Technische Universität Berlin

Institut für Mathematik

On the Impact of Inverse-Based dropping on ILUs derived from direct methods

M. Bollhöfer

Technical Report 758-2002

Preprint-Reihe des Instituts für Mathematik
Technische Universität Berlin

Report 758-2002

On the Impact of Inverse-Based dropping on ILUs derived from direct methods

*Matthias Bollhöfer

Abstract

In this paper we present a new incomplete LU decomposition which is based on an existing sparse direct solver. In contrast to many incomplete LU decompositions this ILU incorporates information about the inverse factors L^{-1} and U^{-1} which have direct influence on the dropping strategy. We demonstrate in several large scale examples that this implementation constructs a robust preconditioner.

Keywords: sparse matrices, ILU, sparse direct methods, approximate inverse, condition estimator.

AMS subject classification: 65F05, 65F10, 65F50.

1 Introduction

The solution of large sparse unstructured linear systems in industrial applications remains one of the major challenges in modern numerical analysis. Often sparse direct solvers [11, 13, 17] are used to solve linear systems despite of the (sometimes) enormous amount of memory which might be required by these methods. This limits the size of the application that can be computed in practice or forces the method to use swap space on the hard disk drive which extremely slows down this approach. In recent years, iterative solvers, e.g. Krylov subspace methods [20, 26, 33] often combined with preconditioners like incomplete LU decompositions [33], have become quite popular and successful in many application problems such as those arising from the discretization of elliptic partial differential equations. However, there are still several problems from industrial applications which yield large unstructured matrices for which iterative solvers fail. One often has to tune several parameters (like drop tolerances or levels of fill–in) over a wide range of possible choices to obtain a successful preconditioner. This time–consuming process requires to select the correct values for every specific application and wrong parameters may cause an enormous fill–in or an unacceptable computational time.

*Institut für Mathematik, MA 4–5, Technische Universität Berlin, D–10623 Berlin, Germany. email: bolle@math.tu-berlin.de, URL: http://www.math.tu-berlin.de/∼bolle/.

In an earlier paper [6] we have presented an incomplete LU decomposition, where the norms of the inverse triangular factors L^{-1} and U^{-1} have direct influence on the dropping strategy. In [6] this new kind of ILU was illustrated with several small and moderate size examples that could be handled with MATLAB [35].

In this paper we show how an existing sparse direct solver can be modified to work with this new strategy. We have implemented the new algorithm in FORTRAN. It has three innovative aspects.

1. During the computation of the approximate triangular factors L and U the algorithm uses the row norm of any row of L^{-1} (resp. $U^{-\top}$) to control the process of dropping entries of small size. This strategy is justified, see [7], by relations between incomplete LU decompositions and factored sparse approximate inverses [4].

2. To compute the norms of the inverse triangular factors approximately, we proceed analogously to [6] and modify an algorithm [8, 25] that is originally used for condition estimation. Even more, we will present a new an improved version of it.

3. The (FORTRAN) implementation is based on the direct solver MA50, which is the main LU decomposition part of MA48 [17]. Several questions that arise when supplementing a direct solver with this new dropping strategy will be discussed.

After a brief introduction of incomplete LU decomposition methods and sparse direct LU decomposition techniques we will discuss these three aspects in detail. Finally, several large scale examples from different application areas will demonstrate the effectiveness of this approach.

2 Incomplete and sparse direct LU decompositions

In this section we briefly present the main ingredients for incomplete LU decompositions as well as for sparse direct factorizations. We start with sparse direct techniques. Suppose that we wish to solve the linear system

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

where A is a real nonsingular $n \times n$ matrix and $b, x \in \mathbb{R}^n$ are the right hand side and the solution of the system. The LU decomposition of A can be characterized as follows. Let $A = (a_{ij})$ and suppose that the pivot $\beta = a_{11} \neq 0$. We can then write A as

$$A = \begin{pmatrix} \beta & d \\ c & E \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ L_c & I \end{pmatrix} \begin{pmatrix} \beta & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} 1 & U_d \\ 0 & I \end{pmatrix}, \tag{2}$$

where $L_c = c/\beta$, $U_d = d/\beta$ and $S = E - cd/\beta$ denotes the Schur complement. After the Schur complement S is computed the same procedure is applied to S (instead of A). Finally one obtains $A = LDU$ provided that no zero pivot is encountered. Some sparse

direct solvers are seeking for row/and column permutations in order to reduce the fill–in (e.g. MA48 [17]). These strategies of Markowitz–type [13] require the Schur–complement to be explicitly computed at every step. Other LU decomposition methods compute only one column S at each step and avoid an explicit representation of the whole Schur complement [10, 11, 24, 27]. These strategies may be combined with an a priori column permutation (see e.g. [22]) to improve sparsity. These strategies are supplemented with a [1]stability constraint like

$$|\beta| \geqslant \gamma \, \|c\|_\infty \tag{3}$$

for a moderate constant γ, say $\gamma = 0.1$.

Incomplete LU factorization techniques follow a different strategy. Here the (approximate) Schur complement $\tilde{S}$ is kept sparse by dropping entries. Typically this is done by dropping entries that are less than a prescribed threshold or that are outside a specific pattern (cf. [33]). In the simplest case one uses the pattern of A. In this case this strategy is known as ILU(0) [29]. The second strategy is used in [30, 32, 33] and essentially corresponds to the ILUT algorithm. For an overview of some variants of incomplete LU decomposition see [33]. Incomplete LU decompositions often compute the approximate factorization column by column (or row by row) and avoid the explicit computation of S at any step. This extremely simplifies the data structures, see e.g. [33]. There exist techniques which try to avoid permutations, e.g. reordering and rescaling the initial matrix appropriately before the incomplete LU decomposition is performed (see e.g.[2, 16]). However, these techniques cannot guarantee that (3) is satisfied and maybe interchanges destroy the initial ordering.

Often incomplete LU decompositions are faster to compute than (complete) LU decompositions, if the correct parameters are known. Since these decompositions are even in exact arithmetic only approximate factorizations, one has to compensate this by an additional iterative process for the solution of $Ax = b$. In particular Krylov subspace methods only refer to A implicitly by applying matrix–vector multiplications. Examples are methods like GMRES [34] or QMR [19] or other iterative solvers (see e.g. [20, 26, 33]). Typically Krylov subspace methods converge in a moderate number of steps if A is close to the identity [26]. To accelerate the iterative process, the incomplete LU decomposition is used as preconditioner, i.e., $Ax = b$ is replaced by an equivalent preconditioned system like

$$(L^{-1}AU^{-1})\; y = L^{-1}b, \text{ where } x = U^{-1}y$$

and hopefully $L^{-1}AU^{-1} = D + E \approx D$ is approximately diagonal so that additional diagonal scaling (e.g. by D^{-1} from the left) should be sufficient.

3 Robustness of incomplete LU decompositions

In order to construct an ILU that is more robust than existing approaches [33, 32], we have to take a closer look on how preconditioners are used. Suppose we have constructed

[1]In principle a stability constraint like $|\beta| \geqslant \gamma \, \|d\|_\infty$ could also be considered. From a practical point of view this additional constraint is too costly for the Markowitz–type algorithms and impossible to satisfy for column–oriented codes

an incomplete LU–decomposition

$$A = \tilde{L}\tilde{D}\tilde{U} - R \approx \tilde{L}\tilde{D}\tilde{U},$$

where $\tilde{L}$ and $\tilde{U}^\top$ are lower triangular with unit diagonal and $\tilde{D}$ is diagonal. When we apply $\tilde{L}\tilde{D}\tilde{U}$ as preconditioner in a Krylov subspace method, then we have to construct $\tilde{L}, \tilde{D}, \tilde{U}$ such that

$$\tilde{L}^{-1}A\tilde{U}^{-1} = \tilde{D} - \tilde{L}^{-1}R\tilde{U}^{-1} \approx \tilde{D}.$$

In other words the inverse matrices $\tilde{L}^{-1}$ and $\tilde{U}^{-1}$ determine the approximation properties of the preconditioner. Instead of simply dropping entries from $\tilde{L}$, $\tilde{U}$ if they are small, i.e. at step k drop $\tilde{l}_{jk}$, $\tilde{u}_{km}$ if they satisfy

$$|\tilde{l}_{jk}| \leqslant \tau, \ |\tilde{u}_{km}| \leqslant \tau$$

for any $j, m > k$, one should take the inverse factors $\tilde{L}^{-1}$, $\tilde{U}^{-1}$ into account. This was the main motivation in [6] to use the dropping rule

$$\left|\tilde{l}_{jk}\right| \cdot \left\|e_k^\top \tilde{L}^{-1}\right\|_\infty \leqslant \tau \tag{4}$$

for any entry $\tilde{l}_{jk}$ of $\tilde{L}$ such that $j > k$. A similar rule is used for $\tilde{U}$.

