A Factored Approximate Inverse Preconditioner With Pivoting

Matthias Bollhöfer* Yousef Saad†

Abstract

In this paper we develop new techniques for stabilizing factored approximate inverse preconditioners (AINV) using pivoting. This method yields stable preconditioners in many cases and can provide successful preconditioners in many situations when the underlying system is highly indefinite. Numerical examples illustrate the effectiveness of this approach.

Keywords. sparse matrices, ILU, sparse approximate inverse, AINV, pivoting.
AMS specification. 65F05, 65F10, 65F50.

1 Introduction

Many applications lead to solving large sparse linear systems of the form

$$Ax = b, \tag{1}$$

with $A \in \mathbb{R}^{n,n}$ and $b \in \mathbb{R}^n$. In many cases, such systems are not only very large but also exceedingly difficult to solve by iterative techniques because A is ill-conditioned or highly indefinite or both. In some instances these equations arise from special applications and solvers tailored to the underlying physical problem may give the best results. However, there are situations in which 'general purpose' solvers are desirable. Such is the case when building general purpose software, or when the linear system has very little inherent structure. General purpose solvers have many other advantages, the most significant being that changes in the physics or model do not require the development of new methods. For these situations, preconditioned Krylov–subspace solvers, see, e.g., [16, 25, 13] are often seen as promising alternatives to 'black-box' direct solution methods. Among all preconditioning techniques, those based on incomplete LU–factorizations, see e.g. [20, 21, 22] are known to give excellent results for many important classes of problems, such as those arising from the discretization of elliptic partial differential equations.

Motivated by the emergence of parallel computing platforms, a number of new techniques have been developed in recent years, which approximate directly the inverse of A. A few of these approaches are based on minimizing the norm $\|I - AM\|$ in some appropriate norm [19, 17, 15, 9] while others approximately solve the equation $W^\top AZ = D$, where the unknown matrices Z, W are unit upper triangular and D is a diagonal matrix, see e.g. [24, 4, 5, 1, 18]. In particular, the algebraic behavior of the latter class of methods bears strong similarities to that of incomplete LU–decompositions, e.g. they are stable for M– and H–matrices. See [6] for a detailed analysis between factored approximate inverses and incomplete LU–decompositions. Without describing the details of these relations we briefly sketch both methods to describe these links.

*Institute of Mathematics, MA 4–5, Berlin University of Technology, D–10623 Berlin, Germany. Supported by grants of the DFG BO 1680/1-1 and by the University of Minnesota. This research was performed while visiting the University of Minnesota at Minneapolis. email: bolle@math.tu-berlin.de, URL: http://www.math.tu-berlin.de/∼bolle/.

†Dep. of Computer Science and Engineering, University of Minnesota, 4–192 EE/CSci Building, 200 Union St., SE, Minneapolis, MN 55455–0154. Work supported by NSF and by the Minnesota Supercomputing Institute. email: saad@cs.umn.edu, URL: http://www.cs.umn.edu/∼saad/

For solving the linear system (1), incomplete LU techniques begin by approximately constructing a factorization

$$A \approx LDU,$$

where $L, U^\top$ are lower triangular matrices, with unit diagonal, and D is diagonal. One way to construct these decompositions is to partition A as

$$A = \begin{bmatrix} B & F \\ E & C \end{bmatrix} \in \mathbb{R}^{n,n}$$

with $B \in \mathbb{R}$ and the other blocks have corresponding size. Then A is factored as

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ L_E & I \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} D_B & 0 \\ 0 & S \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} 1 & U_F \\ 0 & I \end{bmatrix}}_{U}, \tag{2}$$

where

$$S = C - L_E D_B U_F \in \mathbb{R}^{n-k,n-k} \tag{3}$$

denotes the so–called Schur–complement. The exact LU–decomposition of A (if it exists) can be obtained by successively applying (2) to the Schur–complement S. Even if there exists a decomposition (2) for A and for S, there is no need to compute L_E and U_F, S exactly when constructing a preconditioner. A common approach for reducing fill-in consists of discarding entries of L_E and U_F of small size and defining the approximate Schur–complement only with these sparsified vectors $\tilde{L}_E$ and $\tilde{U}_F$. We will use

$$\tilde{S} = B - \tilde{L}_E D_B \tilde{U}_F \tag{4}$$

as one possible definition of an approximate Schur–complement. The associated ILU algorithm is roughly given by Algorithm 1.

Algorithm 1 (Incomplete LU factorization (ILU))
Let $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$ and let $\tau \in (0,1)$ be a drop tolerance. Compute $A \approx LDU$.

Set $L = U = I$, $S = A$

for $i = 1, \ldots, n$
  $d_{ii} = s_{ii}$
  for $j = i+1, \ldots, n$
    $p_j = s_{ji}/d_{ii}, \quad q_j = s_{ij}/d_{ii}$
    drop entries $|p_j|, |q_j|$ if they are less than τ
    $l_{ji} = p_j, \quad u_{ij} = q_j$
    for $k = i+1, \ldots, n$
      $s_{kj} = s_{kj} - l_{ki} d_{ii} u_{ij}$
    end
  end
end

Practical versions of incomplete LU decompositions are typically implemented in a slightly different way. It is usually not advisable to update the whole matrix $\hat{S} = (s_{kl})_{k,l \geqslant i}$ by a rank–1 modification. Instead, the leading row of $\hat{S}$ is typically computed, and the transformations on the other rows are post–poned. In essence this means that the so–called I,K,J version of Gaussian elimination is used. For details, see [24]. In addition to saving memory, this has the advantage that all updates and modifications are performed only once for each row, thus making it possible to use very simple sparse row storage schemes such as the Compressed Sparse Row (CSR) format.

In [4, 5] algorithms have been presented that directly compute upper unit triangular matrices W and Z such that $W^\top AZ \approx D$ is approximately diagonal. Here we choose to outline a version (Algorithm 2) that has already been used for the symmetric positive definite case [1, 18]. In short, at any given step i the algorithm performs a Gram–Schmidt step to A–orthogonalize the columns $w_j, j = i+1, \ldots, n$ against z_i and then the columns $z_j, j = i+1, \ldots, n$ against w_i. Dropping is then applied to the resulting columns. Clearly, A–orthogogonality is only achieved approximately.

