together with the corresponding graphs $G_T(B^{(1)})$ and $G_T(B^{(2)})$. We have chosen this simple example to show the contrast between the Cuthill-McKee (CM) and the Reverse Cuthill-McKee reorderings (RCM) [42]. While $G_T(B^{(1)})$ reminds a part of the recursive structure which we get from the CM algorithm, $G_T(B^{(2)})$ shows a typical structure after reversing the sequence. The following theorem shows that sometimes we can enhance our chances to decrease the number of matvecs needed for the estimation of a triangular part of the matrix by an appropriate ordering of the matrix. Counterintuitively, the reverse Cuthill-McKee reordering may not be generally recommended.

Example 3.2.

$$B^{(1)} = \begin{pmatrix} * & * & * & * \\ * & * & & \\ * & & * & \\ * & & & * \end{pmatrix} \qquad B^{(2)} = \begin{pmatrix} * & & & * \\ & * & & * \\ & & * & * \\ * & * & * & * \end{pmatrix}$$





Theorem 3.2. *Assume that the irreducible matrix B with symmetric sparsity pattern was reordered by the Cuthill-McKee reordering. Further assume that the following condition applies: if $b_{ij} \neq 0$ for some i, j, $1 \le j \le i \le n$, then $b_{lj} \neq 0$ for all l, $j \le l \le i$ (envelope assumption). Denote by $\hat{B}$ the matrix which we obtain from B by reversing the order of rows and columns with respect to B, that is, $\hat{B}$ corresponds to the original matrix reordered by the related RCM reordering. Then the chromatic number of $G_T(B)$ is not larger than the chromatic number $G_T(\hat{B})$.*

Proof: We will use induction on i. Let us first define $f_i = \min\{j|\ b_{ij} \neq 0\}$ for $1 \le i \le n$. If B is reordered by the Cuthill-McKee reordering then we know that $f_i \le f_j$ if $1 \le i \le j \le n$ (monotone envelope property) and $f_i < i$ for $1 < i \le n$ [43].

The assertion is trivially valid for $i = 1$. Consider $i > 1$. Assume that we border the matrix of dimension $i-1$ by a row i from the bottom and a column i from the right. Let j be the minimum

$$\begin{pmatrix} * & * & * & & & \\ * & * & & * & & \\ * & & * & & * & * \\ & * & & * & & * \\ & & * & & * & * \\ & & * & * & * & * \end{pmatrix}$$

Figure 3.1. Matrix B after the Cuthill-McKee reordering for which its graph $G_T(B)$ needs more colors than the graph $G_T(\hat{B})$, where $\hat{B}$ we get from B after the symmetric reversal of its columns and rows.

index such that $b_{ij} \neq 0$. The nonzero entries in the i-th row induce a complete subgraph in $G_T(B)$. Because of the envelope assumption, this complete subgraph must be in $G_T(\hat{B})$ as well since the nonzeros b_{lj} for $j \leq l \leq i$ induce a clique. Consequently, $G_T(\hat{B})$ has all edges from $G_T(B)$ and its chromatic number is not smaller than the chromatic number of $G_T(B)$. $\square$

Note that CM/RCM reorderings are often used to preprocess a matrix of linear systems solved by preconditioned iterative methods. One motivation in the nonsymmetric case is that such reorderings may be very beneficial for the stability of the incomplete decomposition [44]. Further note that Theorem 3.2 is not valid without the envelope assumption. Figure 3.1 shows an example of a matrix B after the CM reordering. The chromatic number of $G_T(B)$ is five. B reordered by the related RCM needs only four colors. Despite this artificial counterexample, Theorem 3.2 points out that the CM reordering may generally be preferable when estimating the lower triangular part of a matrix.

4. Matrix-free updates via implicit back- or forward solves

This section describes an approach for applying the triangular preconditioner updates (2.2) in matrix-free environment which is principally different from the previous one. Whereas in the previous section we focused on efficient estimation of matrix entries, here we will avoid estimation as much as possible. We will show that we can apply the updates without knowledge and storage of the entries of the difference matrix B. This approach is therefore closer to the philosophy of matrix-free environment than the approach of the latest section. The price we pay for the reduced storage costs are forward or backward solves based on function evaluations while applying the updated preconditioner. The resulting increase in computational costs depends on the price of the function evaluation and on the degree of separability of the components of the function as defined in Section 2.

Assume for the moment that the triangular part of the matrix A and its incomplete LU decomposition are available explicitly (for simplicity, we hide the diagonal factor D of the decomposition in L). In practice, these quantities are computed for the reference system of the sequence. Let the current matrix A^+ be given implicitly in the form of its action on vectors, expressed by the function evaluation $\mathcal{F}^+(\cdot)$, where $\mathcal{F}^+ : \mathbb{R}^n \to \mathbb{R}^n$ and let $\mathcal{F}_i^+$ be the i-th component of $\mathcal{F}^+$. The strategy to apply the updated preconditioner

$$(L - tril(B))\, U \tag{4.3}$$

that we propose when the function components are separable is summarized in Algorithm 4.1.