Remark. Condition (4) is the main condition to construct a link between incomplete LU decomposition methods and factored approximate inverse techniques (see e.g. [1, 3, 4, 28]). For a detailed discussion see [7]. It may explain why on the one side incomplete LU decompositions obtained as a by-product from the AINV process [5] and ILUSTAB [6] (MATLAB experiments) on the other side are that robust.

The major problem to satisfy (4) is to estimate $\|e_k^\top \tilde{L}^{-1}\|$, since the inverse of $\tilde{L}$ is usually not available.

4 Condition estimators

In order to drop entries $\tilde{l}_{jk}$ that satisfy satisfy $|\tilde{l}_{jk}| \cdot \|e_k^\top \tilde{L}^{-1}\| \leqslant \varepsilon$ in step k of the incomplete LU decomposition, we need estimates on the norm of $e_k^\top \tilde{L}^{-1}$. In principle this problem requires estimating the row norms of an inverse triangular factor independent on whether $\tilde{L}$ (or L) arises from (approximate) factorizations. For a lower triangular matrix L there is a simple way to get a lower bound of $\|e_k^\top L^{-1}\|_\infty$. The problem how to find a good test vector b has been addressed in [8] in order to construct an estimate for the norm of L^{-1}. Essentially the authors use the inequality $\left\|e_k^\top L^{-1} b\right\|_\infty \leqslant \|e_k^\top L^{-1}\|_\infty \|b\|_\infty$ and seek for a single test vector b such that $\|b\|_\infty = 1$ and

$$\|e_k^\top L^{-1}\|_\infty \approx \left\|e_k^\top L^{-1} b\right\|_\infty$$

for all steps $k = 1, \ldots, n$. Clearly $e_k^\top L^{-1} b$ requires solving a system $Lx = b$ by forward substitution and to use the k–th component $x_k \approx e_k^\top L^{-1} b$ as estimate. In [8] a right hand side b is chosen with values from ± 1. At step k of the forward elimination process $Lx = b$

the sign of b_k is determined such that the sum of $|x_k|$ and the 1-norm of the last $n-k$ components of

$$v^\top = \left(\begin{array}{ccccccc} b_1 & \cdots & b_k & 0 & \cdots & 0 \end{array} \right) L^{-\top}$$

is maximized. In practice v can be easily computed in each step k by a rank–1 update using column k of L. In the sparse case this update is still cheap if one exploits the nonzero structure of L. Note that we need the norm of any row of L^{-1}. Locally one can choose the larger value of $|x_k^-|, |x_k^+|$ depending on the sign of b_k. Globally the norm of the updated components of v is much more important, since we still need good estimates for the remaining components of x. For this reason we only maximize

$$\| (v_l)_{l=k+1,\ldots,n} \|_1$$

but use the larger value of $|x_k^-|, |x_k^+|$ as estimate for $\|e_k^\top L^{-1} b\|_\infty$. This slightly differs from the original approach [8]. For sparse matrices the cost of this condition estimator is of the same order as the forward substitution process. We will demonstrate this condition estimator in an abstract algorithm that essentially follows the idea of the original method [8].

Algorithm 1 *Given a lower triangular matrix L with unit diagonal we compute estimates $x = (x_1, \ldots, x_n)$ such that $x_k \approx \|e_k^\top L^{-1}\|_\infty$.*

Set $v = (0, l_{21}, \ldots, l_{n1})^\top$, $x_1 = 1$.
for $k = 2, \ldots, n$
  $x_+ = 1 - v_k$, $x_- = -1 - v_k$
  Let I be the set of row indices of the nonzero entries from $l_{k+1,k}, \ldots, l_{n,k}$.
  $\nu_+ = \|(v_i + l_{ik} x_+)_{i \in I}\|_1$, $\nu_- = \|(v_i + l_{ik} x_-)_{i \in I}\|_1$.
  if $\nu_+ > \nu_-$: $x_k = x_+$, *else* $x_k = x_-$
  for all $i \in I$: $v_i = v_i + l_{ik} x_k$
  $x_k = \max\{|x_+|, |x_-|\}$
end

This algorithm may serve as a good heuristic to estimate the norm of L^{-1}. There is one drawback that particularly may show up when being applied to triangular factors that arise from LU decompositions. In the sparse case the update procedure $v_i := v_i + l_{ik} x_k$ (hopefully) only requires updating a few values. If only relaxed column pivoting (3) is used in the LU decomposition and no pivoting at all with respect to U, then it is likely that a few large entries dominate the 1–norm of the updated v just because they have different order of magnitude. I.e. while these components possibly increase by Algorithm 1, several others may decrease. This motivates a second strategy how to determine x_k's sign. Instead of taking a norm one could count how many components increase versus how many components decrease and use this to determine the sign of x_k. The additional overhead is small, since one only has to solve a system with a second right hand side.

Algorithm 2 *Given a lower triangular matrix L with unit diagonal we compute estimates $y = (y_1, \ldots, y_n)$ such that $y_k \approx \|e_k^\top L^{-1}\|_\infty$.*

Set $v = (0, l_{21}, \ldots, l_{n1})^\top$, $y_1 = 1$.
for $k = 2, \ldots, n$
  $y_+ = 1 - v_k$, $y_- = -1 - v_k$, $n_+ = 0$, $n_- = 0$
  Let I *be the set of row indices of the nonzero entries from* $l_{k+1,k}, \ldots, l_{n,k}$.
  for $i \in I$: *if* $|v_i + l_{ik} y_+| > \max(2|v_i|, 1/2)\}$: $n_+ = n_+ + 1$
    if $\max(2|v_i + l_{ik} y_+|, 1/2) < |v_i|$: $n_+ = n_+ - 1$
    if $|v_i + l_{ik} y_-| > \max(2|v_i|, 1/2)\}$: $n_- = n_- + 1$
    if $\max(2|v_i + l_{ik} y_-|, 1/2) < |v_i|$: $n_- = n_- - 1$
  if $n_+ > n_-$: $y_k = y_+$, *else* $y_k = y_-$
  for all $i \in I$: $v_i = v_i + l_{ik} y_k$
  $y_k = \max\{|y_+|, |y_-|\}$
end

Essentially Algorithm 2 counts how many components increase or decrease by a factor at least 2. In any case it is ensured that very small components (less than 1/2) are not taken into account.

The conjunction of Algorithm 1 and 2 has been tested for several matrices when being applied to the triangular factors that arise from an LU decomposition. It has been observed that there were only a very few cases when the combination of both strategies produced row-norms of L^{-1}, $U^{-\top}$ that were several orders of magnitude less than the exact values.

Example 3 To illustrate this effect consider the matrix **lnsp3937** from the Harwell–Boeing collection. An LU decomposition of this matrix has been constructed using MATLAB's function **lu**. For the resulting U factor (diagonally scaled) Algorithm 1, 2 are used and compared with the exact row norms of $U^{-\top}$. For better illustration, the original row norms of $U^{-\top}$ were ordered in increasing order. Note that the estimate must be less than or equal to the exact norms.

Figure 1 shows that the Algorithm 1 alone produces estimates that are sometimes much less than the exact values. This only rarely happens when the combination of Algorithm 1 and 2 are used.

Example 4 Another example is the matrix **west2021** from the Harwell-Boeing Collection. The same algorithms as in the previous example were applied. For the results see Figure 2.

This observation was not only made for these examples but for several others, too.

