SIAM J. SCI. COMPUT.
Vol. 36, No. 1, pp. A88–A104
© 2014 Society for Industrial and Applied Mathematics

PRECONDITIONING WITH DIRECT APPROXIMATE FACTORING OF THE INVERSE*

MIKKO BYCKLING† AND MARKO HUHTANEN‡

Abstract. To precondition a large and sparse linear system, two direct methods for approximate factoring of the inverse are devised. The algorithms are fully parallelizable and appear to be more robust than the iterative methods suggested for the task. A method to compute one of the matrix subspaces optimally is derived. Possessing a considerable amount of flexibility, these approaches extend the approximate inverse preconditioning techniques in several natural ways. Numerical experiments are given to illustrate the performance of the preconditioners on a number of challenging benchmark linear systems.

Key words. preconditioning, approximate factoring, parallelizable, sparsity pattern, approximate inverse

AMS subject classifications. 65F05, 65F10

DOI. 10.1137/12088570X

1. Introduction.

Approximate factoring of the inverse means parallelizable algebraic techniques for preconditioning a linear system involving a large and sparse nonsingular matrix $A \in \mathbb{C}^{n\times n}$. The idea is to multiply A by a matrix W from the right (or left) with the aim of having a matrix AW which can be approximated with an easily invertible matrix.[1] As opposed to the usual paradigm of preconditioning, iterations are not expected to converge rapidly for AW. Instead, the task can be interpreted as that of solving

$$\inf_{W\in\mathcal{W},\, V\in\mathcal{V}} \left|\left|AWV^{-1} - I\right|\right|_F \tag{1.1}$$

approximately by linearizing the problem appropriately [16, 4]. Here $\mathcal{W}$ and $\mathcal{V}$ are nonsingular sparse standard matrix subspaces of $\mathbb{C}^{n\times n}$ with the property that the nonsingular elements of $\mathcal{V}$ are assumed to allow a rapid application of the inverse. Approximate solutions to this problem can be generated with the power method as suggested in [4]. In this paper, direct methods are devised for optimal approximate factoring based on solving

$$\min_{W\in\mathcal{W},\, V\in\mathcal{V}} ||AW - V||_F \tag{1.2}$$

with the columns of either W or V being constrained to be of fixed norm. These two approaches allow, once the matrix subspace $\mathcal{W}$ has been fixed, choosing the matrix subspace $\mathcal{V}$ in an optimal way.

The first algorithm solves (1.2) when the columns of V are constrained to be of fixed norm. Then the matrix subspaces $A\mathcal{W}$ and $\mathcal{V}$ are compared as such while other

*Submitted to the journal's Methods and Algorithms for Scientific Computing section July 23, 2012; accepted for publication (in revised form) November 27, 2013; published electronically January 16, 2014.

http://www.siam.org/journals/sisc/36-1/88570.html

†CSC - IT Center for Science, P.O. Box 405, 02101 Espoo, Finland (Mikko.Byckling@csc.fi).

‡Mathematics Division, Department of Electrical and Information Engineering, University of Oulu, P.O. Box 4500, FIN-90401 Oulu, Finland (Marko.Huhtanen@oulu.fi).

[1] Direct methods are typically devised in this way, i.e., both the LU and QR factorizations can be interpreted such that the purpose is to multiply A with a matrix from the left so as to have an upper triangular, i.e., an easily invertible matrix.

properties of A are largely overlooked. The second algorithm solves the problem when the columns of W are constrained to be of fixed norm, allowing taking properties of A more into account. In [4] the approach to this end was based on approximating the smallest singular value of the map

$$W \longmapsto (I - P_{\mathcal{V}})AW \tag{1.3}$$

from $\mathcal{W}$ to $\mathbb{C}^{n\times n}$ with the power iteration. Here $P_{\mathcal{V}}$ denotes the orthogonal projector on $\mathbb{C}^{n\times n}$ onto $\mathcal{V}$. The second algorithm devised in this paper is a direct method for solving the task.

The algorithms proposed extend the standard approximate inverse computational techniques in several ways. (For sparse approximate inverse computations, see [1, section 5], [13] and [17, Chapter 10.5] and references therein.) First of all, our approximations to the inverse need not be sparse. Moreover, aside from possessing an abundance of degrees of freedom, we have an increased amount of optimality if we suppose the matrix subspace $\mathcal{W}$ to be given. Then computable conditions can be formulated for optimally choosing the matrix subspace $\mathcal{V}$. This is achieved without any significant increase in the computational cost. In particular, only a columnwise access to the entries of A is required.[2]

We aim at maximal parallelizability by solving the minimization problem (1.2) columnwise. The cost of such a high parallelism is the need to have a mechanism to somehow control the conditioning of the factors. After all, parallelism means performing computations locally and independently. Also this can be achieved without any significant increase in the computational cost.

Although the choice of the matrix subspace $\mathcal{W}$ is apparently less straightforward, some ideas are suggested to this end. Here we cannot claim achieving optimality, except that once done, thereafter $\mathcal{V}$ can be generated in an optimal way. In particular, because there are so many alternatives to generate matrix subspaces, many ideas outlined in this paper are certainly not fully developed and need to be investigated more thoroughly.

Finally, regarding related work, the idea of improving a computed preconditioner by computing another least squares approximation can be found in [17, Chapter 10.5.8]. The techniques are, by and large, the same as in the standard approximate inverse computations; see, e.g., [7, section 5.2] and [15]. In [14, section 3] there are techniques more related to the ones presented in this paper. There, the authors suggest constraining the diagonal entries in $\mathcal{V}$ to be 1, so that the construction of matrices is done simultaneously. However, the optimality is lost in solving the local subproblems. The optimal choice of $\mathcal{V}$ is not discussed.

The paper is organized as follows. In section 2 two algorithms are devised for approximate factoring of the inverse. Section 3 is concerned with ways to choose the matrix subspace $\mathcal{V}$ optimally. Related stabilization schemes are suggested. In section 4 heuristic schemes are suggested for constructing the matrix subspace $\mathcal{W}$. In section 5 numerical experiments are conducted. The toughest benchmark problems from [3] are used in the tests.

2. Direct approximate factoring of the inverse. In what follows, two algorithms are devised for computing matrices W and V to have an approximate factorization

$$A^{-1} \approx WV^{-1} \tag{2.1}$$

[2] Accessing the entries of the adjoint can be costly in parallel computations.

of the inverse of a given sparse nonsingular matrix $A \in \mathbb{C}^{n\times n}$. The factors W and V are assumed to belong to given sparse standard matrix subspaces $\mathcal{W}$ and $\mathcal{V}$ of $\mathbb{C}^{n\times n}$. A matrix subspace is said to be standard if it has a basis consisting of standard basis matrices.[3] This allows maximal parallelizability by the fact that then the arising computational problems can be solved columnwise independently. Of course, parallelizability is imperative to fully exploit the processing power of modern computing architectures.

2.1. First basic algorithm, DIAF-Q. Consider the minimization problem (1.2) under the assumption that the columns of V are constrained to be unit vectors, i.e., of norm one. Based on the sparsity structure of $\mathcal{W}$ and the corresponding columns of A, the aim is at first choosing V optimally. Thereafter W is determined optimally.