Algorithm 2 (Factored Approximate Inverse (AINV))
Let $A = (a_{ij})_{ij} \in \mathbb{R}^{n,n}$ and let $\tau \in (0,1)$ be a drop tolerance. Compute $A^{-1} \approx ZD^{-1}W^\top$.

Set $p = q = (0, \ldots, 0) \in \mathbb{R}^n$, $Z = [z_1, \ldots, z_n] = I_n$, $W = [w_1, \ldots, w_n] = I_n$.

for $i = 1, \ldots, n$
  $d_{ii} = w_i^\top A z_i$
  for $j = i+1, \ldots, n$
    $p_j = \left(w_j^\top A z_i\right)/d_{ii}, \quad q_j = \left(w_i^\top A z_j\right)/d_{ii}$
    $w_j = w_j - w_i\, p_j, \quad z_j = z_j - z_i\, q_j$
    for all $l \leqslant i$: drop entries w_{lj}, z_{lj} if their absolute values are less than τ.
  end
end

Suppose that Algorithm 1 and Algorithm 2 do not break down. In step i of Algorithm 1, entries in the i–th row and i–th column of the Schur–complement $(s_{kl})_{k,l \geqslant i}$ are eliminated. Analogously Algorithm 2 eliminates the off–diagonal entries in row i and column i of $\left(w_k^\top A z_l\right)_{k,l \geqslant i}$. If no dropping is applied, then we would obtain

$$(s_{kl})_{k,l \geqslant i} = \left(w_k^\top A z_l\right)_{k,l \geqslant i}. \tag{5}$$

This can been seen, e.g., from (2) using $L^{-1} = W^\top$ and $U^{-1} = Z$. This means that p_j and q_j (resp. d_{ii}) play similar roles in both algorithms.

The main advantage of this observation is the possibility of exploiting these connections to adapt pivoting techniques used in (incomplete) Gaussian elimination, and to carry them over to sparse–approximate inverse techniques, with the goal of improving the performance of factored approximate inverses.

2 Approximate Inverses with Pivoting

One way to exploit the connection between approximate inverses and incomplete LU factorizations is to introduce pivoting to approximate inverses. This can be done by first adding pivoting to Algorithm 1 and then using relation (5) to transfer pivoting strategies to Algorithm 2. The main reason for introducing pivoting to Algorithm 1 and Algorithm 2 is the fact that the algorithms might encounter a zero or small pivot during the computation. At some step i of either algorithm it may turn out that $d_{ii} \approx 0$, in which case the algorithms break down. It is possible to shift the zero pivots away by adding an artificial small perturbation (e.g., 10^{-8}) to d_{ii} but this will rarely solve the problem. Instead, we could ensure that zero pivots do not occur and this is traditionally achieved by pivoting in direct Gaussian elimination. This technique was implemented in incomplete factorizations as well [22]. Although this makes the underlying data structure more complex, it often stabilizes the processes and even ensures that the growth in the element size of L, U will remain fairly moderate. However unlike complete Gaussian elimination, ILU with pivoting might still break down.

We first discuss how column and row pivoting could be added to Algorithm 1. We will introduce column and row interchanges that keep the algorithm consistent when $\tau = 0$ is used. In other words, the algorithm without dropping will compute $\Pi^\top A \Sigma = LDU$, where Π and Σ are permutation matrices that will be determined throughout the process. If π and σ are permutation

vectors associated with permutation matrices Π and Σ, then we will sometimes write $A(\pi, \sigma)$ for the permuted matrix $\Pi^\top A \Sigma$.
Suppose that a diagonal pivot s_{ii} is not satisfactory. The property to be satisfied by, say, a column pivot $k \geq i$ at step i, could be a criterion such as

$$|s_{ik}| \geq \alpha \, |s_{ij}| \quad \text{for} \quad j \geq i,$$

for a prescribed constant $0 < \alpha \leqslant 1$. After the column interchange takes place, one could consider in addition the analogous row criterion

$$|s_{ii}| \geq \alpha \, |s_{ji}| \quad \text{for} \quad j \geq i.$$

If this inequality is no longer satisfied, a row interchange will be performed. This process can alternate between both criteria and usually takes a few steps to complete, in many cases requiring just one step. It allows a better selection by iterating on the choice of the pivots, if necessary, without entailing substantial additional cost in most cases. With pivoting, (2) locally changes to

$$\underbrace{\begin{bmatrix} I & \mathrm{O} \\ \mathrm{O} & \hat{\Pi} \end{bmatrix}^\top}_{\Pi^\top} \begin{bmatrix} B & F \\ E & C \end{bmatrix} \underbrace{\begin{bmatrix} I & \mathrm{O} \\ \mathrm{O} & \hat{\Sigma} \end{bmatrix}}_{\Sigma} \approx \begin{bmatrix} I & \mathrm{O} \\ \hat{\Pi}^\top L_E & I \end{bmatrix} \begin{bmatrix} D_B & \mathrm{O} \\ \mathrm{O} & \hat{\Pi}^\top S \hat{\Sigma} \end{bmatrix} \begin{bmatrix} I & U_F \hat{\Sigma} \\ \mathrm{O} & I \end{bmatrix}, \tag{6}$$

where equality holds, if no dropping is applied. The approximate identity (6) shows that the columns of $U - I$ also need to be interchanged with respect to the column pivoting step. The rows of $L - I$ need to be processed similarly if row pivoting is applied. The additional row pivoting step is not so common in practice.

This may be more expensive due to the additional overhead for computing not only the leading row of the Schur–complement, but also its leading column. In particular, we note that this version of pivoting is hard to implement with the common IKJ variant of Gaussian elimination, since columns of the Schur complement are not available and their corresponding data structure expensive to obtain.