Algorithm 4.1. APPLICATION OF TRIANGULAR PRECONDITIONER UPDATES WITH MIXED EXPLICIT-IMPLICIT SOLVES. ***Input:*** *Explicitly stored matrices L, U and $tril(A)$ and the function components of $\mathcal{F}^+$ which represent A^+ implicitly.*

1. **Initialization.** *Find the main diagonal $\{a^+_{11}, \dots, a^+_{nn}\}$ of A^+ before running the iterative method. It can be found by computing*

$$a^+_{ii} = \mathcal{F}^+_i(e_i), \qquad 1 \le i \le n.$$

2. **Forward solve in each iteration.** *Use the following mixed explicit-implicit strategy: Split the lower triangular matrix of (4.3) as $L - tril(B) = E + tril(A^+)$. That is, $E \equiv L - tril(A)$ is stored explicitly, and the implicit part $tril(A^+)$ contains entries of the new system matrix. We then have to solve triangular systems of the form*

$$\left(E + tril(A^+)\right) z = y,$$

which yields the forward solve loop

$$z_i = \frac{y_i - \sum_{j<i} e_{ij} z_j - \sum_{j<i} a^+_{ij} z_j}{e_{ii} + a^+_{ii}}, \qquad i = 1, 2, \dots, n. \tag{4.4}$$

Note that the values e_{ii} and a^+_{ii} in the denominator are known. In the numerator of (4.4), the first sum can be computed explicitly and the second sum can be computed with the function component evaluation

$$\mathcal{F}^+_i\left((z_1, \dots, z_{i-1}, 0, \dots, 0)^T\right) \approx e_i^T A^+ (z_1, \dots, z_{i-1}, 0, \dots, 0)^T = \sum_{j<i} a^+_{ij} z_j. \tag{4.5}$$

3. **Backward solve in each iteration.** *This is a trivial step since the matrix U in (4.3) has been stored explicitly.*

The costs to find the main diagonal in Step 1 (initialization) correspond approximately to the costs of one full function evaluation if the function components are well separable. Step 1 needs to be performed only once, before the preconditioned Krylov subspace method is started. Note that the diagonal of $B = A - A^+$ is known in advance in some applications. In particular, if the diagonal does not change, diag(B) is the zero matrix. In Step 2, the whole forward-solve loop requires n partial evaluations (4.5). In total, this gives approximately the cost of one additional full function evaluation per solve step of the preconditioned iterative method.

We assumed above that the triangular part $tril(A)$ of the reference matrix is stored explicitly. We mention that if the estimation of A has not been efficient or if the sparsity patterns of $tril(A)$ and L differ so much that the storage costs would grow unacceptably, one may also use the function components of $\mathcal{F}$, corresponding to A, to replace operations with $tril(A)$ in the forward solve. Then $tril(A)$ does not need to be stored and, formally, the explicit part E of the forward solve consists of L only. Then in (4.4) there are three sums of which the last two are computed implicitly:

$$z_i = \frac{y_i - \sum_{j<i} e_{ij} z_j - \sum_{j<i} a_{ij} z_j - \sum_{j<i} a^+_{ij} z_j}{e_{ii} + a_{ii} + a^+_{ii}}, \qquad i = 1, 2, \dots, n, \tag{4.6}$$

Table 4.3. Costs comparison with matrix-free updates based on Algorithm 4.1

type	initialization	solve step	memory
Recomp	$est(A^+)$, $A^+ \approx L^+U^+$	solves with L^+, U^+	L^+, U^+
Update	$est(diag(A^+))$	solves with L, U, eval$(\mathcal{F}, \mathcal{F}^+)$	L, U

where a_{ij} represents entries of the reference matrix A. The sum $\sum_{j<i} a_{ij} z_j$ is computed as

$$\mathcal{F}_i\left((z_1, \ldots, z_{i-1}, 0, \ldots, 0)^T\right).$$

The whole forward solve would then cost about two full function evaluations in total.

Let us compare the approach of this section with the strategy which recomputes the preconditioner for each system of a given sequence. With updates applied according to Algorithm 4.1, only the main diagonal of A^+ needs to be estimated (in the initialization). If we would recompute the preconditioner, on the other hand, we would have to estimate the whole matrix A^+, and, mainly, to compute the incomplete factorization. However, *application* of the update using Algorithm 4.1 could be more expensive than applying a new factorization if we would need a similar number of iterations since at least an extra full function evaluation in each forward solve is needed. Note that with the strategy of this section the memory costs for updating can be kept as low as they are for recomputing. The previous observations are displayed schematically in Table 4.3 for the case where forward solves are performed according to (4.6). We denote function evaluations by $eval(.)$.

5. Numerical experiments

This section is devoted to numerical experiments illustrating the techniques from Section 3 and 4. The main focus is not to show that the considered preconditioner updates can be very beneficial as compared to freezing or recomputing preconditioners; this has been shown elsewhere [1, 33, 34]. Here we present experiments with the new algorithms proposed for matrix-free environment. We focus on illustrating aspects of the proposed matrix-free implementation techniques and, in addition, the experiments show that updating is a robust alternative to freezing or recomputation in matrix-free environment. We attempted to use a variety of ILU decompositions and we performed tests with GMRES as well as with BiCGSTAB. In all experiments we consider minimization of functions with Newton-type methods and we use, to avoid storing Jacobians, the standard difference approximation of the Jacobian of the function F that is to be minimized. More precisely, a matvec with the Jacobian, Av, is replaced by

$$\mathcal{F}(v) \equiv \frac{F(x + \epsilon \cdot v/\|v\|) - F(x)}{\epsilon}, \tag{5.7}$$

for some small $\epsilon > 0$, where x is the point (vector) at which the Jacobian is approximated.

We will consider two test problems. The first problem results from a Newton-type method with a flexible stopping criterion for the linear system solution. Here the preconditioners with the triangular updates were fully embedded into the nonlinear solver. The next problem considers, on the other hand, a fixed sequence of linear systems generated from a nonlinear solver. All experiments were implemented in Fortran 95 on Intel Pentium-based machines.