To describe the method, denote by w_j and v_j the jth columns of W and V. The column v_j is computed first as follows. Assume there can appear $k_j \ll n$ nonzero entries in w_j at prescribed positions and denote by $A_j \in \mathbb{C}^{n\times k_j}$ the matrix with the corresponding columns of A extracted. Compute the sparse QR factorization

$$A_j = Q_j R_j \tag{2.2}$$

of A_j. (Recall that the sparse QR factorization is also needed in sparse approximate inverse computations.) Assume there can appear $l_j \ll n$ nonzero entries in v_j at prescribed positions and denote by $M_j \in \mathbb{C}^{k_j\times l_j}$ the matrix with the corresponding columns of Q_j^* extracted. Then v_j, regarded as a vector in $\mathbb{C}^{l_j}$, of unit norm is computed satisfying

$$||M_j v_j|| = ||M_j||\,, \tag{2.3}$$

i.e., v_j is chosen in such a way that its component in the column space of A_j is as large as possible. This can be found by computing the singular value decomposition of M_j. (Its computational cost is completely marginal by the fact that M_j is only a k_j-by-l_j matrix.)

Suppose the column v_j has been computed as just described for $j = 1, \ldots, n$. Then solve the least squares problems

$$\min_{w_j \in \mathbb{C}^{k_j}} ||A_j w_j - v_j||_2 \tag{2.4}$$

to have the column w_j of W.

For each pair v_j and w_j of columns, the computational cost consists of computing the sparse QR factorization (2.2) and, by using it, solving (2.3) and (2.4). For the sparse QR factorization there are codes available [8]. (Now A_j has the special property of being very "tall and skinny.")

The constraint of requiring the columns of V to be unit vectors is actually not a genuine constraint. That is, the method is scaling invariant from the right and thereby any nonzero constraints are acceptable in the sense that the condition (2.3) could equally well be replaced with $||M_j v_j|| = r_j||M_j||$. Let us formulate this as follows.

THEOREM 2.1. *Assume $A \in \mathbb{C}^{n\times n}$ is nonsingular. If $\mathcal{V}$ and $\mathcal{W}$ are standard matrix subspaces of $\mathbb{C}^{n\times n}$, then the factorization* (2.1) *computed as described is independent of the fixed column constraints $||v_j||_2 = r_j > 0$ for $j = 1, \ldots, n$.*

[3] Analogously to the standard basis vectors of $\mathbb{C}^n$, a standard basis matrix of $\mathbb{C}^{n\times n}$ has exactly one entry equaling one while its other entries are zeros.

Proof. Let W and V be the matrices computed with the unit norm constraint for the columns of V. Let $\hat{W}$ and $\hat{V}$ be computed with other strict positivity constraints for the columns of $\hat{V}$, i.e., (2.3) is replaced with the condition

$$||M_j v_j|| = r_j \, ||M_j|| \,. \tag{2.5}$$

Then we have $V = \hat{V}D$ and $W = \hat{W}D$ for a diagonal matrix D with nonzero entries. Consequently, $WV^{-1} = \hat{W}\hat{V}^{-1}$ whenever the factors are invertible. ∎

COROLLARY 2.2. *If a matrix V solving*

$$\min_{W \in \mathcal{W},\, V \in \mathcal{V},\, ||V||_F = 1} ||AW - V||_F \tag{2.6}$$

is nonsingular, then the factorization (2.1) *coincides with the one computed to satisfy* (2.3) *and* (2.4).

Proof. Suppose W and V solve (2.6). Since V is invertible, we have $||v_j||_2 = r_j > 0$. Using these constraints, compute $\hat{W}$ and $\hat{V}$ to satisfy (2.5) and (2.4). This means solving (2.6) columnwise and thereby the corresponding factorizations coincide. ∎

It is instructive to see how the computation of an approximate inverse relates with this. (For sparse approximate inverses and their historical development, see [1, section 5].)

Example 2.1. In approximate inverse computations, the matrix subspace $\mathcal{V}$ is as simple as possible, i.e., the set of diagonal matrices. Regarding the constraints, the columns are constrained to be unit vectors. Therefore one can replace $\mathcal{V}$ with the identity matrix, as is customary. See also Example 3.2 below.

2.2. Second basic algorithm, DIAF-S. Consider the minimization problem (1.2) under the assumption that the columns of W are constrained to be unit vectors instead. Based on the sparsity structure of $\mathcal{W}$ and the corresponding columns of A, the aim now is at first choosing W optimally. Thereafter V is determined optimally. The resulting scheme yields a direct analogue of the power method suggested in [4]. However, the method proposed here has at least three advantages. First, being direct, it seems to be more robust since there is no need to tune parameters used in the power method. Second, the Hermitian transpose of A is not needed. Third, the computational cost is readily predictable by the fact that, in essence, we only need to compute sparse QR factorizations.

To describe the method, denote by w_j and v_j the jth columns of W and V. The column w_j is computed first as follows. Assume there can appear $k_j \ll n$ nonzero entries in w_j at prescribed positions and denote by $A_j \in \mathbb{C}^{n \times k_j}$ the matrix with the corresponding columns of A extracted. Assume there can appear $l_j \ll n$ nonzero entries in v_j at prescribed positions and denote by $\hat{A}_j \in \mathbb{C}^{(n-l_j) \times k_j}$ the matrix with the corresponding rows of A_j removed. Then take w_j to be a right singular vector corresponding to the smallest singular value of $\hat{A}_j$.

To have w_j inexpensively, compute the sparse QR factorization

$$\hat{A}_j = Q_j R_j$$

of $\hat{A}_j$. (Here it might be wise to compute a QR factorization with column pivoting.) Then compute the singular value decomposition of R_j. Of course, its computational cost is completely negligible. (However, do not form the arising product to have the SVD of $\hat{A}_j$ explicitly.) Then take w_j from the singular value decomposition of R_j.

Suppose the column w_j has been computed as just described for $j = 1, \ldots, n$. Then, to have the columns of V, set

$$V = P_{\mathcal{V}} A[w_1 \cdots w_n],$$

i.e., nonzero entries are accepted only in the prescribed sparsity structure of v_j for $j = 1, \ldots, n$.

For an analogue of Corollary 2.2 we have the following proposition corresponding to these computations.

PROPOSITION 2.3. *Assume $A \in \mathbb{C}^{n\times n}$ is nonsingular. If a matrix W solving*

$$\min_{W\in\mathcal{W},\, ||W||_F=1,\, V\in\mathcal{V}} ||AW - V||_F \tag{2.7}$$

is nonsingular, then the factorization (2.1) *coincides with the one corresponding to the smallest singular value of the linear map* (1.3).

Proof. We have

$$\min_{W\in\mathcal{W},\, ||W||_F=1,\, V\in\mathcal{V}} ||AW - V||_F^2$$
$$= \min_{W\in\mathcal{W},\, ||W||_F=1,\, V\in\mathcal{V}} (||(I - P_{\mathcal{V}})AW||_F^2 + ||P_{\mathcal{V}}(AW - V)||_F^2).$$

The second term can always be made zero, regardless of how W is chosen. Therefore the minimum on the left correponds to the smallest singular value of the linear map (1.3). Since A is nonsingular, so is V whenever W is nonsingular. ∎

Since $\mathcal{W}$ and $\mathcal{V}$ are assumed to be standard matrix subspaces, the computations can be performed columnwise. (The corresponding constraint $||W||_F = \sqrt{n}$ is just a matter of scaling.) If there exists a nonsingular solution W, each column is necessarily nonzero and a diagonal scaling of W from the right gets canceled in the factorization.

2.3. Some general remarks. In approximate inverse preconditioning, it is well known that it can make a difference whether one computes a right or left approximate inverse [1, pp. 449–450]. As we have generalized this technique, this is the case with the approximate factoring of the inverse also. Here we have considered only preconditioning from the right.