After showing how pivoting affects Algorithm 1 it is now easy to extend naturally the idea of pivoting to Algorithm 2. To do so we only have to keep in mind that $W = L^{-\top}$ and $Z = U^{-1}$ if $\tau = 0$ is used. Clearly the columns of $W - I$ and $Z - I$ have to be permuted analogously to the rows of $L - I$ and columns of $U - I$. If no dropping is applied then equation (5) now reads

$$(s_{kl})_{k,l \geqslant i} = \left(w_k^\top A(\pi, \sigma) z_l\right)_{k,l \geqslant i}. \tag{7}$$

This restricts the application of π and σ to the initial matrix A, if we reorder columns of $Z - I$ and $W - I$. This leads to Algorithm 3.

Note that the while–loop in Algorithm 3 is optional and has been included for the purpose of greater generality. If no dropping is applied, we obtain $W^\top A(\pi, \sigma) Z = D$, by construction. Pivoting for a related direct projection method can already be found in [3]. Instead of using $W^\top A Z$ to compute p and q only $W^\top A$ is used, which is equivalent in this case because no dropping is applied. In a sense Algorithm 3 generalizes the pivoting approach of [3] in that it is applied to an incomplete factorization and both row and column interchanges are performed.

As it is described, Algorithm 3 does not specify any rule on how to select the pivots k and l. A reasonable strategy could be to choose k such that $|p_k|$ is maximal and this obviously requires p to be computed before pivoting is applied. We observe that in Algorithm 3 now the p and q columns are inside the while loop which searches for adequate pivots. Indeed, p and q must be recomputed whenever q (resp. p) requires an interchange. In the situation when one pivoting step is applied for any i, then there is no need to recompute p and q. However, when more than one pivoting step is required, one of p or q at least must be recomputed. In this case the algorithm incurs some additional overhead. Clearly, any pivoting strategy should try to keep this additional overhead small. For better stability in Algorithm 3 the pivot p_k should satisfy:

$$|p_k| \geqslant \alpha \, \max_m |p_m|$$

Algorithm 3 (Factored Approximate Inverse With Pivoting (AINVP))
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$ and let τ be drop tolerance. Compute $A^{-1} \approx ZD^{-1}W^\top$.

Set $p = q = (0, \ldots, 0) \in \mathbb{R}^n$, $Z = [z_1, \ldots, z_n] = I_n$, $W = [w_1, \ldots, w_n] = I_n$ and $\pi = \sigma = (1, \ldots, n)$.

for $i = 1, \ldots, n$
  while *pivots not satisfactory*
    for all $j \geqslant i$: $p_j = w_j^\top A(\pi, \sigma) z_i$.
    Find a column pivot $k \geqslant i$.
    Interchange columns i, k of $Z - I$ and components i, k of p and σ.
    for all $j \geqslant i$: $q_j = w_i^\top A(\pi, \sigma) z_j$.
    Find a row pivot $l \geqslant i$.
    Interchange columns i, l of $W - I$ and components i, l of q and π.
  end
  $d_{ii} = p_i$.
  for $j = i + 1, \ldots, n$
    $p_j = p_j / d_{ii}, \quad q_j = q_j / d_{ii}$.
    $w_j = w_j - w_i p_j, \quad z_j = z_j - z_i q_j$.
    for all $l \leqslant i$: drop entries w_{lj}, z_{lj}, if their absolute values are less than τ.
  end
end

for some constant $0 < \alpha \leqslant 1$, e.g. $\alpha = 0.1$. However, since Algorithm 3 is a biorthogonalization technique, i.e. the outcome is to compute Z **and** W, we should ensure that $|p_k| = |q_k| \geqslant \alpha \max_m |q_m|$ is also fulfilled to guarantee that the entries of both factors, Z and W are sufficiently bounded. Algorithm 4 gives us a simple and relatively inexpensive strategy for controlling the growth of the entries in Z and W and to stabilize Algorithm 3.

An additional improvement to prevent too many pivoting steps might be to pre–scale the rows and/or columns of A. As a rule, more than one pivoting step, say column pivoting, is performed in any given step of Algorithm 3. However, in order for $|p_i| \geqslant \alpha \max_m |p_m|$ to imply that $|p_i| = |q_i| \geqslant \alpha \max_m |q_m|$ it is necessary that entries of q have magnitudes that are comparable with those of p. By (7), p and q are the first column/row of

$$\left(w_k^\top A(\pi, \sigma) z_l\right)_{k,l \geqslant i}$$

before a further step of pivoting is applied. If W and Z are moderately bounded, which is more or less achieved by pivoting, then $A(\pi, \sigma)$ having rows of comparable absolute row sums might be a good start to prevent too many pivoting steps.

One might of course think of other pivoting strategies, especially in the context of parallel computations. Pivoting is undoubtedly harder to implement in parallel. As is often done however, it is possible to exploit relaxed pivoting to search for satisfactory pivots locally, i.e., in each processor. This means that k and l are restricted to a certain subset to maintain distributed storage schemes. Other strategies could be for example to restrict k and l in order to keep the lower right $(n - i) \times (n - i)$ part of $W^\top A(\pi, \sigma) Z$ as sparse as possible.

3 Numerical Results

This section presents numerical experiments to validate the algorithms. Additional details and comments on the implementations of the algorithms will also be provided.

- All input matrices are assumed to be given in the CSR format [24].

Algorithm 4 (Controlled Pivoting)
Prescribe a tolerance $\alpha \in (0,1]$, *e.g.* $\alpha = 0.1$

satisfied_p=false, satisfied_q=false
while *not satisfied_p*
  for all $j \geqslant i$: $p_j = w_j^\top A(\pi,\sigma) z_i$
  if $|p_i| < \alpha \max_m |p_m|$
    satisfied_q=false, choose k *such that* $|p_k| = \max_m |p_m|$.
    Interchange column i *and* k *of* $Z - I$ *and components* i *and* k *of* σ.
  end
  satisfied_p=true
  if *not satisfied_q*
    for all $j \geqslant i$: $q_j = w_i^\top A(\pi,\sigma) z_j$
  end
  if $|q_i| < \alpha \max_m |q_m|$.
    satisfied_p=false, choose l *such that* $|q_l| = \max_m |q_m|$
    Interchange column i *and* l *of* $W - I$ *and components* i *and* l *of* π.
  end
  satisfied_q=true
end