For this reason the implementation of ILUSTAB uses Algorithm 1 and Algorithm 2.

5 Implementation aspects

We will now describe how the condition estimator from Algorithms 1,2 is implemented based on the direct method MA50, a code from the Harwell–Subroutine–Library. MA50 is

Figure 1: estimated row norms of $U^{-\top}$ using only Algorithm 1 (left) and, using Algorithm 1 and 2 (right)

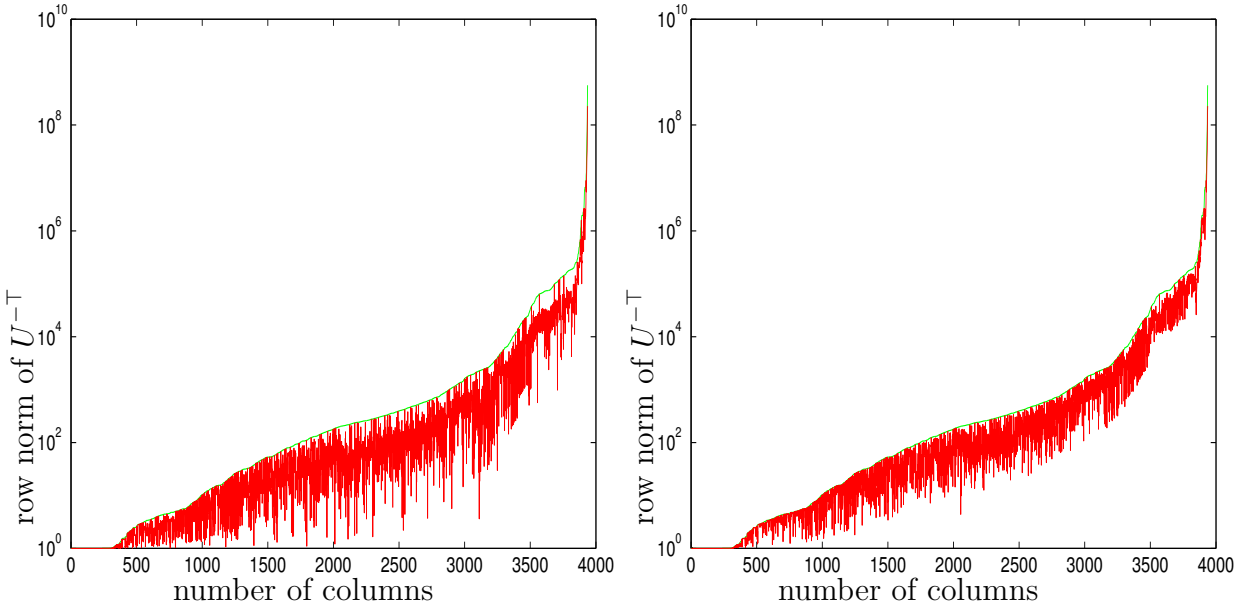
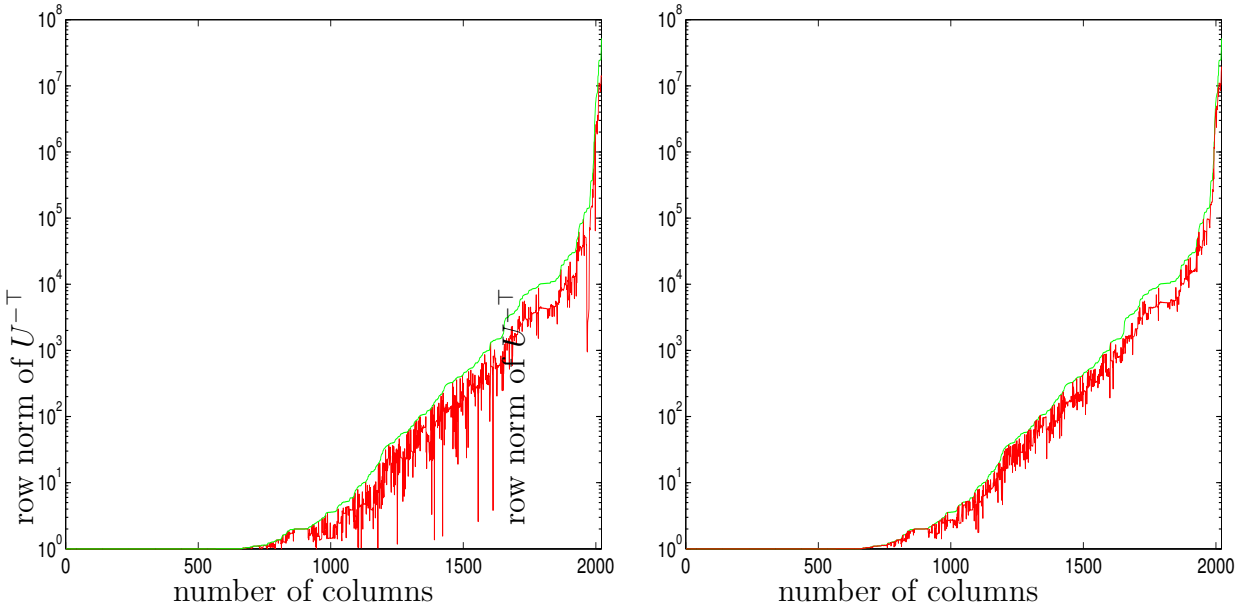


Figure 2: estimated row norms of $U^{-\top}$ using only Algorithm 1 (left) and, using Algorithm 1 and 2 (right)



the core part of MA48 [17]. We will give a sketch on the way the method is implemented and then describe the changes necessary to implement the dropping strategy as well as the updating process for the approximate Schur complement. Analogously to [6] we will refer to this method as ILUSTAB. The algorithm MA50 itself is a two–pass algorithm. So we comment on each pass separately

5.1 First Pass

The whole Schur complement is explicitly computed. Here a pivoting strategy of Markowitz type [13] is used which constructs row and column permutations based on a compromise between sparsity and numerical stability of the diagonal pivots. During this pass the already computed parts of the L and U factor are discarded in order to save memory.

Changes. Algorithms 1,2 are simply interlaced with the first pass of the elimination process. At step k of the incomplete LU decomposition only estimates for $x_{k,L} \approx \|e_k^\top L^{-1}\|$ and $x_{k,U} \approx \|U^{-1}e_k\|$ are required. To do this we simply supplement step k of the incomplete factorization process with step k of Algorithms 1,2 applied to L and with step k of Algorithms 1,2 applied to $U^\top$. This gives the desired estimates $x_{k,L}$ and $x_{k,U}$. Using $x_{k,L}$ and $x_{k,U}$ we get a dropping rule for column k and row k of the Schur complement S. Step k of the first pass roughly looks as follows.

Algorithm 5 (Step k of ILUSTAB, first pass) *Let $S = (s_{ij})_{i,j \geqslant k}$ be the Schur complement on entry to step k. For $k = 1$, S is substituted by the original matrix A.*

Find a row index $i \geqslant k$ and a column index $j \geqslant k$ using MA50's pivoting strategy
Compute $x_{k,L}$ and $x_{k,U}$ from step k of Algorithms 1,2 applied to L and $U^\top$
Interchange row i and k of S and column j and k of S.
for all $l, m > k$
  if $|s_{lk}| > \tau|s_{kk}|/\max\{1, |x_{k,L}|\}$ ***or*** $|s_{km}| > \tau|s_{kk}|/\max\{1, |x_{k,U}|\}$*:*
    $s_{lm} = s_{lm} - s_{lk}s_{km}/s_{kk}$

We will call this version **ILUSTAB–T** since this kind of approximate Schur complement can be seen as the generalization of the Tismenetsky approach [36].
We will also include computations using the more simple and sparser Schur complement which computes updates only if both values $|s_{lk}|$ and $|s_{km}|$ are larger than their prescribed thresholds:

if $|s_{lk}| > \tau|s_{kk}|/\max\{1, |x_{k,L}|\}$ ***and*** $|s_{km}| > \tau|s_{kk}|/\max\{1, |x_{k,U}|\}$*:*
  $s_{lm} = s_{lm} - s_{lk}s_{km}/s_{kk}$

The simplified version will be called **ILUSTAB–S**.