The usage of standard matrix subspaces leads to maximal parallelizability. In view of approximating the inverse, this means that computations are done locally (columnwise) and independently, i.e., without any global control. To compensate for this, with an eye to improving the conditioning of the factors, it seems advisable to impose additional constraints. This is considered in section 3.

The simultaneous (somehow optimal) choice of the matrix subspaces $\mathcal{W}$ and $\mathcal{V}$ is a delicate matter. In [4] we gave a rule of thumb according to which the sparsity structures of the matrix subspaces should differ as much as possible in approximate factoring of the inverse. (This automatically holds in computing approximate inverses and ILU factorizations.) Numerical experiments seem to support this. Although we do not quite understand the reasons for this, it is partially related to the fact that then there are very few redundancies in the factorizations (2.1) as follows.

PROPOSITION 2.4. *Let $\mathcal{V}$ and $\mathcal{W}$ be standard nonsingular matrix subspaces of $\mathbb{C}^{n\times n}$ containing the identity. If in the complement of the diagonal matrices the intersection of $\mathcal{V}$ and $\mathcal{W}$ is empty, then the maximum rank of the map*

$$(V, W) \longmapsto WV^{-1} \tag{2.8}$$

on $\mathcal{V} \times \mathcal{W} \cap \mathrm{GL}(n, \mathbb{C})$ is $\dim \mathcal{V} + \dim \mathcal{W} - n$.

Proof. Linearize the map (2.8) at $(\hat{V}, \hat{W})$ for both $\hat{V}$ and $\hat{W}$ invertible. Using the Neumann series yields the linear term

$$\hat{W}(\hat{W}^{-1}W - \hat{V}^{-1}V)\hat{V}^{-1}.$$

At $(\hat{V}, \hat{W}) = (I, I)$ the rank is $\dim \mathcal{V} + \dim \mathcal{W} - n$. It is the maximum by the fact that for any nonsingular diagonal matrix D we have $(VD, WD) \longmapsto WV^{-1}$, i.e., the map (2.8) can be regarded as a function of $\dim \mathcal{V} + \dim \mathcal{W} - n$ variables. ∎

Aside from this basic principle, more refined techniques are devised for simultaneously choosing the matrix subspaces $\mathcal{W}$ and $\mathcal{V}$ in the sections that follow. Most notably, optimal ways of choosing $\mathcal{V}$ are devised.

3. Optimal construction of the matrix subspace $\mathcal{V}$ and imposing constraints. For the basic algorithms introduced, a method for optimally choosing the matrix subspace $\mathcal{V}$ is devised under the assumption that the matrix subspace $\mathcal{W}$ has been given. Moreover, mechanisms are introduced into the basic algorithms that allow for stabilizing the scheme for better conditioned factors. (In approximate inverse preconditioning the latter task is accomplished in the simplest possible way: the subspace $\mathcal{V}$ is simply $\mathbb{C}I$, i.e., scalar multiples of the identity.)

3.1. Optimally constructing the matrix subspace $\mathcal{V}$. Suppose the matrix subspace $\mathcal{W}$ has been given. Then the condition (2.3) yields a columnwise criterion for optimally choosing the sparsity structure of the matrix subspace $\mathcal{V}$. (Recall that it must be assumed that the nonsingular elements of $\mathcal{V}$ allow a rapid application of the inverse.) Once done, proceed by using one of the basic algorithms to compute the factors.

Consider (2.3). It is beneficial to choose the sparsity structure of v_j in such a way that the norm of M_j is as large as possible, with the constraint that in the resulting $\mathcal{V}$ the nonsingular elements are readily invertible. In other words, among admissible columns of Q_j^*, take l_j columns which yield M_j with the maximal norm. This means that for the optimization problem (1.2), with a fixed matrix subspace $\mathcal{W}$, the matrix subspace $\mathcal{V}$ is constructed in an optimal way.

Certainly, the problem of choosing l_j columns to maximize the norm is combinatorial and thereby rapidly finding a solution does not appear to be straightforward. A suboptimal choice for the matrix M_j can be readily generated by taking l_j admissible columns of Q_j^* with largest norms. When done with respect to the Euclidean norm, the Frobenius norm of the submatrix is maximized instead. This can be argued, of course, by the fact that

$$\frac{1}{\sqrt{\max\{k_j, l_l\}}} ||M_j||_F \leq ||M_j|| \leq ||M_j||_F$$

holds.

This approach starts with $\mathcal{W}$ and then yields $\mathcal{V}$ (sub)optimally. This process can be used to assess how $\mathcal{W}$ was initially chosen. Let us illustrate this with the following example.

Example 3.1. The choice of upper (lower) triangular matrices for $\mathcal{V}$ has the advantage that then we have a warning signal in case $\mathcal{W}$ is poorly chosen. Namely, suppose $\mathcal{V}$ has been (sub)optimally constructed as just described. If the factor V computed to satisfy (1.2) is poorly conditioned, one should consider updating the sparsity structure

of $\mathcal{W}$ to have a matrix subspace which is better suited for approximate factoring of the inverse of A.[4]

In this optimization scheme, let us illustrate how the matrix subspace $\mathcal{W}$ actually could be poorly chosen. Namely, the way the above optimization scheme is set up means that the sparsity structure of $\mathcal{W}$ should be such that no two columns share the same sparsity structure. (Otherwise V will have equaling columns.) Of course, this may be too restrictive. In the section that follows, a way to circumvent this problem is devised by stabilization.

3.2. Optimizing under additional constraints. There are instances which require imposing additional constraints on computing the factors. Aside from the problems described above, in tough problems the approximate factors may be poorly conditioned of even singular.[5] Because there holds

$$\frac{\left|\left|AWV^{-1}-I\right|\right|}{||V^{-1}||} \leq ||AW-V|| \leq \left|\left|AWV^{-1}-I\right|\right| ||V||\,, \tag{3.1}$$

this certainly cannot be overlooked. To overcome this, it is advisable to stabilize the computations by appropriately modifying the optimality conditions in computing the factors.

For the first basic algorithm this means a refined computation of V. Thereafter the factor W is computed columnwise as before to satisfy the conditions (2.4). For a case in which the conditioning is readily controlled, consider a matrix subspace $\mathcal{V}$ belonging to the set of upper (or lower) triangular matrices. Then, suppose the jth column v_j computed to satisfy (2.3) results in a tiny jth component. To stabilize the computations for the first basic algorithm, we replace v_j by first imposing the jth component of v_j to equal a constant $r_j > 0$. For the remaining components, let $\hat{M}_j$ be a submatrix consisting of the l_j-1 largest columns of Q_j^* among its first $j-1$ columns. Denote the jth column of Q_j^* by p_j. Then consider the optimization problem

$$\max_{||\tilde{v}_j||_2=1} \left|\left|r_j p_j + \hat{M}_j \hat{v}_j\right|\right|_2 . \tag{3.2}$$

By invoking the singular value decomposition $\hat{M}_j = \hat{U}_j \hat{\Sigma}_j \hat{V}_j^*$ of $\hat{M}_j$, this is equivalent to solving

$$\max_{||\tilde{v}_j||_2=1} \left|\left|r_j \tilde{p}_j + \hat{\Sigma}_j \tilde{v}_j\right|\right|_2 , \tag{3.3}$$

where $\tilde{p}_j = \hat{U}_j^* p_j$ and $\tilde{v}_j = \hat{V}_j^* \hat{v}_j$. Consequently, choose $\tilde{v}_j = (e^{i\theta}, 0, 0, \ldots, 0)$, where θ is the argument of the first component of $\tilde{p}_j$. (If the first component is zero, then any θ will do.) Set the column v_j to be the sum of $r_j e_j$ and the vector obtained after putting the entries of $\hat{V}_j \tilde{v}_j$ at the positions where the corresponding $l_j - 1$ largest columns of Q_j^* appeared. (Here e_j denotes the jth standard basis vector of $\mathbb{C}^n$.)