- The matrices are initially scaled such that they have unit 1–norm for any row. As mentioned in Section 2 this is done to reduce the number of necessary column/row interchanges.
- $W^\top$ and $Z^\top$ are stored in CSR format.
- Interchanges of columns of W (resp. Z) are performed by only interchanging the references (pointers) instead of the whole data array.
- The computation of p and q in Algorithm 3 requires a multiplication $A(\pi,\sigma)z_i$, $A(\pi,\sigma)^\top w_i$. To be efficient, this operation must be done in sparse–sparse mode. If A is given in CSR format only $A(\pi,\sigma)^\top w_i$ is easy to access, while $A(\pi,\sigma)z_i$ requires $A^\top$ to be stored in CSR format. For this purpose we initially compute the pattern of $A^\top$ in CSR format but we omit the numerical values. The nonzero components of the computed vectors $A(\pi,\sigma)z_i$, $A(\pi,\sigma)^\top w_i$ are stored as a full vector but with an additional index list of the nonzeros. The index list is important to inherit the sparse nature of this vector, when the computations are performed. Otherwise dense computations would slow down the algorithm. See e.g. [24], p.291 for this kind of technique. Finally to compute for all $j \geqslant i$, $w_j^\top(A(\pi,\sigma)z_i)$ and $z_j^\top(A(\pi,\sigma)^\top w_i)$ we use a list which contains the non–trivial columns of W and Z, i.e., those columns which contain more than the diagonal entry only. The use of permutation vectors π and σ requires to have the inverse permutations π^{-1}, σ^{-1} which are computed simultaneously.
- Two values were used for the parameter α which controls the pivoting process: $\alpha = 0.1$ and $\alpha = 1.0$.
- Two different values were used for the drop tolerance $\tau = 0.1$, and $\tau = 0.01$.

For the numerical experiments several collections were chosen from the Harwell–Boeing Collection [11], the SPARSKIT Collection [23], and finally from the Davis collection [10].
Throughout the computations the matrices were initially reordered using the symmetric minimum degree ordering [14]. But clearly for specific problems other orderings can be more beneficial (cf. [2]). Note that for unsymmetric matrices other orderings might be used. See e.g. [7] for a reordering (MIP) for approximate inverses.

The computations were performed on an IBM RS6000 (44P model 270) under AIX 4.3 and 4GB memory. The approximate inverse algorithms were implemented in C. Dynamic memory allocation in C is a flexible and convenient tool to use relative to a 'manual' memory management that would be required under standard FORTRAN 77. However, the overhead is sometimes non–negligible, since the memory manager cannot take advantage of the underlying structure of the problem, leading to non–optimal layout of data in memory.

As iterative solvers we used GMRES(30) and QMR. The iteration was stopped after the residual norm was less than $\sqrt{\text{eps}}$ times the initial residual norm, where eps $\approx 2.2204 \cdot 10^{-16}$ denotes the machine precision. For some matrices a smaller tolerance was necessary, since the exact solution $(1, \ldots, 1)^\top$ was not sufficiently well approximated. In this case eps was used. The iteration was stopped after 500 steps. Every iterative solution which broke down or did not converge within this number of steps was noted as a failure. The approximate inverse algorithms were compared with the SPARSKIT algorithms ILUT and ILUTP [24] using the same settings.

We briefly describe the results for several matrices and then give detailed numerical results for several selected examples. We focus on examples where we observed major differences for AINV with and without pivoting.

To give a rough idea on how the method performed on the selected collections, Table 1 summarizes which method successfully solved how many problems with respect to the parameters τ and α. The tests were done on 94 matrices from the Harwell-Boeing collection, 26 matrices from the Davis–collection and 58 matrices from the SPARSKIT–collection.

From Table 1 one gets the impression, that Algorithm 3 (AINVP) behaves slightly better than ILUTP. This might have the following reasons.

1. AINVP uses column **and** row pivoting to ensure that W **and** Z are well–bounded. Pivoting only applied to the columns, for example, would locally only bound one factor. But this is essentially what ILUTP does. For reasons of efficiency pivoting with respect to the rows is not done.

2. Dropping in AINVP seems to be less harmful than in ILUTP. Approximation errors caused by dropping in AINVP behave somehow between linear and quadratic with respect to the values that are dropped when regarding the off–diagonal part of $W^\top AZ$. For ILUTP the analogous effect is rational which means that small perturbation in L, U may cause huge approximation errors in the off–diagonal entries of $L^{-1}AU^{-1}$.

3. AINVP sometimes ends up with more fill–in. In the numerical examples dropping was only performed with respect to a fixed drop tolerance but not with respect to the number of nonzeros. The results show that sometimes AINVP needs significantly more fill–in than ILUTP (e.g. Table 7). The higher amount of fill–in then slows down AINVP. But the fill–in could be reduced using more suitable symbolic factorization techniques as well as a reformulation of the algorithm to exploit more zeros [8].

We now comment briefly on some matrix families from the three collections (Harwell–Boeing, Sparskit, Davis). We use tables that indicate on which matrix family pivoting showed improvements. We use the following symbols to indicate how strongly the treatment of the matrix family changed when pivoting was used.

++	+	∘	−	−−
much improved	improved	no great changes	worse	much worse

In general a + is used whenever the choice of parameters moderately improved either the fill–in or the time for the iterative solution (or both) for several matrices of a matrix family. If any of these two criteria were not improved it had essentially to stay constant. In a similar way − is used. ++ is used if the fill–in or the iterative process were significantly improved for most of the matrices of a matrix family. This includes the case when the iterative process changed from no convergence to convergence within $\min\{n, 500\}$ steps. Again, any criteria had to stay at least constant.

For some matrix families the behavior was not uniform. In these cases we used two symbols. If for example, for some matrices the behavior was better, but for others it was worse, then we used the symbol $+/-$.

Table 2 gives an overview on the matrix families of the Harwell–Boeing collection. As one can see from Table 2 the most significant improvements using pivoting were achieved for the CHEMWEST (chemical engineering), ECONAUS families (economic models) and the *shl*,str** matrices (linear programming) . The opposite behavior was observed when pivoting was applied to the FACSIMILE matrices (chemical kinetics). As representative examples see Table 5 (chemical engineering) and Table 6 (thermal simulation, steam injection).