5.2 Second Pass

The permutation from the first pass is recovered. Instead of the whole Schur complement only one column at each step is computed. The amount of memory necessary to hold L and U is known from the first pass and only a small overhead of memory is needed.

One major problem of this approach is the fast computation of column k of the Schur complement in step k of the algorithm. In principle one has to solve a triangular system with that part of L that has already been computed. MA50 uses an elegant depth–first–search strategy [23] to detect only that part of L that is needed. Even more, this strategy is supplemented with a technique called pruning [18, 24].

Changes. The condition estimates for L, U have been saved from the first pass and need not be recomputed. For the sparse triangular solve the Gilbert-Peierls strategy [23] and as an alternative, a binary search tree have been tested. For ILUSTAB–T even a modified version of pruning [18] was possible. For details see Section 6.1.

Remark. In contrast to [6] here pivoting is only performed with respect to the columns (see (3)). Numerical experiments which also ensure row stability have been made. In most of the examples the additional overhead did not pay off. So row stability is not considered here.

We briefly comment on some minor supplements to the algorithm. To update the l–th row s_{lm}, $m = k+1, \ldots, n$ of the Schur complement S we have to form a linear combination of row k and row l of S. In order to skip an update $s_{lm} \to s_{lm} - s_{lk}s_{km}/s_{kk}$ of row l, s_{lk}/s_{kk} should not only satisfy $|s_{lk}| \leqslant \tau |s_{kk}| / \max\{x_{k,L}, 1\}$, but the norm of row k compared with row l should be small, i.e.

$$\frac{|s_{lk}|}{|s_{kk}|} \| \, (s_{k,k+1}, \ldots s_{k,n}) \, \|_1 \leqslant \tau \| \, (s_{l,k+1}, \ldots s_{l,n}) \, \|_1 .$$

Likewise a similar criterion should be satisfied column–wise for column m:

$$\frac{|s_{km}|}{|s_{kk}|} \| \, (s_{k+1,k}, \ldots s_{n,k}) \, \|_1 \leqslant \tau \| \, (s_{k+1,m}, \ldots s_{n,m}) \, \|_1 .$$

In the first pass of the algorithm the norms of each column and row of the Schur complement can be easily computed, since the Schur complement is explicitly available. For the second pass this is only feasible for the columns, since in every step k of the second pass only column k of the Schur complement is available. As a consequence, the criterion for the row is modified to

$$\frac{|s_{lk}|}{|s_{kk}|} \| \, (a_{k1}, \ldots a_{kn}) \, \|_1 \leqslant \tau \| \, (a_{l1}, \ldots a_{ln}) \, \|_1 .$$

This additional criterion is already used as dropping rule in ILUTP [33] from SPARSKIT.

6 Numerical results

This section will present numerical results for the MA50–based ILUSTAB. The numerical experiments first discuss the different strategies for the second pass. Next ILUSTAB

is applied to a broad class of problems from different application areas, namely circuit simulation, computational fluid dynamics and chemical engineering.

We will compare ILUSTAB with the direct solver MA50 as well as with incomplete *LU* decompositions. As ILU we will use MA50 with the built–in dropping strategy that drops entries whose modulus are less than a given drop tolerance τ. As second incomplete *LU* decomposition ILUTP [33] from SPARSKIT is used. All codes are written in FORTRAN77. The experiments are performed on an IBM RISC 6000 with 4 Power 3–II (375 MHz) processors and 4 GBytes of memory. The matrices were computed using 64–bit address length.

- For ILUTP the matrices are initially reordered using the symmetric minimum degree ordering [21]. Since the matrices tested here are unsymmetric, this reordering does not guarantee that the fill–in will be small.
- An a priori scaling is used such any row of the given matrix has unit 1–norm. Like the symmetric reordering, scaling does not necessarily simplify the problem.
- For the pivoting process $\gamma = 0.1$ is used in (3) for all algorithms.
- For the MA50–based codes the default settings were used, i.e., the codes switch to full matrix processing once the fill–in of the Schur–complement reaches 50% nonzero entries.
- Different values were used for the drop tolerance. We will comment on this in detail.

For the numerical experiments several unsymmetric matrices were chosen from the Harwell–Boeing collection [14, 15, 31], the SPARSKIT collection [31] and Tim Davis's collection [9].

As iterative solvers GMRES(30) [34] is used. The iteration was stopped after the residual norm was less than $\sqrt{\varepsilon}$ times the initial residual norm, where $\varepsilon \approx 2.2204 \cdot 10^{-16}$ denotes the machine precision. The iteration was stopped after 500 steps. Every iterative solution which broke down or did not converge within the number of steps was noted as a failure.

6.1 Fast computation of the approximate Schur complement

For the second pass of ILUSTAB four different implementations were tested.

1. ILUSTAB-T with a binary search tree
2. ILUSTAB-T with the Gilbert–Peierls strategy [23] and modified pruning [18]
3. ILUSTAB-S with a binary search tree
4. ILUSTAB-S with the Gilbert–Peierls strategy

The graph–oriented strategy [23] requires to change the dropping rules. I.e. the 1–norm of any column of the Schur complement has to be replaced by the 1–norm of the original column. In order to have comparable results for these different kind of implementations, we temporarily used the same simplified dropping strategy for the binary search tree. In Subsection 6.2 we turn back to the dropping strategy based on the norm of any column of the Schur complement.

Table 1: Legend of symbols used for all figures

∗ · · · ∗	ILUTP	
× · · · ×	ILUSTAB–T	default, binary search tree
+ · · · +	ILUSTAB–S	default, binary search tree
◇ · · · ◇	ILUSTAB–T	Gilbert-Peierls strategy with modified pruning
⋆ · · · ⋆	ILUSTAB–T	Gilbert-Peierls strategy
○ · · · ○	ILUSTAB–S	Gilbert-Peierls strategy
——	MA50	
□ · · · □	MA50	with simple dropping strategy

Figure 3 shows the amount of computation time for the second pass of ILUSTAB depending on the choice of the drop tolerance τ. The algorithms are applied to the matrix twotone (see Example 6) and to the matrix rma10 from Example 8. See Table 1 for the meaning of the symbols.

In the sequel, for **all** figures the drop tolerances are taken from the left to the right starting with $\tau = 1$ and decreasing. The scale in the x–direction is logarithmic! The dotted lines are only included to give a better impression about the development of the performance when changing the drop tolerance. Only the marks represent numerically computed values. If any method did not converge for coarser drop tolerances, then the leading marks on the left are skipped. On the right end of the x-scale computations were not necessarily performed for all smaller drop tolerances.

From Figure 3 one can see that at least in these examples the binary search tree is faster than the backtracking versions. This holds for ILUSTAB-T as well as for ILUSTAB-S. For rma10 the second pass slows down although the drop tolerance increases but one might expect an acceleration. This is not related with the slight modification of pruning. This can verified from the performance of the algorithm [23] without pruning which also included in Figure 3 for the matrix rma10, it is even slower.

As another example we consider Example 9 where the same effect can be observed (see Figure 4).

While the binary search tree incorporates dropping during the triangular solves, the backtracking strategy does not. This may be an explanation for the dramatic differences. In this sense the two pass approach significantly differs from the ILU approach in [24]. For most of the sample matrices the binary search tree was superior. As a consequence, the backtracking versions are not considered anymore for the remaining numerical examples.