Observe that the solution does not depend on the value of $r_j > 0$. In particular, it is not clear how large r_j should be.

Again it is instructive to contrast this with the approximate inverse computations.

[4]This is actually the case in the (numerically) exact factoring: To recover whether a matrix $A \in \mathbb{C}^{n\times n}$ is nonsingular, it is advisable to compute its partially pivoted LU factorization, i.e., use a numerically reliable algorithm.

[5]This is a well-known phenomenon in preconditioning. For ILU factorization there are many ways to stabilize the computations [1]. Stabilization has turned out to be indispensable in practice.

Example 3.2. The sparse approximate inverse computations yield the simplest case of imposing additional constraints as just described. That is, the sparse approximate inverse computations can be interpreted as having $l_j = 1$ for every column, combined with imposing $r_j = 1$.

The LU factorization and thereby triangular matrices are extensively used in preconditioning. Because the LU factorization without pivoting is unstable, some kind of stabilization is needed. It is clear that the QR factorization also gives reasons to look at triangular matrices. The approach differs from that of using the LU factorization in that its computation does not require a stabilization, i.e., nothing like partial pivoting is needed. Of course, our intention is not to propose computing the full QR factorization. Understanding the Q factor is critical as follows.

Example 3.3. The QR factorization $A^* = QR$ of the Hermitian transpose of A can be used as a starting point to construct matrix subspaces for approximate factoring of the inverse. Namely, we have $AQ = R^*$. Therefore $\mathcal{V}$ belonging to the set of lower triangular matrices is a natural choice. For $\mathcal{W}$ one needs to generate an approximation to the sparsity structure of Q. For this there are many alternatives.

Aside from upper (lower) triangular matrices, there are, of course, completely different alternatives. Consider, for example, choosing V among diagonally dominant matrices. Since the set of diagonally dominant matrices is not a matrix subspace, dealing with this structure requires using constraints. It is easy to see that the problem can be tackled completely analogously, by imposing $r_j \geq \#\text{nz}$, where #nz denotes the number of nonzero elements outside the diagonal in v_j. Thereafter (3.2) is solved for having the other components in the column. The inversion of V can be performed by simple algorithms such as the Gauss–Seidel method.

4. Constructing the matrix subspace $\mathcal{W}$. Optimally constructing the matrix subspace $\mathcal{W}$ for approximate factoring of the inverse appears seemingly challenging. Some ideas are suggested in what follows, although no claims concerning the optimality are made. We suggest starting the process by taking an initial standard matrix subspace $\mathcal{V}_0$ which precedes the actual $\mathcal{V}$. Once $\mathcal{W}$ has been constructed, then $\mathcal{V}_0$ should be replaced with $\mathcal{V}$ computed with the techniques introduced in section 3.

It is noteworthy that SPAI [13] can also be regarded as an algorithm that generates $\mathcal{W}$ (as well as computes $W \in \mathcal{W}$). So SPAI could be used in generating $\mathcal{W}$ from which $\mathcal{V}$ is then generated for DIAF-Q and DIAF-S.

4.1. The Neumann series constructions. For approximate inverse computations the selection of an a priori sparsity pattern is a well-known problem [6, 2]. Good sparsity patterns are, at least in some cases, related to the transitive closures of subsets of the connectivity graph of $G(A)$ of A. This can also be interpreted as computing level set expansions on the vertices of a sparsified $G(A)$.

In [6] numerical dropping is used to sparsify $G(A)$ or its level set expansions. Denote by $v \in \mathbb{C}^n$ a vector with entries v_j. To select the relatively large entries of v numerically, entries are dropped by relative tolerance τ and by count p, i.e., only those entries of v_j of v for which it holds $|v_j| \geq \tau||v||_\infty$ with the restriction of p largest entries at most are stored. (Note that the diagonal elements are not subjected to numerical dropping.) In what follows, these rules are referred to as numerical dropping by tolerance and count.

The dropping can be performed on an initial matrix or during the intermediate phases of the level set expansion. Thus we have two sets of parameters (τ_i, p_i) controlling the initial sparsification and (τ_l, p_l) controlling the sparsification during level

set expansion. In addition, we adopt the convention that setting any parameter as zero implies that the dropping parameter is not used.

With these preparations for approximate factoring of the inverse, take an initial standard matrix subspace $\mathcal{V}_0$ and consider generating a sparsity pattern for $\mathcal{W}$. Assuming $V_0 = P_{\mathcal{V}_0}A \in \mathcal{V}_0$ is invertible, we have

$$A = V_0(I - V_0^{-1}(I - P_{\mathcal{V}_0})A) = V_0(I - S).$$

Whenever $||S|| < 1$, there holds $A^{-1} = (I + \sum_{j=1}^{\infty} S^j)V_0^{-1} = WV_0^{-1}$ by invoking the Neumann series. Therefore then

$$W = I + \sum_{j=1}^{\infty} S^j. \tag{4.1}$$

Although the assumption $||S|| < 1$ is generally too strict in practice, we may formally truncate the series (4.1) to generate a sparsity pattern. To make this economical and to retain $\mathcal{W}$ sparse enough, compute powers of S only approximately by using sparse-sparse operations combined with numerical dropping and level of fill techniques.

Observe that, to operate with the series (4.1) we need $S = V_0^{-1}(I - P_{\mathcal{V}_0})A$. It is this which requires setting an initial standard matrix subspace $\mathcal{V}_0$.

Example 4.1. For $S = V_0^{-1}(I - P_{\mathcal{V}_0})A$ we need to set an initial standard matrix subspace. The most inexpensive alternative is to take $\mathcal{V}_0$ to be the set of diagonal matrices. Then $V_0 = P_{\mathcal{V}_0}A$ is immediately found.

There are certainly other inexpensive alternatives for $\mathcal{V}_0$, such as block diagonal matrices. Once fixed, thereafter the scheme can be given as Algorithm 1 below.

ALGORITHM 1. SPARSIFIED POWERS FOR CONSTRUCTING $\mathcal{W}$.

```
Set a truncation parameter k
Compute V_0^{-1}
Compute S = V_0^{-1}(I - P_{V_0})A
Apply numerical dropping by tolerance τ_i and count p_i to columns of S
for columns j in parallel do
    Set s_j = t_j = e_j
    for l = 1, ..., k do
        Compute t_j = S t_j
        Apply numerical dropping by tolerance τ_l and count p_l to t_j
        Compute s_j = s_j + t_j
    end for
    Set sparsity structure of w_j to be the sparsity structure of s_j
end for
Set W = W \ {V_0 \ I}
```

Note that final step of Algorithm 1 is to keep the intersection of $\mathcal{W}$ and $\mathcal{V}_0$ empty apart from the diagonal; see section 2.3. After the sparsity structure for a matrix subspace $\mathcal{W}$ has been generated, the sparsity structure of $\mathcal{V}_0$ can be updated to be $\mathcal{V}$ by using $\mathcal{W}$.