Next are some comments on matrices from the Davis–Collection. The improvements using pivoting are summarized in Table 3. Pivoting was especially successful for the strongly off–diagonal dominant ZITNEY matrices (chemical process separation). Similarly, pivoting resulted in some improvements on the SHYY (Navier–Stokes equations) and MALLYA matrices (light hydrocarbon recovery) which are also extremely strong off–diagonal dominant. As representative example see Table 7 (chemical process separation). Here only the algorithms with pivoting worked at all.

Finally, we comment on our experiments with sample matrices from the SPARSKIT–Collection. Results are summarized in Table 4. Pivoting improved the solution significantly for the DRIVCAV (CFD, driven cavity problems) and FIDAP (fully–coupled Navier-Stokes eq.) matrices. For the TOKAMAK matrices (nuclear physics, plasmas) the algorithms with pivoting were superior. See Table 8 for example.

In most cases the codes for ILUT/ILUTP are much faster than those for the approximate inverses. In fact the implementation of the approximate inverse algorithm with or without pivoting is much more technical and the codes used for the experiments are research codes which have not been profiled and optimized yet. A much improved implementation is still possible, see, e.g., the numerical results in [5, 8].

After illustrating the benefits of using pivoting in the approximate inverse preconditioner with several examples, we now examine the combination of pivoting with an a priori permutation and scaling suggested in [12, 2]. At first glance the use of pivoting and especially the use of strict pivoting seems to be a complementary approach to gain more stability. However, it is clear that combining two different approaches in an appropriate way can be a good compromise. We illustrate this on some matrices which have been reordered and scaled using the method from [12, 2] together with relaxed pivoting ($\alpha = 0.1$). We compare these results with strict pivoting ($\alpha = 1$) and no a priori permutation and with only a priori permutation but no pivoting. We applied these algorithms to some strongly indefinite problems as well as some ill–conditioned problems. For a summary of the results see Table 9. For some problems from partial differential equations (LNS, NNC, DRIVCAV, FIDAP) the factors of either AINV–version were quite dense. These systems from partial differential equations have dense inverses even if entries of small magnitude are skipped. Therefore plain approximate inverse techniques (without any additional techniques like multigrid) might not be suitable to solve these specific systems. The TOKAMAK matrices (nuclear physics) were again easier to solve without dynamic pivoting, just as was the case when no preprocessing is applied. Conversely several of the FIDAP matrices were only solvable with dynamic pivoting, even if preprocessing was used.
As some examples see Tables 10 (linear programming), 11 (chemical engineering), 12 (chemical process separation), 13 (light hydrocarbon recovery). These tables show that for several problems the a priori permutation in combination with the factored approximate inverse with dynamic pivoting result in significantly better results compared with those cases where only one technique, either pivoting or a priori-permutation, is used.

4 Conclusions

We have presented a version of a factored approximate inverse with enhanced stability properties. The algorithm is obtained by carrying over pivoting strategies from LU–decomposition techniques to approximate inverse, exploiting a strong connection between ILU-type methods and factored

approximate inverse type methods. A test with a fairly large collection of test matrices established clearly the advantages of using pivoting. Pivoting in AINV increases robustness in the harder cases and is unlikely to hamper performance too much in the easier cases. Combining approximate inverse with pivoting, row scaling, and a technique of nonsymmetric permutation developed elsewhere [2, 12], shows excellent improvements in robustness of AINV, and opens the possibility of developing reliable preconditioners for very poorly structured matrices.

Acknowledgment. We wish to thank Michele Benzi for providing us with some of the sample matrices that have been preprocessed using the technique from [2]. This helped us obtain some of the results at the end of Section 3. We are also thankful to the reviewers for their valuable comments.

References

[1] M. Benzi, J. K. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM J. Sci. Comput.*, 22:1318–1332, 2000.

[2] M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Sci. Comput.*, 22:1333–1352, 2000.

[3] M. Benzi and C. D. Meyer. A direct projection method for sparse linear systems. *SIAM J. Sci. Comput.*, 16(5):1159–1176, 1995.

[4] M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17:1135–1149, 1996.

[5] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19(3):968–994, 1998.

[6] M. Bollhöfer and Y. Saad. *ILU*s and factorized approximate inverses are strongly related. Part I: Overview of results. Technical Report umsi–2000-39, Minnesota Supercomputer Institue, University of Minnesota, 2000. Submitted to SIAM Matrix. Anal. App.

[7] R. Bridson and W.-P. Tang. Ordering, anisotropy and factored sparse approximate inverse. *SIAM J. Sci. Comput.*, 21(3):867–882, 1999.

[8] R. Bridson and W.-P. Tang. Refining an approximate inverse. *J. Comput. Appl. Math.*, 123:293–306, 2000.

[9] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse–sparse iterations. *SIAM J. Sci. Comput.*, 19(3):995–1023, 1998.

[10] T. Davis. Sparse matrix collection. NA Digest, 1994.

[11] I. Duff, R. Grimes, and J. Lewis. Sparse matrix test problems. *ACM Trans. Math. Software*, 15:1–14, 1989.

[12] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20:889–901, 1999.

[13] R. Freund, G. Golub, and N. Nachtigal. Iterative solution of linear systems. *Acta Numerica*, pages 1–44, 1992.

[14] J. A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[15] M. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18(3):838–853, 1997.

[16] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.

[17] I. E. Kaporin. New convergence results and preconditioning strategies for the conjugate gradient method. *Numer. Lin. Alg. w. Appl.*, 1(2):179–210, 1994.

[18] S. Kharchenko, L. Kolotilina, A. Nikishin, and A. Yeremin. A reliable AINV–type preconditioning method for constructing sparse approximate inverse preconditioners in factored form. *Numer. Lin. Alg. w. Appl.*, 8(3):165–179, 2001.

[19] Y. Kolotilina and Y. Yeremin. Factorized sparse approximate inverse preconditionings I. Theory. *SIAM J. Matrix Anal. Appl.*, 14:45–58, 1993.