5.1. Test Problem 1

In this first example we minimize a function with easily separable components resulting from a two-dimensional nonlinear convection-diffusion model problem with finite difference discretization. The convection-diffusion model problem has the form (see, e.g., [2])

$$-\Delta u + Cu\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) = f(x,y), \quad f(x,y) = 2000x(1-x)y(1-y), \tag{5.8}$$

where $C > 0$ is the Reynolds number, and it is discretized on the unit square. The standard five-point discretization stencil (central differences) results in a function F to minimize with components of the simple form

$$F_k(x) \equiv \frac{4x_k - x_{k-1} - x_{k+1} - x_{k+N} - x_{k-N}}{h^2} + Cx_k\frac{x_{k+1} - x_{k-1} + x_{k+N} - x_{k-N}}{h} - f_k,$$

for $1 \le k \le n$, where f_k is the discretization of f and N is the number of inner nodes. To solve $F(x) = 0$ we use an inexact Newton-Krylov method where the Krylov subspace method is BiCGSTAB and the stopping criterion of the iterative method is chosen adaptively. Newton's method is combined with a line-search technique for global convergence; the used method is described in detail on [45, page 215], see method DNS. The final matrix-free solver was embedded into the UFO-software [46] for nonlinear problems. The initial approximation is the discretization of $u_0(x,y) = 0$.

The preconditioner we use in the experiments is ILUT, see [47], and for our implementation we used Saad's ILUT code. We considered changes in the number of additional nonzeros allowed in the factorization, ranging from ILUT$(tol, 0) \equiv$ ILU(0) with the same sparsity pattern as the system matrix, to ILUT$(tol, 5)$. The drop tolerance was always $tol = 0.01$.

The ILUT factorizations are computed from the estimations of Jacobians obtained by running a graph coloring algorithm and performing matvecs corresponding to the obtained color groups. The graph coloring algorithm (which is based on a simple heuristic) yields between 5 and 7 colors, hence the matrix is estimated with about 5-7 matvecs calculated through means of function evaluations of the form (5.7). When, for the updates, we run Algorithm 3.1 and estimate only one triangular part of the system matrices A^+, then this always yields 5 colors and thus it will require 5 matvecs. In other words, the graph $G_T(A^+)$ of Theorem 3.1 has approximately as many edges as the intersection graph $G((A^+)^T A^+)$ of the whole matrix in this example (in the other test problem the situation is different). This is due to the structure of the system matrices and to the fact that the filtration in step 3 of Algorithm 3.1 does not sparsify the initial matrix (which is just a Laplacian). Of course, with the update strategy of Algorithm 4.1, the triangular parts of A^+ need not be computed at all.

In Figures 5.2, 5.3 and 5.4 we display results for several types of ILUT and different grid sizes. More precisely, the figures 5.2, 5.3 and 5.4 present experiments with the dimensions 62 500, 96 100 and 204 100, respectively, and for each dimension preconditioning with ILUT$(0.01, f)$ for the fill parameters $f = 0, 1, \ldots, 5$ is tested (see the x-axes of the graphs). Each figure contains two graphs where the first graph displays the number of BiCGSTAB iterations needed to reduce $\|F(x)\|$ to the value 10^{-15} and the second graph gives the needed CPU-time. The Reynolds number is chosen as $C = 500$, which yields sequences of about 10-12 linear systems. With this Reynolds number, ILU(0) is in general not the most efficient preconditioner; preconditioners with a number of nonzeros clearly larger than the system matrix yield less BiCGSTAB iterations (in this series of experiments, ILUT$(0.01, 5)$ has about three times as many nonzeros as the system matrix). The graphs compare four ways to precondition the sequences of linear systems: Dashed lines represent freezing of the preconditioner computed for the initial linear system, dash-dotted lines represent preconditioner recomputation for every linear system of the sequence, solid lines represent matrix-free updating based on (2.2) applied with Algorithm 3.1 and dotted lines represent matrix-free updating based on (2.2) applied with Algorithm 4.1. The choice between the two update types in (2.2) was made adaptively according to the triangular part of $B^1 = A^0 - A^1$ with the entries largest in magnitude (as proposed in [1]). This was always the upper triangular part, hence we always updated this triangular part.

The first observation which we get from the figures is that the total number of BiCGSTAB iterations needed to solve (5.8) is lowest when we recompute from scratch. However, updating based on (2.2) follows the curves for recomputing at close distance. As one would expect, the difference between using Algorithm 3.1 and Algorithm 4.1 for updating is marginal because they are just different implementations (i.e. based on different matrix-free computation techniques) of the same updated preconditioner. The number of iterations needed with the frozen preconditioner deteriorates soon during the solution of the sequence and is rather unpredictable. In particular, there is no clear dependence on the fill parameter f of the ILUT$(0.01, f)$ factorization. The plus ,,+" in Figure 5.4 indicates that the corresponding frozen ILUT factorization was too weak to solve the sequence arising from the nonlinear problem (5.8) at all.

The situation is quite different when we consider CPU-time. The repeated computation of the ILUT factorization is relatively expensive in this example of a matrix-free implementation and therefore the recomputation strategy is in general clearly less time-efficient than updating. An exception is given by the case of ILU(0) factorizations, which are, of course, cheap to (re)compute. For example, for the first series of tests with dimension $n = 62\,500$ the average time to compute ILU(0) is 1.5 seconds and the average computation times of ILUT$(0.01, 1)$, ILUT$(0.01, 2)$, ILUT$(0.01, 3)$, ILUT$(0.01, 4)$ and ILUT$(0.01, 5)$ are, respectively, 1.8, 2.6, 3.3, 3.7 and 4,3 seconds. Seen the length of the linear system sequence, this represents a considerable part of the total solution time when we use the recomputation strategy. For larger factorizations we observe that recomputing takes longer for denser factorizations, whereas updating is the *faster* the denser is the factorization. Algorithm 3.1 is more time-efficient than Algorithm 4.1; this is what one expects as Algorithm 4.1 requires about one additional function evaluation per backward solve with the updated preconditioner, hence two more function evaluations per BiCGSTAB iteration (in BiCGSTAB the preconditioner is applied twice in every iteration). On the other hand, Algorithm 4.1 has the advantage that no triangular parts of the current system matrices need to be stored (or estimated). If we would perform the forward solves according to (4.6), then we would not even need to store the lower triangular part of the reference system matrix, but the whole computation would be slower than when using Algorithm 4.1. The freezing strategy performs in general worst of all. This is explained by the deteriorating number of BiCGSTAB iterations, but note that in Figure 5.4 the timing of freezing and recomputing are the same though freezing needs much more BiCGSTAB iterations for convergence (this shows the high costs of recomputing).