Figure 3: **computation time for pass 2 using different implementations. Matrix twotone (left) and, matrix rma10 (right). Legend see Table 1.**

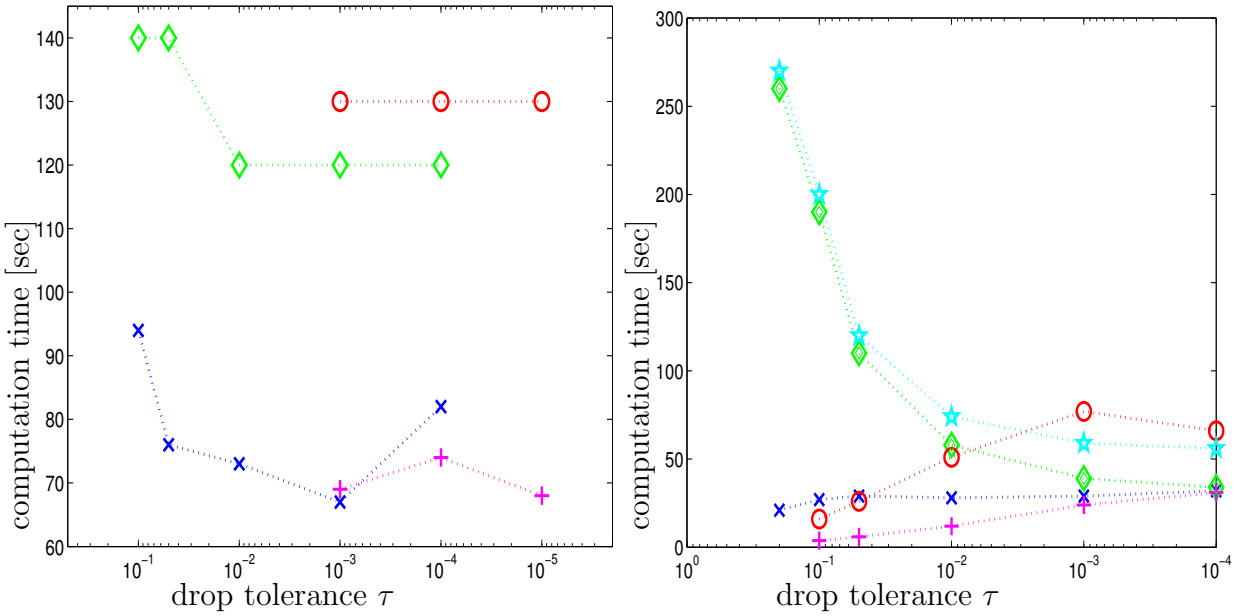
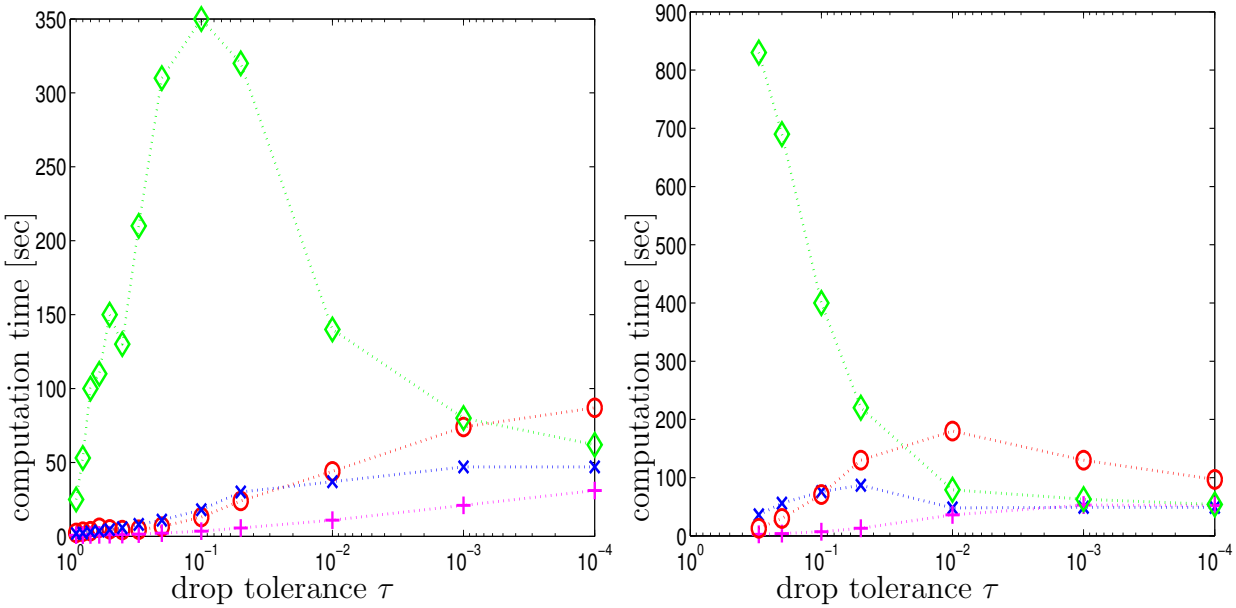


Figure 4: **computation time for pass 2 using different implementations. Matrix venkat01 (left) and, matrix venkat50 (right). Legend see Table 1.**



All examples in Section 6 refer to ILUSTAB-T and ILUSTAB-S using the binary search tree.

6.2 Linear systems arising from different application areas

We now comment on several sample matrices. We will distinguish these matrices by their application area, such that circuit simulation or chemical engineering or CFD (computational fluid dynamics). Several matrices (those from CFD) have almost symmetric patterns. For these kind of matrices one may prefer symmetric permutations. Non–symmetric permutations may sometimes reorder the matrix such that nonzeros occur far away from the diagonal. In this situation the matrix may fill up quickly [12]. Since MA50 also includes this option, we will comment on several examples with symmetric pivoting. Recently a new method has been proposed to reorder matrices and to scale them a–priori [16]. This has a dramatic impact on many iterative solution techniques based on approximate factorization techniques [2, 16]. We will compare the results on the original matrices and on the preprocessed matrices.

In each application area we will first give an overview over the numerical results subject to matrices from specific areas. The matrices examined are collected in the following three classes:

- circuit simulation, Table 2
- Computational fluid dynamics (CFD), Table 3
- chemical engineering, Table 4

In Tables 2, 3 and 4 we illustrate for several drop tolerances τ, how many linear systems can be solved using the associated preconditioner and at most 500 steps of GMRES(30). The comparison is illustrate for the new MA50–based ILU and ILUTP from SPARSKIT. Tables 2, 3, 4 give a first impression on the effectiveness of the new approach. They also confirm earlier experiments [6] performed on smaller matrices in **MATLAB**.

6.3 Matrices from circuit simulation problems

We now give detailed numerical results on matrices which arise from circuit simulation problems. An overview on the performance depending on the drop tolerance τ is shown in Table 2.

From Table 2 one can observe that especially ILUSTAB-T is relatively insensitive with respect to the choice of τ. In fact systems with all sample matrices could be solved with 500 steps of GMRES(30) already using a drop tolerance $\tau = 0.1$.

A large scale example from this application area will be presented to demonstrate the performance of ILUSTAB compared with a sparse direct method MA50, MA50 with drop

Table 2: **Circuit Simulation**

Summary of results — Successful Computation, 11 test matrices

Precnd.	Drop tolerance τ							
	0.5	0.3	0.1	0.01	10^{-3}	10^{-4}	10^{-5}	10^{-6}
ILUSTAB-T	7	9	11	11	11	11	11	11
ILUSTAB-S	6	6	6	7	11	11	11	11
MA50(τ)	0	0	1	2	3	8	9	10
ILUTP	6	6	6	7	8	8	10	10

tolerance τ and ILUTP. It should be pointed out that the computation time of ILUSTAB will usually be closer to that of the direct solver MA50 than to that of ILUTP, since the ILUSTAB has been derived from this direct method and it uses the associated data structures.