[20] J. Meijerink and H. A. V. der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M–matrix. *Math. Comp.*, 31:148–162, 1977.

[21] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method. *ACM Trans. Math. Software*, 6:206–219, 1980.

[22] Y. Saad. ILUT: a dual treshold incomplete ILU factorization. *Numer. Lin. Alg. w. Appl.*, 1:387–402, 1994.

[23] Y. Saad. SPARSKIT and sparse examples. NA Digest, 1994.

[24] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, 1996.

[25] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

5 Appendix

Table 1: Summary of results - total of 178 matrices. Number of successful computed problems for drop tolerance τ and pivot threshold α.

Preconditioner	Accelerator	Parameters			
		$\tau = 0.1$		$\tau = 0.01$	
		$\alpha = 0.1$	$\alpha = 1.0$	$\alpha = 0.1$	$\alpha = 1.0$
Harwell–Boeing Collection (94 test matrices)					
AINV	GMRES(30)	35		39	
AINV	QMR	38		38	
AINVP	GMRES(30)	57	63	78	86
AINVP	QMR	66	72	85	84
ILUT	GMRES(30)	44		43	
ILUT	QMR	41		44	
ILUTP	GMRES(30)	53	54	69	71
ILUTP	QMR	59	58	74	76
Davis Collection (26 matrices)					
AINV	GMRES(30)	14		14	
AINV	QMR	14		14	
AINVP	GMRES(30)	12	16	17	19
AINVP	QMR	15	17	18	20
ILUT	GMRES(30)	14		14	
ILUT	QMR	14		14	
ILUTP	GMRES(30)	13	15	16	18
ILUTP	QMR	14	15	16	17
SPARSKIT Collection (58 matrices)					
AINV	GMRES(30)	2		6	
AINV	QMR	4		9	
AINVP	GMRES(30)	3	16	14	32
AINVP	QMR	4	27	17	34
ILUT	GMRES(30)	7		18	
ILUT	QMR	9		20	
ILUTP	GMRES(30)	6	9	19	19
ILUTP	QMR	11	12	25	23

Table 2: Harwell–Boeing–Collection. Changes in each matrix family when pivoting is used for drop tolerance τ and pivot threshold α

matrix family	$\tau = 0.1$		$\tau = 0.01$	
	$\alpha = 0.1$	$\alpha = 1.0$	$\alpha = 0.1$	$\alpha = 1.0$
ASTROPH	$\circ$	$\circ$	$\circ$	$\circ$
CHEMIMP	$++$	$++$	$++$	$++$
CHEMWEST	$+$	$++$	$+$	$++$
CIRPHYS	$\circ$	$\circ$	$\circ$	$\circ$
ECONAUS	$++$	$++$	$++$	$++$
FACSIMILE	$-$	$-$	$\circ/-$	$\circ/-$
GEMAT	$\circ$	$\circ$	$\circ$	$+$
GRENOBLE	$\circ$	$+$	$+$	$\circ/+$
LNS	$\circ$	$\circ/+$	$\circ/+$	$\circ/+$
NNCENG	$\circ$	$\circ$	$\circ$	$\circ$
NUCL	$\circ$	$\circ$	$\circ$	$\circ/+$
OILGEN	$\circ$	$\circ/-$	$\circ$	$\circ$
PORES	$+/-$	$\circ/+$	$\circ$	$\circ/+$
PSMIGR	$\circ/++$	$\circ/++$	$\circ/+$	$\circ/+$
SAYLOR	$\circ$	$\circ$	$\circ$	$\circ$
SHERMAN	$\circ/+$	$\circ/++$	$\circ/++$	$\circ/++$
SMTAPE(BP*)	$\circ$	$\circ$	$+$	$+/++$
SMTAPE(SHL*,STR*)	$++$	$++$	$++$	$++$
STEAM	$\circ$	$\circ$	$\circ$	$\circ$
WATT	$\circ$	$\circ$	$\circ$	$\circ$

Table 3: Davis–Collection. Changes in each matrix family when pivoting is used for drop tolerance τ and pivot threshold α

matrix family	$\tau = 0.1$		$\tau = 0.01$	
	$\alpha = 0.1$	$\alpha = 1.0$	$\alpha = 0.1$	$\alpha = 1.0$
HAMM	$\circ$	$\circ$	$\circ$	$\circ$
MALLYA	$\circ$	$\circ$	$\circ$	$\circ/+$
PORTFOLIO	$\circ$	$\circ$	$\circ$	$\circ$
SIMON	$\circ/-$	$\circ$	$\circ/-$	$\circ$
SHYY	$\circ$	$\circ$	$\circ$	$+$
WANG	$\circ/-$	$\circ/-$	$\circ/-$	$\circ/-$
ZITNEY	$\circ/+$	$\circ/++$	$\circ/++$	$++$

Table 4: SPARSKIT–Collection. Changes in each matrix family when pivoting is used for drop tolerance τ and pivot threshold α

matrix family	$\tau = 0.1$		$\tau = 0.01$	
	$\alpha = 0.1$	$\alpha = 1.0$	$\alpha = 0.1$	$\alpha = 1.0$
DRIVCAV	$\circ$	$\circ/+$	$\circ/+$	$\circ/++$
FIDAP	$\circ$	$\circ/+$	$\circ$	$\circ/+$
TOKAMAK	$\circ/-$	$\circ/-$	$-$	$-$

Table 5: Matrix CHEMWEST/WEST2021. AINVP and ILUTP with different drop tolerances τ and pivot thresholds α

Method	pivot thresh. α	drop tol. τ	Decomposition Fill–in/time[sec]	$GMRES(30)$ Steps/time[sec]	QMR Steps/time[sec]
AINVP	0.1	10^{-3}	9.9, $1.0{\cdot}10^{0}$	30, $1.2{\cdot}10^{-1}$	33, $2.2{\cdot}10^{-1}$
	1.0	10^{-1}	1.5, $2.0{\cdot}10^{-1}$	121, $3.0{\cdot}10^{-1}$	81, $4.0{\cdot}10^{-1}$
		10^{-2}	2.8, $3.2{\cdot}10^{-1}$	31, $9.0{\cdot}10^{-2}$	37, $1.9{\cdot}10^{-1}$
ILUTP	0.1	10^{-5}	3.1, $2.0{\cdot}10^{-2}$	14, $1.0{\cdot}10^{-2}$	23, $6.0{\cdot}10^{-2}$
	1.0	10^{-5}	3.0, $2.0{\cdot}10^{-2}$	14, $2.0{\cdot}10^{-2}$	27, $6.0{\cdot}10^{-2}$