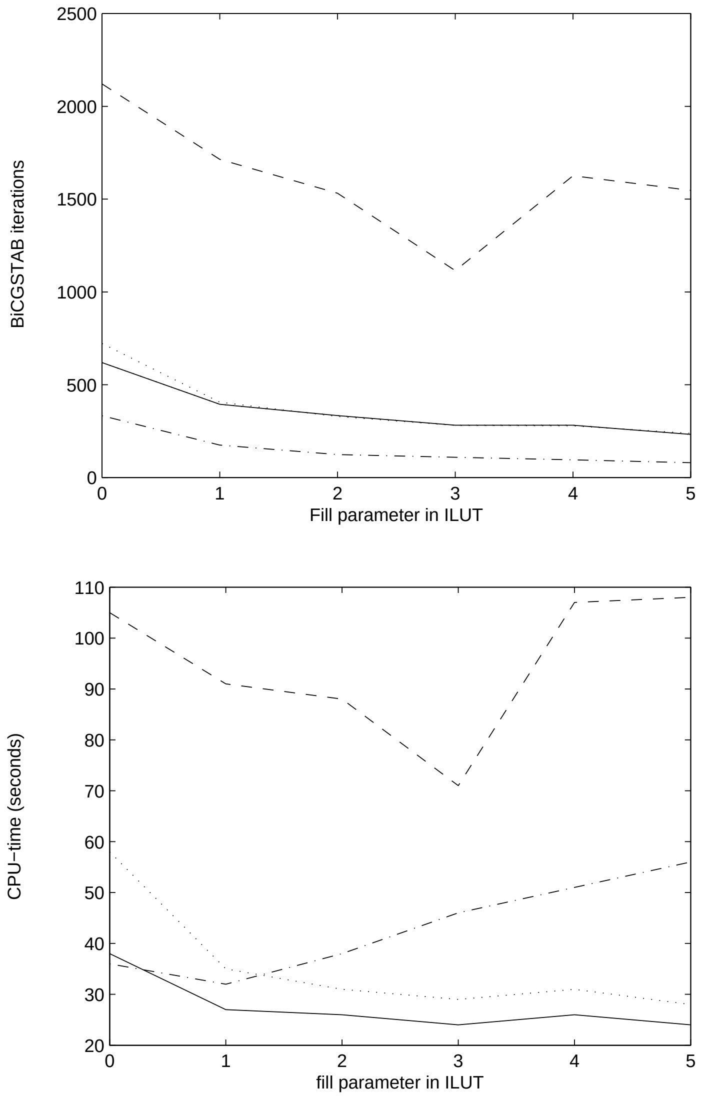
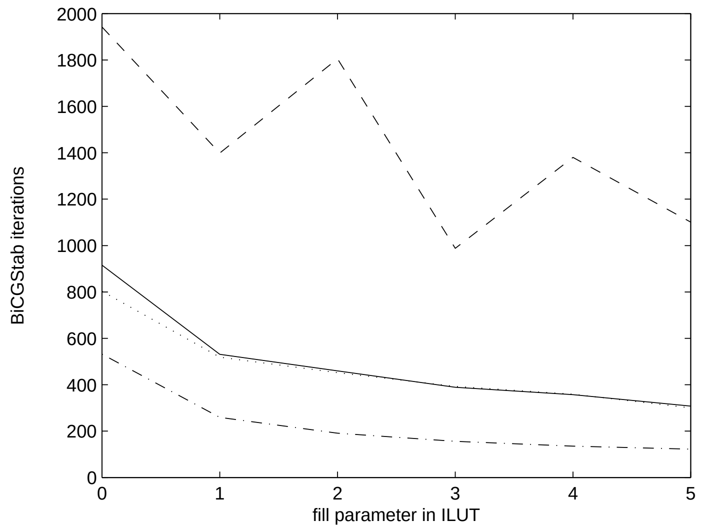


Figure 5.2. BiCGStab iterations and CPU-times to solve problem (5.8) with $C = 500$ on a 250×250 grid (dimension 62 500) with varying sizes of ILUT-factorizations (depending on the fill parameter) for freezing (dashed lines), recomputing (dash-dotted lines), updating with Algorithm 3.1 (solid lines) and updating with Algorithm 4.1 (dotted lines).

5.2. Test Problem 2

The second set of experiments is devoted to solving a sequence of linear problems arising during the computation of a constitutive model from structural mechanics provided by Karsten Quint.