4.2. Algebraic constructions. Next we consider some purely algebraic arguments which might be of use in constructing $\mathcal{W}$. Again start with an initial standard matrix subspace $\mathcal{V}_0$. Take the sparsity structure of the jth column of $\mathcal{V}_0$ and consider

the corresponding rows of $A \in \mathbb{C}^{n\times n}$. Choose the sparsity structure of the jth column of $\mathcal{W}$ to be the union of the sparsity structures of these rows. This is a necessary (but not sufficient) condition for $A\mathcal{W}$ to have an intersection with $\mathcal{V}_0$. This simply means choosing $\mathcal{W}$ to have the sparsity structure of $A^*\mathcal{V}_0$.

Most notably, the process is very inexpensive and can be executed in parallel. One only needs to control that the columns of $\mathcal{W}$ remain sufficiently sparse. With probability one, the following algorithm yields the desired sparsity structure.

ALGORITHM 2. COMPUTING A SPARSITY STRUCTURE FOR $\mathcal{W}$.

Require: A sparse matrix $A \in \mathbb{C}^{n\times n}$ and a random column $v_j \in \mathcal{V}_0$.
Ensure: Sparsity structure of the column w_j.

```
Compute w = A*v_j
if w is not sparse enough then
    Sparsify w to have the sparsity structure of w_j.
end if
Take the sparsity structure of w_j to be the sparsity structure of w.
```

Observe that we do not have $A^*\mathcal{V}_0 = \mathcal{W}$ since the computation is concerned with sparsity structures.

Approximate inverse preconditioning corresponds to choosing $\mathcal{V}_0$ to be the set of diagonal matrices. Then the sparsity structure of $\mathcal{W}$ equals that of A^*. The following two examples illustrate two extremes cases of this choice.

Example 4.2. Take $\mathcal{V}_0$ to be the set of diagonal matrices. Then the first basic algorithm reduces to the approximate inverse preconditioning. Algorithm 2 yields now a standard matrix subspace $\mathcal{W}$ whose sparsity structure equals that of A^*. This can yield very good results. If A has orthogonal rows (equivalently, columns) then and only then does this give exactly a correct matrix subspace $\mathcal{W}$ for factoring the inverse of A as $AWV^{-1} = I$ when $\mathcal{V}$ is taken to be $\mathcal{V}_0$.[6]

Having identified an ideal structure for the approximate inverse preconditioning when $\mathcal{W}$ is constructed with Algorithm 2, how about when A is far from being a scaled unitary matrix? An upper (lower) triangular matrix is a scaled unitary matrix only when it reduces to a diagonal matrix.

Example 4.3. Take again $\mathcal{V}_0$ to be the set of diagonal matrices. Then the basic algorithm reduces to the approximate inverse preconditioning. Algorithm 2 yields a standard matrix subspace $\mathcal{W}$ whose sparsity structure equals that of A^*. This yields very poor results if A is an upper (lower) triangular matrix. Namely, then its inverse is also upper (lower) triangular.

Algorithm 2 is set up in such a way that if $\mathcal{V}_0 \subset \tilde{\mathcal{V}}_0$, then $\mathcal{W} \subset \tilde{\mathcal{W}}$. Thereby matrix subspaces can be constructed to handle the two extremes of Examples 4.2 and 4.3 simultaneously.

In practice $\mathcal{V}_0$ should be more complex, i.e., the set of diagonal matrices is a too simple structure. One option is to start with $\mathcal{V}_0$ having the sparsity structure of the Gauss–Seidel preconditioner.

DEFINITION 4.1. *A standard matrix subspace $\mathcal{V}$ of $\mathbb{C}^{n\times n}$ is said to have the sparsity structure of the Gauss–Seidel preconditioner of $A \in \mathbb{C}^{n\times n}$ if the nonzero entries in $\mathcal{V}$ appear on the diagonal and there where the strictly lower (upper) triangular part of A has nonzero entries.*

[6] In view of this, it seems like a natural problem to ask how well A can be approximated with matrices of the form DU with D diagonal and U unitary.

TABLE 5.1
Matrices of the experiments, their application area, size, number of nonzeros, and density.

Problem	Area	n	$\mathrm{nz}(A)$	$k_1 = \mathrm{nz}(A)/n$
west1505	Chemical engineering	1505	5414	3.6
west2021	Chemical engineering	2021	7310	3.62
lhr02	Chemical engineering	2954	36875	12.5
bayer10	Chemical engineering	13436	71594	5.33
sherman2	PDE	1080	23094	21.4
gemat11	Linear programming	4929	33108	6.72
gemat12	Linear programming	4929	33044	6.7
utm5940	PDE	5940	83842	14.1
e20r1000	PDE	4241	131430	31

5. Numerical experiments. The purpose of this final section is to illustrate, with the help of five numerical experiments, how the preconditioners devised in sections 2 and 3 perform in practice. Since there is an abundance of degrees of freedom to construct matrix subspaces for approximate factoring of the inverse, only a very incomplete set of experiments can be presented. In particular, we feel that there is a lot of room for new ideas and improvements.

In choosing the benchmark sparse linear systems, we used the University of Florida collection [9]. The problems were selected to be the most challenging ones to precondition among those tested in [3]. For the matrices used and some of their properties, see Table 5.1. Assuming the reader has access to [3], the comparison between the methods proposed here and the diagonal Jacobi preconditioning, ILUT(0), ILUT(1), ILUT, and AINV can be readily made. Just in case the reader does not have access to these publications, a comparison with ILUT is made in Example 5.5. For a comparision between ILUs and AINV, see, e.g., [5].

Regarding preprocessing, in each experiment the original matrix has been initially permuted to have nonzero diagonal entries and scaled with `MC64`. (See [10] for `MC64`.) It is desirable that the matrix subspace $\mathcal{V}$ contains hierarchically connected parts of the graph of the matrix. To this end we use an approach to find the strongly connected subgraphs of the matrix; see Duff and Kaya [11]. We then obtain a permutation P such that after the permutations, the resulting linear system can be split as

$$Ax = (L + D + U)x = b, \tag{5.1}$$

where L^T and U are strictly block upper triangular and D is a block diagonal matrix. The construction of this permutation consumes at most $O(n \log(n))$ operations.[7]

In the experiments, the right-hand side $b \in \mathbb{C}^n$ in (5.1) was chosen in such a way that the solution of the original linear system was always $x = (1, 1, \ldots, 1)$. As in [3], we used BiCGSTAB [20] as a linear solver. The iteration was considered converged when the initial residual had been reduced by eight orders of magnitude.

The numerical experiments from Example 5.1 to Example 5.4 were carried out with MATLAB.[8] The experiments of Example 5.5 were carried out with a very preliminary implementation of the developed algorithms in FORTRAN 90.

Example 5.1. We compare the minimization algorithm presented in [4], PAIF, with the QR factorization based minimization algorithm of section 2.1 (DIAF-Q). We

[7] Preprocessing is actually a part of the process of constructing the matrix subspaces $\mathcal{W}$ and $\mathcal{V}$. That is, it is insignificant whether one orders correspondingly the entries of the matrix or the matrix subspaces.

[8] VersionR2010a.