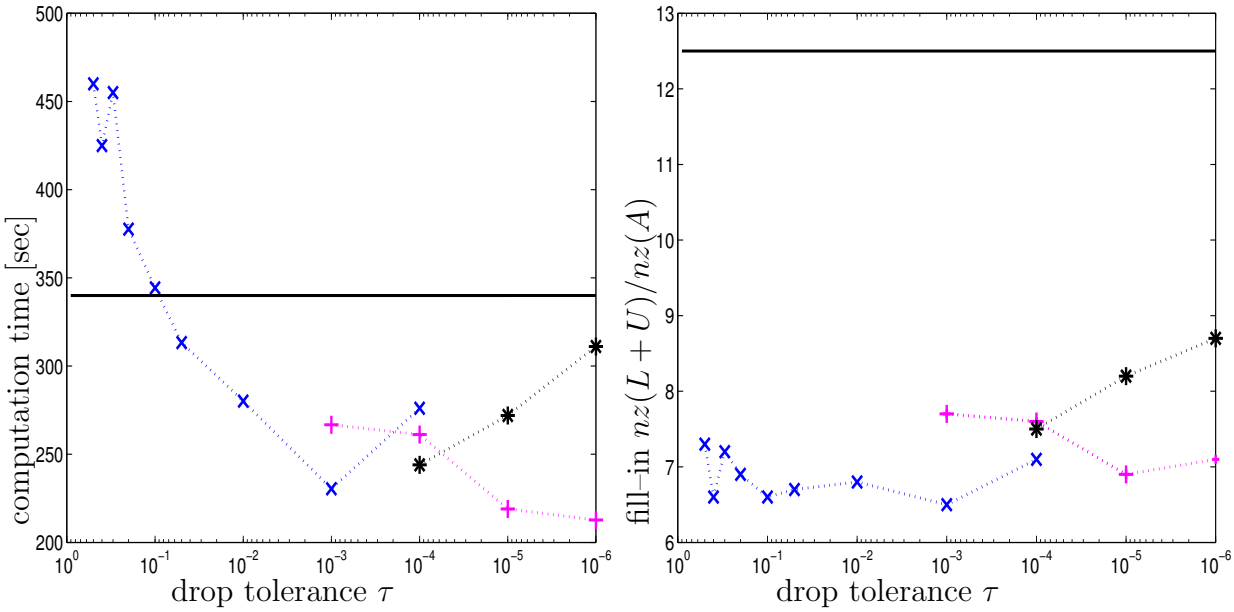
The numerical experiments will also report on the fill–in (number of nonzeros of $L + U$ normalized by the number of nonzeros of A). Note that the MA50–based codes switch to full matrix processing once the density of the Schur complement exceeds 50% of the nonzeros. This aspect has influence on the amount of memory, since row indices need not be stored for the associated submatrix.

Example 6 The matrix **twotone** from the Davis collection arises from the harmonic balance method for efficient frequency domain analysis of large nonlinear circuits, it has $n = 120750$ with 1224224 nonzero entries.

For this matrix ILUTP was not able to solve the problem without using MC64, since either the method did not converge ($\tau \geqslant 10^{-1}$) or the memory requirement was gigantic ($\tau \leqslant 10^{-2}$). Figure 5 refers to ILUTP after MC64 is applied.

No improvement was observed for the MA50–based codes using preprocessing. Therefore the results shown in Figure 5 are based on the original matrix. The simple dropping strategy that is included to MA50 required a drop tolerance of $\tau \leqslant 10^{-5}$ to compute an ILU that ended up in a convergent GMRES iteration. Even in this case the time ($1.8 \cdot 10^3$ [sec] for $\tau = 10^{-5}$) is far beyond the time of the direct solver. These results are not included in Figure 5 since they are far out of range. Here ILUSTAB-T is best with respect to fill–in, ILUSTAB-S is the fastest (but more sensitive with respect to τ). Note that for relatively large drop tolerances the total computation time is typically overlayed by the iterative solution part while the factorization part starts dominating for smaller τ.

Figure 5: **Solvers for matrix twotone, computation time (left) and, memory requirement (right). Legend see Table 1.**



6.4 Linear systems from CFD applications

In this subsection the performance of the numerical methods is examined for many examples from CFD. Problems arising in CFD typically have symmetrically structured nonzero patterns. For this reasons we include results using symmetric pivoting. But it should be noted that even the direct solver was not able to handle all problems using diagonal pivoting (88 of 91 matrices). Table 3 shows again the robustness of ILUSTAB with respect to the drop tolerance τ. For this class of problems symmetric pivoting often has given the best results.

Example 7 The first example in this application area is the matrix **fidapm11** of size $n = 22294$ with 623554 nonzero entries. It is obtained from the SPARSKIT collection. It arises from finite element modeling of a fully-coupled Navier-Stokes equation with temperature equation and chemical convection-diffusion.

The nature of this problem gives an (almost) symmetric pattern and therefore MA50–based codes use symmetric permutations in this case. MA50 gains from its option to compute part of the decomposition in full storage mode. The fill–in factor 94.5 is less serious once one takes into account that a submatrix of size 6260 is factored as dense matrix.

For ILUTP again preprocessing is strongly recommended to end up with fill–in that is still computable.

As Figure 6 shows, even preprocessing does not always cure the problem of memory requirement. While the MA50–based codes did not need preprocessing and even performed better on the original matrix, ILUTP still needs a significant amount of memory, although

Table 3: **Computational fluid dynamics (CFD)**

Summary of results — Successful Computation, 91 test matrices

MA50 (sym. piv)		88							
Precond.		Drop tolerance τ							
		0.5	0.3	0.1	0.01	10^{-3}	10^{-4}	10^{-5}	10^{-6}
sym. piv.	ILUSTAB-T	45	65	75	84	86	88	88	88
	ILUSTAB-S	29	41	64	73	81	84	84	86
	MA50 (τ)	0	0	4	10	28	41	56	62
	ILUSTAB-T	46	66	77	83	83	87	88	89
	ILUSTAB-S	26	43	64	72	77	83	85	88
	MA50 (τ)	0	0	4	11	28	48	64	65
	ILUTP	11	15	27	46	55	57	59	62

its fill–in is still much below that of the direct solver.

Example 8 The matrix **rma10** (Davis collection) arises from finite element modeling in CFD. Its size is $n = 46835$ with 2374001 nonzero entries.

Here again symmetric pivoting is recommended for the MA50–based codes. Pivoting without preserving the symmetric pattern has turned out to be a little bit worse. Again from a practical point of view, ILUTP only worked in combination with MC64. See Figure 7 for a summary of the performance of the algorithms. The gain of using an incomplete ILU decomposition compared with the direct solver reduces to the memory savings. In this example the simple dropping strategy of MA50 needed $\tau = 10^{-7}$ to construct an ILU that converges. Even in this case the time was $1.97 \cdot 10^2$ which more than factor 2 more than the direct method. The fill-in was close to that of the direct method. These results are not included in Figure 7 since they are partially far out of range.

Example 9 The next two examples are the matrix **venkat01** and **venkat50** from the Davis collection. They are obtained from an unstructured solver for the 2D Euler equations at different time steps ($t = 1$ and $t = 50$). Their size is $n = 62424$ with 1717792 nonzero entries.

For this kind of matrix symmetric pivoting has turned out to be more efficient. In Figures 8 and 9 we summarize the methods. Here we note that MC64 did not significantly change neither the computation time nor the memory requirements. For the matrix venkat50, MA50 needed at least $\tau = 10^{-4}$. The computation time was more than 10^3 (out of range).

Figure 6: **Solvers for matrix fidapm11, computation time (left) and, memory requirement (right). Legend see Table 1**