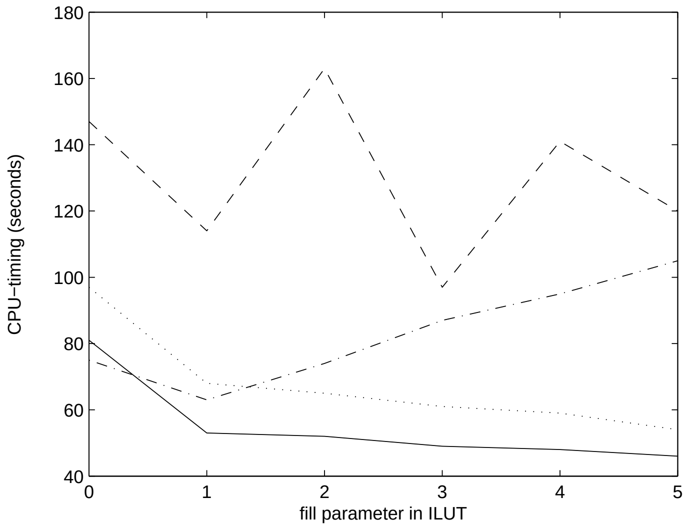
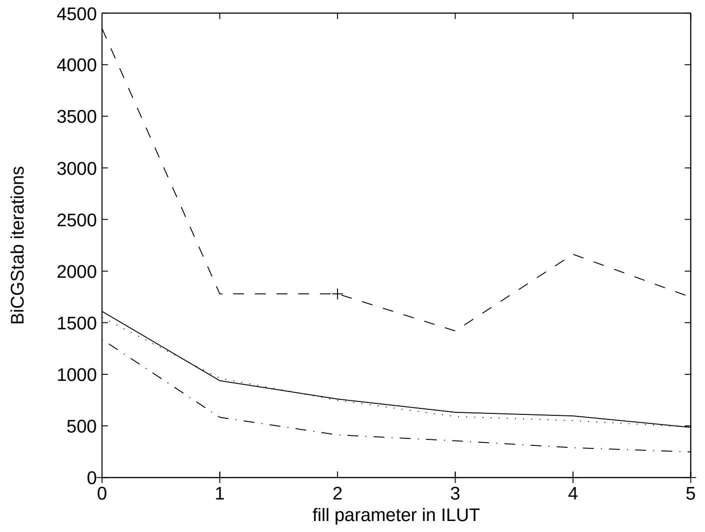


Figure 5.3. BiCGStab iterations and CPU-times to solve problem (5.8) with $C = 500$ on a 310×310 grid (dimension 96 100) with varying sizes of ILUT-factorizations (depending on the fill parameter) for freezing (dashed lines), recomputing (dash-dotted lines), updating with Algorithm 3.1 (solid lines) and updating with Algorithm 4.1 (dotted lines).

More precisely, a small strain metal viscoplasticity model was developed for a rectangular plate of length 100, width 21.2 and height 9.62 cm with a hole in the middle. The discretization used 1 350 quadratic elements in most of the domain with a somewhat finer grid in the center. When



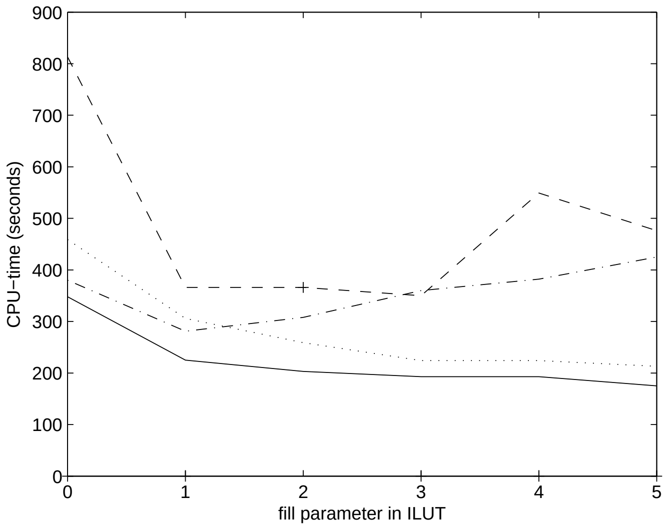


Figure 5.4. BiCGStab iterations and CPU-times to solve problem (5.8) with $C = 500$ on a 490×490 grid (dimension 240 100) with varying sizes of ILUT-factorizations (depending on the fill parameter) for freezing (dashed lines), recomputing (dash-dotted lines), updating with Algorithm 3.1 (solid lines) and updating with Algorithm 4.1 (dotted lines).

applying the Multilevel-Newton algorithm, every time-step contains an inner loop that requires the solution of nonlinear systems, which in turn leads to a sequence of linear systems. For more details on the parameters of the material and of the Multilevel-Newton algorithm which were used, we refer to the description of the first application in [48]. We consider here a sequence of linear systems from a randomly chosen time-step in the middle of the simulation process. This sequence consists of 8 linear systems of dimension 4 936 with matrices containing about 315 000 nonzeros. To solve the sequence, we use the GMRES(40) method preconditioned by ILUT. The (fixed) stopping criterion for GMRES is relative reduction of the residual norm below the level 10^{-6}. We again present results of several experiments differing in parameters provided to the preconditioner. In particular, we would like to show that the matrix-free updating strategies are successful over large variations in the preconditioner density.

Tables 5.4-5.8 present the results in terms of number of GMRES iterations and needed matrix-vector multiplications (that is, the function evaluations (5.7)). We compare again the four computational strategies: preconditioner recomputation by matrix estimation for each system (Recomp), preconditioner computation only for the reference matrix (Freeze), and preconditioning with the triangular updates based on Algorithm 3.1 and on Algorithm 4.1. Let us remind that Algorithm 3.1 evaluates in its loop also a less accurate approximation of the triangular part of $A^{(0)}$ using the filtrated pattern $\mathcal{S}(\overline{A^{(0)}})$ which we use to compute the updates, although we have a more accurate $A^{(0)}$ available. $A^{(0)}$ was filtrated in Algorithm 3.1 such that all the entries with magnitude smaller than half of the magnitude of the largest entry in their rows were dropped.