TABLE 5.2
Comparison of PAIF *and* DIAF-*Q algorithms.*

				PAIF			DIAF-Q		
Problem	$\|D_j\|_M$	$\#D_j$	ρ	$\kappa(V)$	nrm	its	$\kappa(V)$	nrm	its
west1505	50	34	2.75	3.17E+04	3.59	18	1.85E+03	3.49	18
west2021	50	47	2.69	5.53E+03	3.84	23	3.33E+03	3.53	26
lhr02	50	66	1.11	1.65E+03	6.69	24	9.05E+02	7.01	32
bayer10	250	67	2.56	8.00E+05	22.27	56	2.50E+05	14.27	36
sherman2	50	24	1.05	4.32E+02	2.45	5	3.77E+02	1.84	5
gemat11	50	115	1.91	2.28E+05	3.58	109	1.50E+05	2.79	68
gemat12	50	114	1.91	5.96E+06	6.87	77	4.58E+06	5.20	77
utm5940	250	29	1.73	3.91E+06	14.66	295	1.84E+06	12.86	221
e20r1000	200	27	4.23	3.22E+06	13.67	465	4.44E+03	8.82	364

construct $\mathcal{W}$ with the heuristic Algorithm 1 of section 4.1. For all test matrices, we use $k = 3$ and $\tau_i = 1E-1$, $p_i = 0$, $\tau_l = 0$, and $p_l = 0$ as parameters. For PAIF, 40 refinement iterations were always used which is somewhat more than what we have found to be necessary in practice. However, we want to be sure that the comparison is descriptive in terms of the quality of the preconditioner.

We choose $\mathcal{V}$ to be the subspace of block diagonal matrices with block bounds and sparsity structure chosen according to the block diagonal part of A, i.e., the matrix D in (5.1). Then in the heuristic construction of $\mathcal{W}$ with Algorithm 1, V_0 is taken to be a diagonal matrix.

We denote by $|D_j|_M$ the maximum blocksize of $\mathcal{V}$ and by $\#D_j$ the number of blocks in $\mathcal{V}$ in total. Density of the preconditioner, denoted by ρ, is computed as $\rho = (\mathrm{nz}(W) + \mathrm{nz}(L_V) + \mathrm{nz}(U_V))/\mathrm{nz}(A)$, where $\mathrm{nz}(A)$, $\mathrm{nz}(W)$, $\mathrm{nz}(L_V)$, and $\mathrm{nz}(U_V)$ denote the number of nonzeroes in A, W, and the LU decomposition of V. For both PAIF and DIAF-Q, we also compute the condition number estimate $\kappa(V)$ and norm of the minimizer $||AW - V||_F$, denoted by nrm. Finally, its denotes the number of BiCGSTAB iterations. By † we denote no convergence of BiCGSTAB within 1000 iterations. Breakdown of BiCGSTAB is denoted by ‡. Table 5.2 shows the results.

Results very similar to those seen in Table 5.2 were also observed in other numerical tests that were conducted. As a general remark, the iteration counts with BiCGSTAB when preconditioned with DIAF-Q are not dramatically different from those achieved with PAIF. The main benefits of DIAF-Q are that neither the Hermitian transpose of A is required in the computations nor an estimate for the norm of A. Moreover, DIAF-Q is a direct method, so that its computational cost is easily estimated, while it is not so clear when to stop the iterations with PAIF.

The computational cost and parallel implementation of DIAF-Q are very similar to the established preconditioning techniques based on norm minimization for sparse approximate inverse. (For these issues, see [6].) That is, DIAF-Q scales essentially in terms of the computational cost and parallelizability properties.

Example 5.2. Next we compare PAIF with the SVD based algorithm of section 2.2 (DIAF-S). Again $\mathcal{W}$ is constructed with the heuristic Algorithm 1 of section 4.1. All the parameters were kept the same as in the previous example, i.e., $k = 3$ and $\tau_i = 1E-1$, $p_i = 0$, $\tau_l = 0$, and $p_l = 0$. Also, 40 refinement steps were again used in the power method, so that the results for PAIF are identical to those presented in Example 5.1.

Table 5.3 shows the results.

The results of Table 5.3 with DIAF-S are very similar to those in Table 5.2. The only notable exception is the matrix utm5940, for which no convergence was achieved

TABLE 5.3
Comparison of PAIF *and* DIAF-*S algorithms.*

				PAIF			DIAF-S		
Problem	$\|D_j\|_M$	$\#D_j$	ρ	$\kappa(V)$	nrm	its	$\kappa(V)$	nrm	its
west1505	50	34	2.75	3.17E+04	3.59	18	4.06E+03	3.04	14
west2021	50	47	2.69	5.53E+03	3.84	23	8.31E+03	3.26	27
lhr02	50	66	1.11	1.65E+03	6.69	24	1.90E+03	5.68	55
bayer10	250	67	2.56	8.00E+05	22.27	56	9.50E+05	11.68	46
sherman2	50	24	1.05	4.32E+02	2.45	5	4.33E+02	1.67	5
gemat11	50	115	1.91	2.28E+05	3.58	109	2.28E+05	2.90	113
gemat12	50	114	1.91	5.96E+06	6.87	77	1.51E+08	3.72	201
utm5940	250	29	1.73	3.91E+06	14.66	295	3.91E+06	7.43	‡
e20r1000	200	27	4.23	3.22E+06	13.67	465	1.96E+04	10.37	444

with DIAF-S. With the metrics used, we do not quite understand why DIAF-S fails to produce a good preconditioner for this particular problem. The computed norm $||AW - V||_F$ is smaller than the one attained with DIAF-Q and the condition number estimate is only slightly worse. The reason is most likely related to the fact that the bound (3.1) cannot be expected to be tight enough when $\kappa(V)$ is large.

The following example illustrates how the matrix subspace $\mathcal{V}$ can be optimally constructed with the techniques of section 3.

Example 5.3. In this example we consider an optimal construction of $\mathcal{V}$. To this end, we first construct $\mathcal{W}$ with the heuristic Algorithm 1 presented in section 4.1. Then, to construct $\mathcal{V}$, we apply the techniques presented in section 3. After the sparsity structures of the subspaces have been fixed, the resulting minimization problem is solved with DIAF-Q.

Consider the minimization problem (2.4). If no restrictions on the number of nonzero entries in a matrix subspace $\mathcal{V}$ are imposed, the norm $||AW - V||_F$ can be decreased by choosing as many entries as possible from the sparsity structure of AW to be in the sparsity structure of $\mathcal{V}$.[9] To illustrate this, we take $\mathcal{V}$ to be a subspace of block diagonal matrices by allowing only certain degree of sparsity $k_{\mathcal{V}}$ per column. The nonzero entries are chosen with the techniques of section 3.

We again set $k = 3$ and $\tau_i = 1E-1$, $p_i = 0$, $\tau_l = 0$, and $p_l = 0$ as parameters for all test matrices. To have the locations for the entries in the diagonal blocks of $\mathcal{V}$, we then apply the method presented in section 3. Subspace $\mathcal{W}$ is constructed with the heuristic Algorithm 1 by setting $\mathcal{V}_0$ to be a subspace of block diagonal matrices with full blocks. This is to ensure that intersection of the final $\mathcal{V}$ and $\mathcal{W}$ is empty.

Table 5.3 shows the results, where at most $k_{\mathcal{V}}$ entries in each column of the sparsity pattern of $\mathcal{V}$ have been allowed. For each test problem we have used the same block structure as in Examples 5.1 and 5.2, only the locations of the nonzero entries in $\mathcal{W}$ and $\mathcal{V}$ is varied.

As seen in Table 5.4, choosing more entries in $\mathcal{V}$, i.e., increasing $k_{\mathcal{V}}$ always improves the norm of the minimizer, which is well supported by the theory. Allowing more entries in $\mathcal{V}$ generally produces a better preconditioner. In a few cases where a slightly worse convergence can be observed, we also observe a slightly worse condition number estimate for the computed V.

The following example illustrates the optimal selection of a block upper triangular subspace $\mathcal{V}$ as well as optimization under additional constraints.

[9] For example, $\mathcal{V}$ can never be the full set of upper triangular matrices since it would require storing $O(n^2)$ complex numbers. The problem is, then, how to choose a subspace $\mathcal{V}$ of upper triangular matrices.