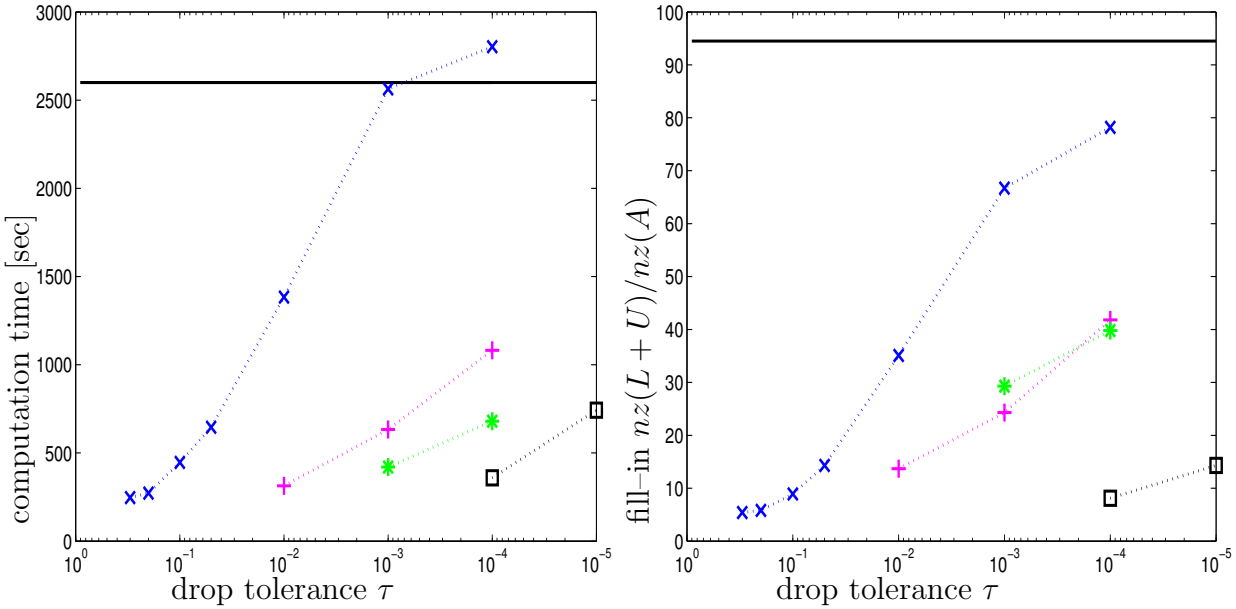


Figure 7: **Solvers for matrix rma10, computation time (left) and, memory requirements (right). Legend see Table 1**

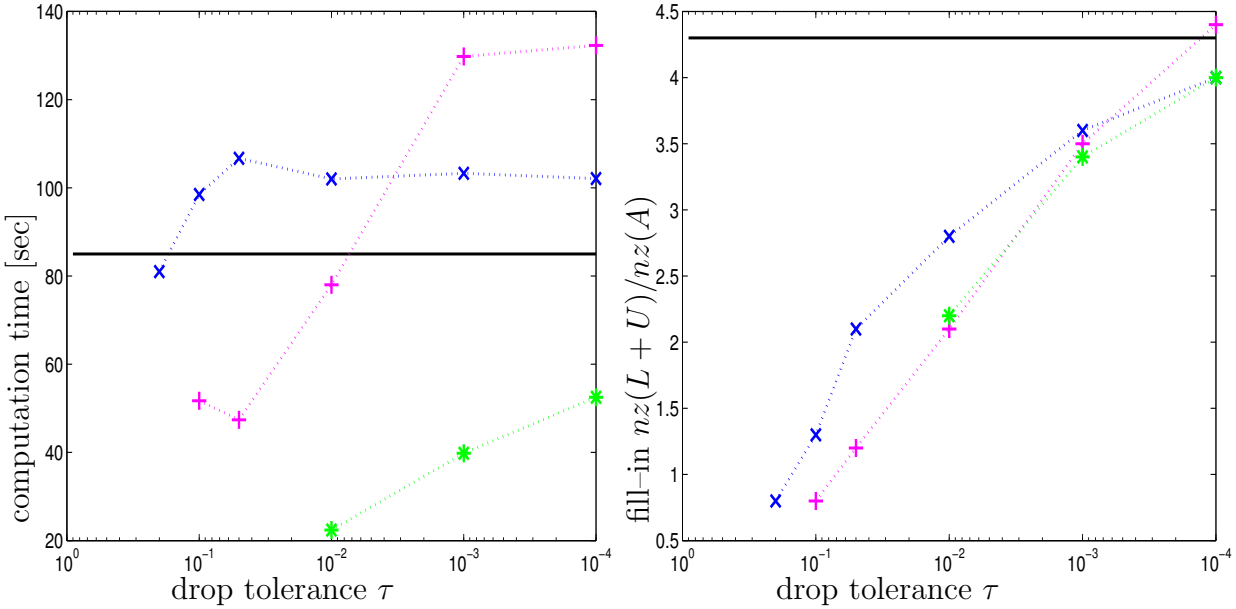


Figure 8: **Solvers for matrix venkat01, computation time (left) and, memory requirements (right). Legend see Table 1.**

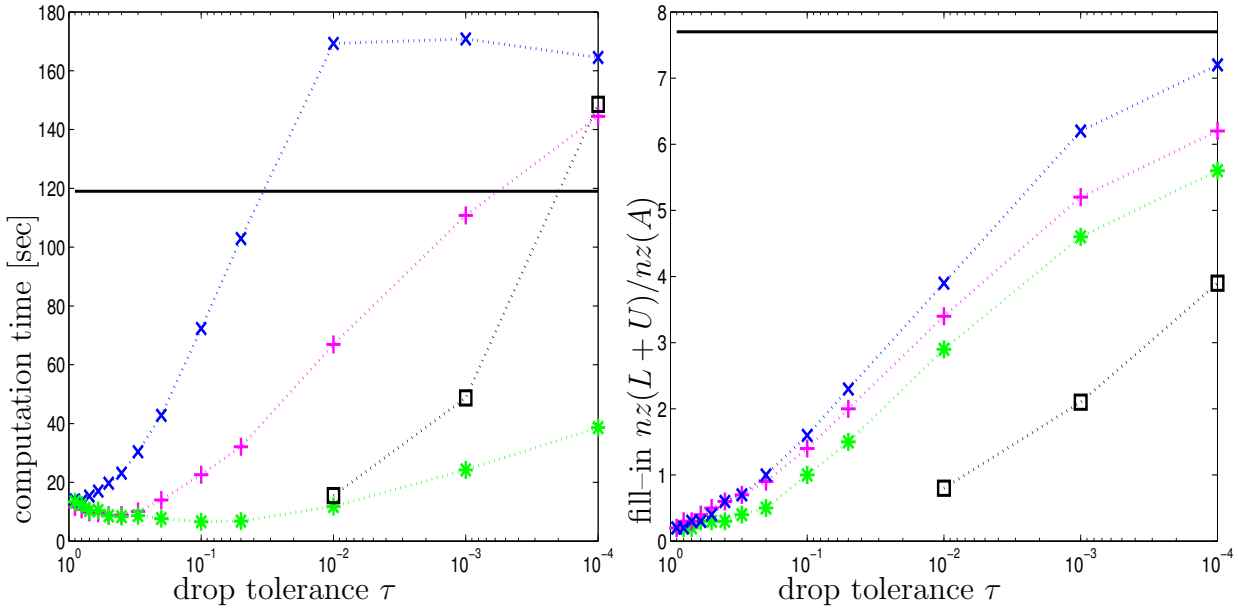
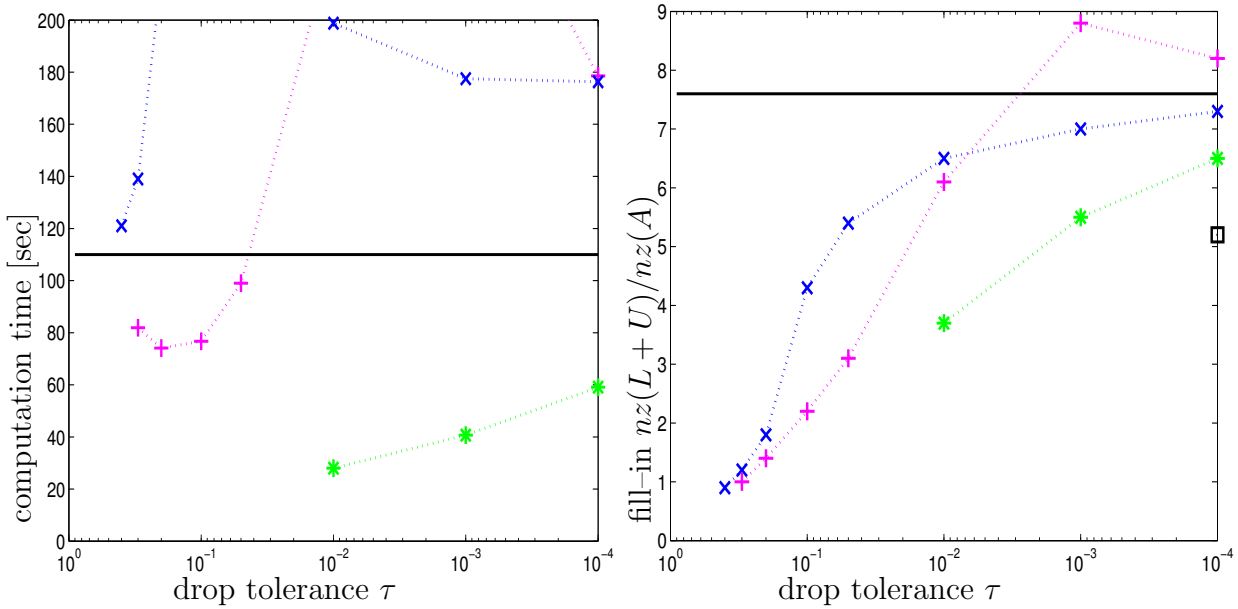