The average number of nonzeros of the factorizations is denoted with "Psize". The column "est. fevals" gives the number of function evaluations (fevals) needed for matrix estimation and "overall fevals" presents the total number of fevals needed to solve the sequence. The two "est. fevals" numbers for $A^{(0)}$ in the column "Update (Alg. 3.1)" correspond to its estimation with full and filtrated pattern. The other "est. fevals" numbers in this column give the number of matvecs needed to estimate only the triangular part of the current system matrix. To compute "overall fevals" we counted one function evaluation per GMRES iteration, as there is one matvec with the system matrix in every iteration of the GMRES method. In the "Update (Alg. 4.1)" strategy, however, every application of the updated preconditioner requires an additional function evaluation. Therefore we counted two function evaluations per GMRES iteration for this strategy.

The first fact which we observe is that the updating strategy works very well in terms of iteration counts and, in contrast with the previous test problem, it is from this point of view only slightly worse than recomputing. Consequently, it is clear that the updates are very often able to recover a lot of the information missing in the LU decomposition of the reference matrix. Second we observe that in terms of function evaluations, updating with Algorithm 3.1 is always the cheapest of all strategies. This is for an important part due to the difference between estimating the whole matrix and estimating only one triangular part.

Updating with Algorithm 4.1 requires less function evaluations than recomputing only for the densest factorizations in our series of experiments. It requires always more function evaluations than with Algorithm 3.1 (though the difference becomes smaller with more powerful initial factorizations). In addition, separate computation of the function components as needed in (4.5) is rather expensive due to the given finite volume implementation. This is the price we pay for the lower memory demands when using Algorithm 4.1. By using Algorithm 4.1 we save the storage of triangular parts of the size of about 160.000 nonzeros. Note that there may be applications where one cannot afford to store $tril(B)$ or $triu(B)$ in addition to the factors L and U at all and using the mixed explicit-implicit solves of Algorithm 4.1 would be the only feasible updating option. Both recomputation and updates are in general much better than the

Table 5.4. *Number of iterations and function evaluations for solving preconditioned linear systems from the structural mechanics problem with ILUT(0.01,5).*

ILUT(0.01,5), Psize $\approx$ 260 000								
Matrix	Recomp		Freeze		Update (Alg. 3.1)		Update (Alg. 4.1)	
	its	est. fevals	its	est. fevals	its	est. fevals	its	est. fevals
$A^{(0)}$	343	89	343	89	343	89+25	343	89
$A^{(1)}$	172	89	623	0	237	25	237	0
$A^{(2)}$	201	89	694	0	298	25	298	0
$A^{(3)}$	294	89	723	0	285	25	285	0
$A^{(4)}$	298	89	799	0	334	25	334	0
$A^{(5)}$	386	89	708	0	320	25	320	0
$A^{(6)}$	348	89	714	0	318	25	318	0
$A^{(7)}$	317	89	717	0	318	25	318	0
overall fevals	3 071		5 410		2 742		4 652	

freezing strategy. An exception is given in Table 5.8 where the ILUT$(10^{-6}, 70)$ factorization gives such low GMRES iteration counts that the estimation and recomputation costs start to dominate; here freezing is cheaper than recomputing.

As this sequence is a fixed sequence extracted from a structural mechanics problem solver and the experiments were not embedded in the solver, we do not present CPU timings and use just the number of function evaluations to get an idea of the computational effort. Note however, that if recomputing gives numbers of GMRES iterations close to updating, then its total timing, including factorization times, must necessarily be much worse. Let us also remind the experimental dependence of timings and iteration counts presented for the triangular updates in [1] if we assume similar sizes of preconditioners used in the compared strategies.

6. Conclusions

We have presented theoretical results and numerical experiments related to matrix-free strategies for solving sequences of linear systems by preconditioned iterative methods. In particular, we introduced two new approaches to apply triangular updates for enhancing the linear solver of the sequences. The experiments in matrix-free environment seem to confirm that the proposed strategies are typically the best of all compared possibilities. Moreover, the updates can be easily embedded into matrix-free nonlinear solvers.

Table 5.5. *Number of iterations and function evaluations for solving preconditioned linear systems from the structural mechanics problem with ILUT(0.001,20).*

ILUT(0.001,20), Psize $\approx$ 404 000								
Matrix	Recomp		Freeze		Update (Alg. 3.1)		Update (Alg. 4.1)	
	its	est. fevals	its	est. fevals	its	est. fevals	its	est. fevals
$A^{(0)}$	187	89	187	89	187	89+25	187	89
$A^{(1)}$	89	89	393	0	146	25	146	0
$A^{(2)}$	126	89	448	0	182	25	182	0
$A^{(3)}$	221	89	480	0	184	25	184	0
$A^{(4)}$	234	89	513	0	190	25	190	0
$A^{(5)}$	193	89	487	0	196	25	196	0
$A^{(6)}$	178	89	521	0	196	25	196	0
$A^{(7)}$	246	89	521	0	196	25	196	0
overall fevals	2 186		3 639		1 766		2 856	

Table 5.6. *Number of iterations and function evaluations for solving preconditioned linear systems from the structural mechanics problem with ILUT$(10^{-4}, 30)$.*

ILUT$(10^{-4}, 30)$, Psize $\approx$ 550 000								
Matrix	Recomp		Freeze		Updated (Alg. 3.1)		Updated (Alg. 4.1)	
	its	est. fevals	its	est. fevals	its	est. fevals	its	est. fevals
$A^{(0)}$	85	89	85	89	85	89+25	85	89
$A^{(1)}$	59	89	233	0	78	25	78	0
$A^{(2)}$	72	89	313	0	84	25	84	0
$A^{(3)}$	78	89	344	0	85	25	85	0
$A^{(4)}$	78	89	289	0	108	25	108	0
$A^{(5)}$	78	89	289	0	108	25	108	0
$A^{(6)}$	79	89	318	0	108	25	108	0
$A^{(7)}$	86	89	318	0	108	25	108	0
overall fevals	1 327		2 278		1 053		1 532	

7. Acknowledgments

We would like to gratefully acknowledge the help of Stefan Hartmann, Reijo Kouhia and Karsten Quint who provided the test problems. We are also very much indebted to Ladislav Lukšan for helping us to embed the updated preconditioners into the UFO software.