Table 6: Matrix SHERMAN/SHERMAN2. AINVP and ILUTP with different drop tolerances τ and pivot thresholds α

Method	pivot thresh. α	drop tol. τ	Decomposition Fill–in/time[sec]	$GMRES(30)$ Steps/time[sec]	QMR Steps/time[sec]
AINV		10^{-4}	3.1, $5.6{\cdot}10^{-1}$	371, $9.6{\cdot}10^{-1}$	104, $4.9{\cdot}10^{-1}$
		10^{-5}	4.7, $9.8{\cdot}10^{-1}$	14, $5.0{\cdot}10^{-2}$	18, $1.2{\cdot}10^{-1}$
AINVP	0.1	10^{-2}	1.0, $4.3{\cdot}10^{-1}$	24, $4.0{\cdot}10^{-2}$	25, $9.0{\cdot}10^{-2}$
	1.0	10^{-1}	0.3, $2.3{\cdot}10^{-1}$	151, $2.1{\cdot}10^{-1}$	100, $2.6{\cdot}10^{-1}$
		10^{-2}	0.6, $3.4{\cdot}10^{-1}$	22, $3.0{\cdot}10^{-2}$	24, $7.0{\cdot}10^{-2}$
ILUT		10^{-5}	1.5, $2.0{\cdot}10^{-2}$	82, $8.0{\cdot}10^{-2}$	111, $2.2{\cdot}10^{-1}$
		10^{-6}	1.9, $3.0{\cdot}10^{-2}$	10, $1.0{\cdot}10^{-2}$	17, $3.0{\cdot}10^{-2}$
ILUTP	0.1	10^{-5}	2.0, $5.0{\cdot}10^{-2}$	89, $1.0{\cdot}10^{-1}$	76, $1.7{\cdot}10^{-1}$
		10^{-6}	2.6, $8.0{\cdot}10^{-2}$	8, $1.0{\cdot}10^{-2}$	9, $3.0{\cdot}10^{-2}$
	1.0	10^{-5}	2.3, $6.0{\cdot}10^{-2}$	30, $4.0{\cdot}10^{-2}$	59, $1.5{\cdot}10^{-1}$

Table 7: Matrix ZITNEY/RDIST1. AINVP and ILUTP with different drop tolerances τ and pivot thresholds α

Method	pivot thresh. α	drop tol. τ	Decomposition Fill–in/time[sec]	$GMRES(30)$ Steps/time[sec]	QMR Steps/time[sec]
AINVP	0.1	10^{-2}	21.8, $9.8{\cdot}10^{1}$	—	248, $2.6{\cdot}10^{1}$
		10^{-3}	34.5, $1.8{\cdot}10^{2}$	11, $8.1{\cdot}10^{-1}$	11, $1.7{\cdot}10^{0}$
	1.0	10^{-2}	7.7, $3.5{\cdot}10^{1}$	50, $1.5{\cdot}10^{0}$	46, $2.6{\cdot}10^{0}$
ILUTP	0.1	10^{-2}	3.2, $5.3{\cdot}10^{-1}$	60, $6.0{\cdot}10^{-1}$	55, $9.4{\cdot}10^{-1}$
	1.0	10^{-2}	2.9, $4.2{\cdot}10^{-1}$	23, $2.2{\cdot}10^{-1}$	30, $5.0{\cdot}10^{-1}$

Table 8: Matrix SPARSKIT/FIDAP31. AINVP and ILUT(P) with different drop tolerances τ and pivot thresholds α

Method	pivot thresh. α	drop tol. τ	Decomposition Fill–in/time[sec]	$GMRES(30)$ Steps/time[sec]	QMR Steps/time[sec]
AINVP	0.1	10^{-2}	9.9, $6.1{\cdot}10^{1}$	—	155, $1.1{\cdot}10^{1}$
		10^{-3}	15.8, $1.0{\cdot}10^{2}$	13, $6.1{\cdot}10^{-1}$	13, $1.2{\cdot}10^{0}$
	1.0	10^{-1}	0.6, $1.8{\cdot}10^{0}$	—	194, $4.6{\cdot}10^{0}$
		10^{-2}	3.8, $1.0{\cdot}10^{1}$	48, $1.1{\cdot}10^{0}$	37, $1.6{\cdot}10^{0}$
ILUT		10^{-2}	1.2, $1.0{\cdot}10^{-1}$	78, $4.8{\cdot}10^{-1}$	72, $8.0{\cdot}10^{-1}$
		10^{-3}	1.5, $1.3{\cdot}10^{-1}$	23, $1.6{\cdot}10^{-1}$	26, $3.1{\cdot}10^{-1}$
ILUTP	0.1	10^{-1}	1.8, $3.4{\cdot}10^{-1}$	—	421, $5.7{\cdot}10^{0}$
		10^{-2}	3.7, $1.2{\cdot}10^{0}$	22, $2.7{\cdot}10^{-1}$	24, $5.5{\cdot}10^{-1}$
	1.0	10^{-2}	5.8, $3.6{\cdot}10^{0}$	27, $4.5{\cdot}10^{-1}$	29, $9.3{\cdot}10^{-1}$

Table 9: AINV(P) — Comparison of Pivoting and Preprocessing.