TABLE 5.4
Adaptive selection of $\mathcal{V}$ for different values of $k_{\mathcal{V}}$.

	$k_{\mathcal{V}} = 10$				$k_{\mathcal{V}} = 30$				$k_{\mathcal{V}} = 50$			
Problem	ρ	$\kappa(V)$	nrm	its	ρ	$\kappa(V)$	nrm	its	ρ	$\kappa(V)$	nrm	its
west1505	2.81	2.24E+03	3.54	20	2.83	4.57E+03	3.36	20	2.83	4.57E+03	3.36	20
west2021	2.75	5.11E+04	3.70	30	2.76	5.67E+04	3.46	31	2.76	5.67E+04	3.46	31
lhr02	1.15	1.07E+04	6.95	47	1.17	1.27E+04	6.92	43	1.17	1.27E+04	6.92	43
bayer10	2.68	6.72E+07	13.83	104	2.75	1.86E+05	13.42	29	2.76	1.86E+05	13.42	31
sherman2	1.01	1.49E+02	1.88	7	1.11	3.77E+02	1.82	5	1.11	3.77E+02	1.82	5
gemat11	2.14	1.50E+05	2.84	81	2.24	1.50E+05	2.75	68	2.24	1.50E+05	2.75	69
gemat12	2.09	4.24E+06	5.21	80	2.17	4.58E+06	5.13	71	2.17	4.58E+06	5.13	70
utm5940	2.11	1.82E+06	13.12	‡	2.49	2.09E+06	12.72	209	2.51	2.11E+06	12.71	201
e20r1000	3.77	2.16E+06	20.66	‡	4.99	1.01E+05	11.76	‡	5.49	6.67E+04	8.77	418

TABLE 5.5
Adaptive selection of $\mathcal{V}$ for different values of $k_{\mathcal{V}}$.

	$k_{\mathcal{V}} = 10$				$k_{\mathcal{V}} = 30$				$k_{\mathcal{V}} = 100$			
Problem	ρ	$\kappa(V)$	nrm	its	ρ	$\kappa(V)$	nrm	its	ρ	$\kappa(V)$	nrm	its
west1505	2.37	4.45E+06	4.79	860	2.68	1.88E+06	2.52	78	2.75	6.95E+04	2.42	24
west2021	2.31	1.66E+11	5.34	†	2.57	8.30E+05	2.53	67	2.66	3.25E+05	2.27	23
lhr02	1.03	2.45E+03	4.22	36	1.40	4.89E+03	3.21	22	1.46	4.93E+03	3.18	21
bayer10	2.09	3.74E+09	11.76	963	2.42	2.90E+06	8.19	390	2.49	3.39E+06	8.10	16
sherman2	0.89	1.61E+02	0.89	6	1.24	4.66E+02	0.63	3	1.24	4.66E+02	0.63	3
gemat11	1.84	1.50E+05	2.37	55	1.96	1.60E+05	1.74	32	1.97	1.60E+05	1.74	32
gemat12	1.82	1.45E+09	3.37	160	1.95	1.99E+09	2.38	39	1.96	2.03E+09	2.37	38
utm5940	1.60	5.29E+07	9.16	653	2.07	1.12E+09	8.17	141	2.18	8.56E+08	8.14	165
e20r1000	1.90	2.00E+11	22.32	†	2.77	6.59E+10	12.48	†	3.85	2.52E+09	6.05	792

Example 5.4. We consider optimal construction $\mathcal{V}$ in the case where $\mathcal{V}$ is block upper triangular. As parameters we again use $k = 3$ and $\tau_i = 1E-1$, $p_i = 0$, $\tau_l = 0$, and $p_l = 0$ and use the strongly connected subgraph approach to have a block structure for the subspace $\mathcal{V}$.

To have locations for the entries in the block upper triangular $\mathcal{V}$, we apply the method presented in section 3. As in Example 5.3, we construct $\mathcal{W}$ with Algorithm 1 by setting $\mathcal{V}_0$ as a subspace of block upper triangular matrices with full blocks. The resulting $\mathcal{W}$ is a lower triangular matrix subspace consisting of a diagonal part and a strictly block lower triangular part. The resulting minimization problem is solved with DIAF-Q. Note that by the structure of such a subspace, the conditioning of $W \in \mathcal{W}$ can be readily verified.

Table 5.5 shows the results, where at most $k_{\mathcal{V}}$ entries in each column in the block upper triangular part of $\mathcal{V}_0$ have been allowed. The used block structure is the same as in Examples 5.1, 5.2, and 5.3, only the locations and the number of the nonzero entries is varied in $\mathcal{W}$ and $\mathcal{V}$.

The results of Table 5.5 are very similar to those of Table 5.4. Again, allowing more entries in $\mathcal{V}$ always improves the norm of the minimizer and usually also produces a better preconditioner.

Similarly to Example 5.2, it is again hard to understand why $k_{\mathcal{V}} = 100$ produces a worse preconditioner than $k_{\mathcal{V}} = 30$ for utm5940. We attribute this behavior to the looseness of the bound (3.1), i.e., when $\kappa(V)$ is large, the minimization of $||AW - V||_F$ may not compensate sufficiently for this in these cases.

To improve the conditioning of $V \in \mathcal{V}$, we now consider the same problems using the technique of imposing constraints as described in section 3. As a constraint we

TABLE 5.6
Constrained selection of $\mathcal{V}$ for $k_{\mathcal{V}} = 100$.

Problem	ρ	$\kappa(V)$	nrm	stab	its
west1505	2.75	6.16E+04	3.06	1	25
west2021	2.66	1.63E+05	2.92	1	21
lhr02	1.46	4.93E+03	3.18	0	21
bayer10	2.49	3.39E+06	8.10	0	16
sherman2	1.24	4.66E+02	0.63	0	3
gemat11	1.97	1.60E+05	1.74	0	32
gemat12	1.96	2.03E+09	2.37	0	38
utm5940	2.18	7.56E+06	8.24	1	113
e20r1000	3.85	8.66E+09	6.27	3	738

require that for the diagonal entries v_{jj} of V it holds $|v_{jj}| \geq 1e-2$. In case the requirement is not met, we impose a constraint with $r = 2$. Table 5.6 describes the results for $k_{\mathcal{V}} = 100$, where the number of constrained columns is denoted by stab.

As seen in Table 5.6, if only a small number of columns have to be constrained, the technique is effective. In other numerical experiments not reported here we observed that if too many columns have to be constrained, the norm of the minimizer $||AW - V||_F$ tends to increase. An approach to find the right balance is needed then.

In this final example PAIF, DIAF-Q, and DIAF-S are compared with ILUT. Here one should bear in mind that these methods really belong to different categories by the fact that ILUT is a sequential algorithm. In a sense, this means comparing "apples and oranges." Moreover, these algorithms are implemented in FORTRAN 90.

Example 5.5. We compare PAIF, DIAF-Q, and DIAF-S with ILUT with RCM preprocessing. For all matrices, we use $k = 4$ and $\tau_i = 1E-2$, $p_i = 0$, $\tau_l = 0$. For most matrices we set $p_l = 20$, except for bayer10 and e20r1000, where we set $p_l = 40$ and $p_l = 50$, respectively. For PAIF 40 refinement iterations were taken to compute a preconditioner. For ILUT, we choose drop tolerance as $\tau = 1E-2$ and level of fill as $p = 10$. Again the exceptions are bayer10 and e20r1000, where we use $p = 20$ and $p = 75$, respectively.