Table 5.7. *Number of iterations and function evaluations for solving preconditioned linear systems from the structural mechanics problem with* $ILUT(10^{-5}, 50)$.

$\mathrm{ILUT}(10^{-5}, 50), \mathrm{Psize} \approx 812\,000$								
Matrix	Recomp		Freeze		Updated (Alg. 3.1)		Updated (Alg. 4.1)	
	its	est. fevals	its	est. fevals	its	est. fevals	its	est. fevals
$A^{(0)}$	65	89	65	89	65	89+25	65	89
$A^{(1)}$	31	89	128	0	52	25	52	0
$A^{(2)}$	35	89	163	0	45	25	45	0
$A^{(3)}$	35	89	237	0	45	25	45	0
$A^{(4)}$	37	89	167	0	52	25	52	0
$A^{(5)}$	38	89	169	0	51	25	51	0
$A^{(6)}$	37	89	168	0	51	25	51	0
$A^{(7)}$	50	89	168	0	51	25	51	0
overall fevals	1 040		1 354		701		848	

Table 5.8. *Number of iterations and function evaluations for solving preconditioned linear systems from the structural mechanics problem with* $ILUT(10^{-6}, 70)$.

$\mathrm{ILUT}(10^{-6}, 70), \mathrm{Psize} \approx 950\,000$								
Matrix	Recomp		Freeze		Updated (Alg. 3.1)		Updated (Alg. 4.1)	
	its	est. fevals	its	est. fevals	its	est. fevals	its	est. fevals
$A^{(0)}$	32	89	32	89	32	89+25	32	89
$A^{(1)}$	21	89	78	0	54	25	54	0
$A^{(2)}$	28	89	88	0	38	25	38	0
$A^{(3)}$	24	89	101	0	39	25	39	0
$A^{(4)}$	26	89	92	0	38	25	38	0
$A^{(5)}$	26	89	87	0	38	25	38	0
$A^{(6)}$	26	89	86	0	38	25	38	0
$A^{(7)}$	28	89	86	0	38	25	38	0
overall fevals	923		739		604		687	

REFERENCES

1. Duintjer Tebbens J, Tůma M. Efficient preconditioning of sequences of nonsymmetric linear systems. *SIAM J. Sci. Comput.* 2007; **29**(5):1918–1941.
2. Kelley CT. *Iterative Methods for Linear and Nonlinear Equations.* SIAM: Philadelphia, 1995.
3. Kelley CT. *Solving nonlinear equations with Newton's method.* Fundamentals of Algorithms, Society for Industrial and Applied Mathematics (SIAM): Philadelphia, PA, 2003.
4. Brown PN, Saad Y. Hybrid Krylov methods for solving systems of nonlinear equations. *SIAM J. Sci. Stat. Comput.* 1990; **11**:450–481.
5. Mousseau VA, Knoll DA, Rider WJ. Physics-based preconditioning and the Newton-Krylov method for non-equilibrium radiation diffusion. *J. Comput. Phys.* 2000; **160**:743–765.