Number of successful computed problems for drop tolerance τ						
matrix family (# matrices)	$\tau = 10^{-1}$			$\tau = 10^{-2}$		
	only pivoting $\alpha = 1.0$ GMRES / QMR	only preproc. GMRES / QMR	prepr.+ pivoting $\alpha = 0.1$ GMRES / QMR	only pivoting $\alpha = 1.0$ GMRES / QMR	only preproc. GMRES / QMR	prepr.+ pivoting $\alpha = 0.1$ GMRES / QMR
Harwell–Boeing Collection (20 test matrices)						
CHEMW.(11)	10 / 11	9 / 10	11 / 11	11 / 11	11 / 11	11 / 11
LNS(6)	4 / 4	4 / 6	4 / 4	5 / 6	6 / 6	6 / 6
NNC(3)	0 / 1	0 / 0	1 / 2	0 / 1	0 / 0	2 / 3
Davis Collection (11 test matrices)						
MALLYA(6)	0 / 0	1 / 2	1 / 2	0 / 1	2 / 2	2 / 3
ZITNEY(6)	3 / 3	0 / 0	1 / 2	4 / 5	3 / 4	6 / 6
SPARSKIT Collection (64 test matrices)						
DRIVC.(22)	5 / 13	4 / 8	7 / 12	15 / 17	10 / 10	14 / 14
FIDAP(37)	12 / 14	8 / 8	11 / 12	20 / 19	9 / 9	23 / 25
TOKAM.(5)	0 / 1	3 / 4	0 / 1	1 / 2	2 / 5	1 / 3

Table 10: Matrix BP/BP1200. AINV(P) with different versions of pivoting and preprocessing

version of AINV(P)	drop tol. τ	Fill–in/ time[sec]	$GMRES(30)$ steps/time[sec]	QMR steps/time[sec]
only	10^{-2}	8.3, $4.2 \cdot 10^{-1}$	—	56, $1.5 \cdot 10^{-1}$
pivoting ($\alpha = 1.0$)	10^{-3}	13.4, $6.9 \cdot 10^{-1}$	15, $3.0 \cdot 10^{-2}$	15, $5.0 \cdot 10^{-2}$
only	10^{-1}	5.1, $9.0 \cdot 10^{-2}$	—	281, $5.5 \cdot 10^{-1}$
preprocessing	10^{-2}	7.3, $1.1 \cdot 10^{-1}$	20, $3.0 \cdot 10^{-2}$	40, $8.0 \cdot 10^{-2}$
preprocessing	10^{-1}	4.8, $1.4 \cdot 10^{-1}$	49, $5.0 \cdot 10^{-2}$	37, $7.0 \cdot 10^{-2}$
+ pivoting ($\alpha = 0.1$)	10^{-2}	7.5, $1.6 \cdot 10^{-1}$	9, $1.0 \cdot 10^{-2}$	8, $2.0 \cdot 10^{-2}$

Table 11: Matrix WEST/WEST2021. AINV(P) with different versions of pivoting and preprocessing

version of AINV(P)	drop tol. τ	Fill–in/ time[sec]	$GMRES(30)$ steps/time[sec]	QMR steps/time[sec]
only pivoting ($\alpha = 1.0$)	10^{-1}	1.5, $2.0 \cdot 10^{-1}$	121, $3.0 \cdot 10^{-1}$	81, $4.0 \cdot 10^{-1}$
	10^{-2}	2.8, $3.2 \cdot 10^{-1}$	31, $9.0 \cdot 10^{-2}$	37, $1.9 \cdot 10^{-1}$
only preprocessing	10^{-2}	8.1, $1.7 \cdot 10^{-1}$	13, $3.0 \cdot 10^{-2}$	13, $7.0 \cdot 10^{-2}$
preprocessing + pivoting ($\alpha = 0.1$)	10^{-1}	3.5, $1.7 \cdot 10^{-1}$	24, $4.0 \cdot 10^{-2}$	25, $9.0 \cdot 10^{-2}$
	10^{-2}	7.8, $2.5 \cdot 10^{-1}$	9, $2.0 \cdot 10^{-2}$	8, $4.0 \cdot 10^{-2}$

Table 12: Matrix ZITNEY/EXTR1.. AINV(P) with different versions of pivoting and preprocessing

version of AINV(P)	drop tol. τ	Fill–in/ time[sec]	$GMRES(30)$ steps/time[sec]	QMR steps/time[sec]
only pivoting ($\alpha = 1.0$)	10^{-2}	19.3, $4.7 \cdot 10^{0}$	—	159, $2.9 \cdot 10^{0}$
	10^{-3}	31.3, $9.8 \cdot 10^{0}$	26, $3.5 \cdot 10^{-1}$	29, $7.4 \cdot 10^{-1}$
only preprocessing	10^{-2}	19.0, $5.9 \cdot 10^{-1}$	—	183, $3.0 \cdot 10^{0}$
	10^{-3}	24.0, $7.2 \cdot 10^{-1}$	21, $1.9 \cdot 10^{-1}$	29, $4.9 \cdot 10^{-1}$
preprocessing + pivoting ($\alpha = 0.1$)	10^{-1}	11.5, $5.5 \cdot 10^{-1}$	—	255, $3.9 \cdot 10^{0}$
	10^{-2}	16.0, $7.9 \cdot 10^{-1}$	12, $1.2 \cdot 10^{-1}$	12, $2.7 \cdot 10^{-1}$

Table 13: Matrix MALLYA/LHR07C. AINV(P) with different versions of pivoting and preprocessing

version of AINV(P)	drop tol. τ	Fill–in/ time[sec]	$GMRES(30)$ steps/time[sec]	QMR steps/time[sec]
only pivoting ($\alpha = 1.0$)	10^{-3}	9.3, $1.2 \cdot 10^{2}$	—	281, $2.9 \cdot 10^{1}$
	10^{-4}	16.5, $2.9 \cdot 10^{2}$	27, $2.0 \cdot 10^{0}$	29, $4.3 \cdot 10^{0}$
only preprocessing	10^{-4}	14.4, $2.2 \cdot 10^{1}$	—	257, $3.2 \cdot 10^{1}$
	10^{-5}	17.9, $2.9 \cdot 10^{1}$	13, $1.0 \cdot 10^{0}$	13, $2.2 \cdot 10^{0}$
preprocessing + pivoting ($\alpha = 0.1$)	10^{-3}	7.3, $3.3 \cdot 10^{1}$	21, $9.3 \cdot 10^{-1}$	22, $2.0 \cdot 10^{0}$