The sparsity structures for subspaces $\mathcal{W}$ and $\mathcal{V}$ were chosen as in Examples 5.1 and 5.2, i.e., we construct $\mathcal{W}$ with the heuristic Algorithm 1 and choose $\mathcal{V}$ to be the subspace of block diagonal matrices with sparsity structure chosen according to the block diagonal part of A.

We use a very preliminary implementation of the algorithms with FORTRAN 90. With PAIF, DIAF-Q, and DIAF-S we treat the small system A_j or $\hat{A}_j$ as a dense matrix after the nonzero indices have been locally renumbered. With them we use standard BLAS and LAPACK routines taken from the Intel Math Kernel Library.[10] For an implementation of ILUT, we use Sparskit [18]. For BiCGSTAB and block linear system solution with $\mathcal{V}$, we use routines MI26 and MA48 from the Harwell Subroutine Library [12].

To compile the codes, we use the Intel Fortran compiler,[11] with flags `-O3 -fp-model precise`.

We denote by t_p the preconditioning time and t_s the solution time of BiCGSTAB. Both t_p and t_s are in seconds on a single core of Intel Xeon E5-2670 processor. As in the previous examples, its denotes the number of BiCGSTAB iterations. Prepro-

[10] Version 11.
[11] Version 13.

TABLE 5.7
Comparison of PAIF, DIAF-*Q*, DIAF-*S*, *and* ILUT *algorithms.*

	PAIF			DIAF-Q			DIAF-S			ILUT		
Problem	t_p	t_s	its	t_p	t_s	its	t_p	t_s	its	t_p	t_s	its
west1505	3.80E-02	6.00E-03	21	3.50E-02	4.00E-03	25	5.20E-02	4.00E-03	28	1.00E-03	7.00E-03	33
west2021	3.10E-02	7.00E-03	26	3.10E-02	6.00E-03	32	5.40E-02	6.00E-03	26	1.00E-05	2.00E-03	28
lhr02	1.44E-01	1.60E-02	33	1.26E-01	1.60E-02	43	3.15E-01	1.20E-02	35	1.00E-03	7.00E-03	30
bayer10	7.33E-01	1.08E-01	66	9.36E-01	6.20E-02	39	3.02E+00	8.20E-02	51	6.00E-03	2.50E-02	22
sherman2	6.10E-02	1.00E-05	5	8.30E-02	1.00E-05	3	1.45E-01	1.00E-05	3	9.90E-05	1.00E-05	5
gemat11	1.44E-01	5.50E-02	79	1.78E-01	3.10E-02	45	4.25E-01	5.20E-02	74	1.00E-03	2.00E-02	46
gemat12	1.34E-01	1.09E-01	156	1.71E-01	7.20E-02	102	4.03E-01	4.40E-02	63	1.00E-03	2.00E-02	46
utm5940	4.07E-01	3.95E-01	432	4.82E-01	1.53E-01	166	1.07E+00	2.27E-01	247	9.00E-03	3.13E-01	521
e20r1000	2.50E+00	5.40E-01	393	4.34E+00	6.87E-01	499	6.96E+00	-	†	9.20E-02	-	†

cessing times were not taken into account. (For the preprocessing times of `MC64`, see [10].) Table 5.7 shows the results.

As can be seen, the performance of the PAIF, DIAF-Q, and DIAF-S preconditioners is quite similar to the ILUT preconditioner, when only iteration counts are taken into account. As for the computing times, the construction times for the preconditioner can be one or two order of magnitudes smaller for ILUT than for those of PAIF, DIAF-Q, or DIAF-S. This is something to be expected in a sequential environment; see, e.g., [19]. The iteration times are generally of the same order of magnitude when the number of iterations is roughly the same.

The parameters of the numerical dropping were chosen such that the relative densities of the preconditioners were roughly equal for PAIF, DIAF-Q, DIAF-S, and ILUT. With the selected parameters, the mean relative density was $\rho = 2.6$ for our approximate inverse preconditioners. To guarantee that the computations were not biased in favor of our algorithms, for ILUT we allowed it to be $\rho = 3.7$, mainly due to the high relative density of the ILUT preconditioner for the matrix bayer10.

Let us emphasize that the main advantage of PAIF, DIAF-Q, and DIAF-S is parallelism, available both in the construction and application phases of the preconditioner, which is not reflected in these single core experiments. By our experience treating the small systems as dense matrices is rather inefficient and generally not recommended for a production code. For instance, for the matrix e20r1000 the local matrices A_j were quite large in size but sparse, having only 9.8 nonzeroes per row on average. Treating the local systems as full matrices raised the relative density to about 86.1. Thus it is reasonable to expect the computation times to be lower if the systems are treated as sparse. It is nevertheless useful to have a worst case complexity for the algorithms in order to make comparison with the established techniques easier.

To sum up these numerical experiments, the iteration counts obtained with DIAF-Q and DIAF-S (which are fully parallelizable) seem to be competitive with the iteration counts obtained with the standard algebraic (sequential) preconditioning techniques. Moreover, a good problem specific tuning of matrix subspaces possesses a lot of potential for significantly speeding up the iterations.

Acknowledgments. The authors would like to thank Kamer Kaya and Iain Duff for their strongly connected components code and the reference [11]. A large part of this work was done when the first author was a postdoc researcher in CERFACS and would like to thank the ALGO group for hospitality and a nice atmosphere.

We also thank the referees for useful remarks and for bringing up many relevant references.

REFERENCES

[1] M. Benzi, *Preconditioning techniques for large linear systems: A survey,* J. Comput. Phys., 182 (2002), pp. 418–477.
[2] M. Benzi, L. Giraud, and G. Alléon, *Sparse approximate inverse preconditioning for dense linear systems arising in computational electromagnetics*, Numer. Algorithms, 16 (1997), pp. 1–15.
[3] M. Benzi, J. C. Haws, and M. Tuma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM J. Sci. Comput., 22 (2000), pp. 1333–1353.
[4] M. Bycкling and M. Huhtanen, *Approximate factoring of the inverse*, Numer. Math., 117 (2011), pp. 507–528.
[5] M. Bollhöfer amd Y. Saad, *On the relations between ILUs and factored approximate inverses*, SIAM J. Matrix Anal. Appl., 24 (2002), pp. 219–237.
[6] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.
[7] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations,* SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.
[8] T. A. Davis, *Direct Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2006.
[9] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), 1.
[10] 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 (2001), pp. 973–996.
[11] I. S. Duff and K. Kaya, *Preconditioners based on strong components*, Electron. Trans. Numer. Anal., 40 (2013), pp. 225–248.
[12] HSL, *HSL* 2013: *A Collection of Fortran Codes for Large Scale Scientific Computation*, http://www.hsl.rl.ac.uk (2013).
[13] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses,* SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
[14] R. M. Holland, A. J. Wathen, and G. J. Shaw, *Sparse approximate inverses and target matrices,* SIAM J. Sci. Comput., 26 (2005), pp. 1000–1011.
[15] T. Huckle and A. Kallischko, *Frobenius norm minimization and probing for preconditioning,* Int. J. Comput. Math., 84 (2007), pp. 1225–1248.
[16] M. Huhtanen, *Factoring matrices into the product of two matrices*, BIT, 47 (2007), pp. 793–808.
[17] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[18] Y. Saad, *SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations*, http://www-users.cs.umn.edu/~saad/software/SPARSKIT/.
[19] N. I. M. Gould and J. A. Scott, *Sparse approximate-inverse preconditioners using norm-minimization techniques,* SIAM J. Sci. Comput., 19 (1998), pp. 605–625.
[20] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.