6. Keyes D. Terascale implicit methods for partial differential equations. *Contemporary Methematics, AMS, Providence* 2001; **306**:29–84.
7. Reisner J, Mousseau VA, Wyszogrodzki AA, Knoll DA. An efficient physics-based preconditioner for the fully implicit solution of small-scale thermally driven atmospheric flows. *J. Comput. Phys.* 2003; **189**:30–44.
8. Knoll DA, Keyes D. Jacobian-free Newton-Krylov methods: A survey of approaches and applications. *J. Comp. Phys.* 2004; **193**:357–397.
9. Bernsen E, Dijkstra HA, Wubs FW. A method to reduce the spin-up time of ocean models. *Ocean Modell.* 2008; **20**:380–392.
10. Li X, Primeau F. A fast Newton-Krylov solver for seasonally varying global ocean biogeochemistry models. *Ocean Modell.* 2008; **23**:13–20.
11. Khatiwala S. Fast spin up of ocean biogeochemical models using matrix-free Newton-Krylov. *Ocean Modelling* 2008; **23**:121–129.
12. Chan T, Jackson K. Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms. *SIAM J. Sci. Statist. Comput.* 1984; **5**(3):533–542.
13. Luo H, Baum JD, Löhner R. A fast, matrix-free implicit method for compressible flows on unstructured grids. *J. Comp. Physics* 1998; **146**(2):664–690.
14. Choquet R. A matrix-free preconditioner applied to CFD. *Technical Report No. 940*, INRIA, France 1995.
15. Chen Y, Shen C. A Jacobian-free Newton-GMRES(m) method with adaptive preconditioner and its application for power flow calculations. *IEEE Transactions on Power Systems* 2006; **21**(3):1096–1103.
16. Morales JL, Nocedal J. Automatic preconditioning by limited memory quasi-Newton updating. *SIAM J. Optim.* 2000; **10**(4):1079–1096 (electronic).
17. Bergamaschi L, Bru R, Martínez A, Putti M. Quasi-Newton preconditioners for the inexact Newton method. *ETNA* 2006; **23**:63–74.
18. Nocedal J. Updating quasi-Newton matrices with limited storage. *Math. Comp.* 1980; **35**(151):773–782.
19. Serra Capizzano S, Tablino Possio C. High-order finite difference schemes and Toeplitz based preconditioners for elliptic problems. *Electron. Trans. Numer. Anal.* 2000; **11**:55–84 (electronic).
20. Serra Capizzano S, Tablino Possio C. Preconditioning strategies for 2D finite difference matrix sequences. *Electron. Trans. Numer. Anal.* 2003; **16**:1–29 (electronic).
21. Bertaccini D, Golub GH, Serra Capizzano S, Tablino Possio C. Preconditioned HSS methods for the solution of non-Hermitian positive definite linear systems and applications to the discrete convection-diffusion equation. *Numer. Math.* 2005; **99**(3):441–484.
22. Kharchenko SA, Yeremin AY. Eigenvalue translation based preconditioners for the GMRES(k) method. *Numer. Linear Algebra Appl.* 1995; **2**(1):51–77.
23. Baglama J, Calvetti D, Golub GH, Reichel L. Adaptively preconditioned GMRES algorithms. *SIAM J. Sci. Comput.* 1998; **20**:243–269.
24. Erhel J, Burrage K, Pohl B. Restarted GMRES preconditioned by deflation. *J. Comput. Appl. Math.* 1996; **69**(2):303–318.
25. Morgan RB. A restarted GMRES method augmented with eigenvectors. *SIAM J. Matrix Anal. Appl.* 1995; **16**(4):1154–1171.
26. Saad Y, Yeung M, Erhel J, Guyomarc'h F. A deflated version of the conjugate gradient algorithm. *SIAM J. Sci. Comput.* 2000; **21**(5). Iterative methods for solving systems of algebraic equations (Copper Mountain, CO, 1998).
27. Parks ML, de Sturler E, Mackey G, Johnson DD, Maiti S. Recycling Krylov subspaces for sequences of linear systems. *SIAM J. Sci. Comput.* 2006; **28**(5):1651–1674 (electronic).
28. Wang S, de Sturler E, Paulino GH. Large-scale topology optimization using preconditioned Krylov subspace methods with recycling. *Internat. J. Numer. Methods Engrg.* 2007; **69**(12):2441–2468.
29. de Sturler E, Le C, Wang S, Paulino G. Large scale topology optimization using preconditioned Krylov subspace recycling and continuous approximation of material distribution. *Multiscale and Functionally Graded Materials 2006 (M&FGM 2006), Oahu Island (Hawaii), 15-18 October 2006*, G H Paulino, M-J Pindera, R H Dodds, Jr, F A Rochinha, E Dave, and L Chen, (ed.). AIP Conference Proceedings, 2008; 279–284.
30. Meurant G. On the incomplete Cholesky decomposition of a class of perturbed matrices. *SIAM J. Sci. Comput.* 2001; **23**(2):419–429 (electronic). Copper Mountain Conference (2000).
31. Benzi M, Bertaccini D. Approximate inverse preconditioning for shifted linear systems. *BIT* 2003; **43**(2):231–244.
32. Bertaccini D. Efficient preconditioning for sequences of parametric complex symmetric linear systems. *Electronic Transactions on Numerical Mathematics* 2004; **18**:49–64.
33. Duintjer Tebbens J, Tůma M. Improving triangular preconditioner updates for nonsymmetric linear systems. *LNCS* 2008; **4818**:737–744.
34. Birken P, Duintjer Tebbens J, Meister A, Tůma M. Preconditioner updates applied to CFD model

problems. *Appl. Num. Math.* 2008; **58**(11):1628–1641.

35. Curtis AR, Powell MJD, Reid JK. On the estimation of sparse Jacobian matrices. *J. Inst. Maths. Applics.* 1974; **13**:117–119.
36. Coleman TF, Moré JJ. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.* 1983; **20**:187–209.
37. Gebremedhin AH, Manne F, Pothen A. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Review* 2005; **47**:629–705.
38. Griewank A, Toint PL. On the unconstrained optimization of partially separable functions. *Nonlinear optimization, 1981 (Cambridge, 1981).* NATO Conf. Ser. II: Systems Sci., Academic Press: London, 1982; 301–312.
39. Cullum J, Tůma M. Matrix-free preconditioning using partial matrix estimation. *BIT Numer. Math.* 2006; **46**:711–729.
40. Garey MR, Johnson DS. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* Freeman & Co., 1979.
41. Siefert C, de Sturler E. Probing methods for saddle-point problems. *Electron. Trans. Numer. Anal.* 2006; **22**:163–183 (electronic).
42. Cuthill EH, McKee J. Reducing the bandwidth of sparse symmetric matrices. *Proc.* 24th *National Conference of the ACM*, ACM Press, 1969; 157–172.
43. Liu JWH, Sherman AH. Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM J. Numer. Anal.* 1976; **13**:198–213.
44. Benzi M, Szyld DB, van Duin A. Orderings for incomplete factorization preconditioning of nonsymmetric problems. *SIAM J. Sci. Comput.* 1999; **20**(5):1652–1670.
45. Lukšan L, Vlček J. Computational experience with globally convergent descent methods for large sparse systems of nonlinear equations. *Optim. Methods Softw.* 1998; **8**(3-4):201–223.
46. Lukšan L, Tůma M, Vlček J, Ramešová N, Šiška M, Hartman J, Matonoha C. UFO 2008 - interactive system for universal functional optimization. *Technical Report V-1040, downloadable from* http://www.cs.cas.cz/luksan/ufo.html, ICS AS CR 2008.
47. Saad Y. ILUT: a dual threshold incomplete *LU* factorization. *Numer. Linear Algebra Appl.* 1994; **1**(4):387–402.
48. Hartmann S, Duintjer Tebbens J, Quint K, Meister A. Iterative solvers within sequences of large linear systems in non-linear structural mechanics. *ZAMM* 2009; **89**:711–728.