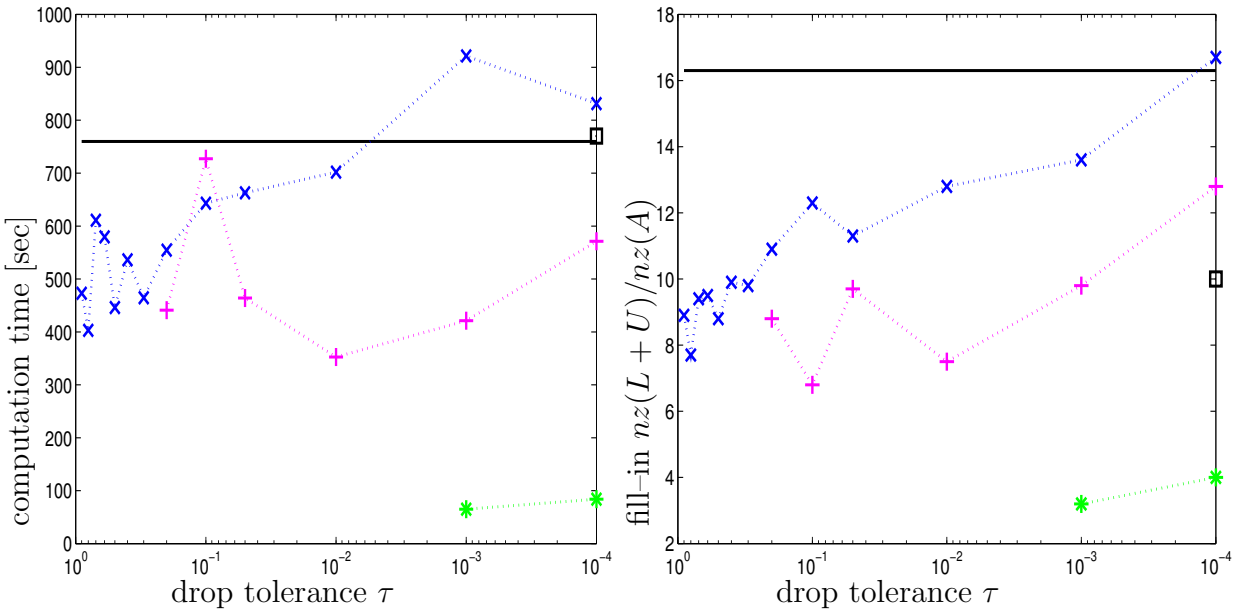
Figure 9: **Solvers for matrix venkat50, computation time (left) and, memory requirements (right). Legend see Table 1.**



Example 10 The matrix **av41092** (Davis collection) arises from a finite element problem. Its size is $n = 41092$ with 1683902 nonzeros.

From the pattern of this matrix one may expect the methods to be significantly improved by MC64. The MA50–based codes are not significantly improved using MC64. In contrast to this, ILUTP fails without MC64 since the memory requirement is too big. See Figure 10 for the numerical results. Here the first pass of both ILUSTAB versions consumes most computation time, especially between .2 and 10^{-2}.

Figure 10: **Solvers for matrix av41092, computation time (left) and, memory requirements (right). Legend see Table 1**



For problems from CFD, ILUSTAB in combination with diagonal pivoting has turned out to be a good compromise between memory requirement and computation time. Usually ILUSTAB-T is more robust but slower than ILUSTAB-S. In the rarely happened that for the relevant drop tolerances the performance of ILUSTAB was significantly behind that of the direct method, but often significant improvements especially with respect to memory were observed. ILUTP is very sensitive with respect to the choice of τ. Using MC64 preprocessing is often useful for ILUTP but it does not always help.

6.5 Problems arising from chemical engineering

As final area of problems several matrices arising in chemical engineering are discussed. These matrices are typically unstructured and symmetric pivoting is not used for these problems. Table 4 shows that ILUSTAB-T performs excellent when being applied to these kind of matrices. Especially the robustness with respect to the choice of the drop tolerance τ is impressive.

Table 4: **Chemical Engineering**

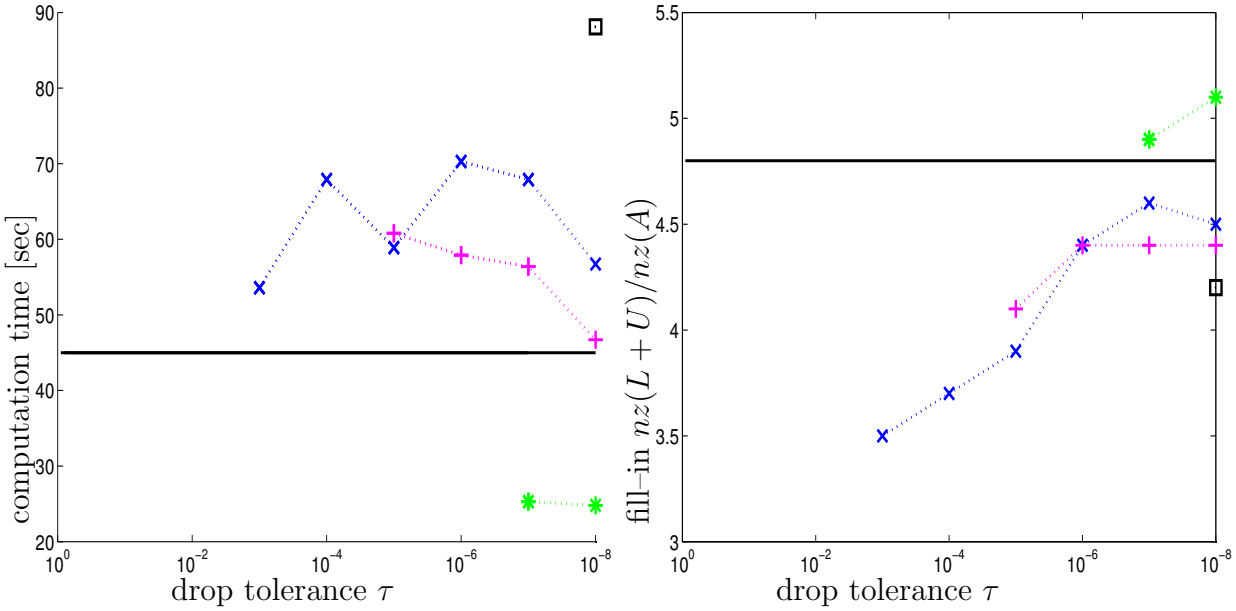
Summary of results — Successful Computation, 62 test matrices

Precond.	Drop tolerance τ							
	0.5	0.3	0.1	0.01	10^{-3}	10^{-4}	10^{-5}	10^{-6}
ILUSTAB-T	50	50	55	60	62	62	62	62
ILUSTAB-S	23	28	33	38	47	57	60	62
MA50(τ)	2	3	9	14	20	29	36	45
ILUTP	17	20	22	32	36	42	45	46

Example 11 The matrix **lhr71c** from the Davis collection arises from light hydrocarbon recovery problems in chemical engineering. Its size is $n = 70304$ with 1528092 nonzero entries.

In Figure 11 we show the numerical results. Surprisingly in this case, the MA50–based codes are not very much affected by MC64, although the original system is strongly unsymmetric already with respect to the pattern. But without MC64, ILUTP performs worse. With respect to memory savings ILUSTAB-T is a good compromise between the pure ILU (without pivoting with respect to sparsity) and the exact direct solver.

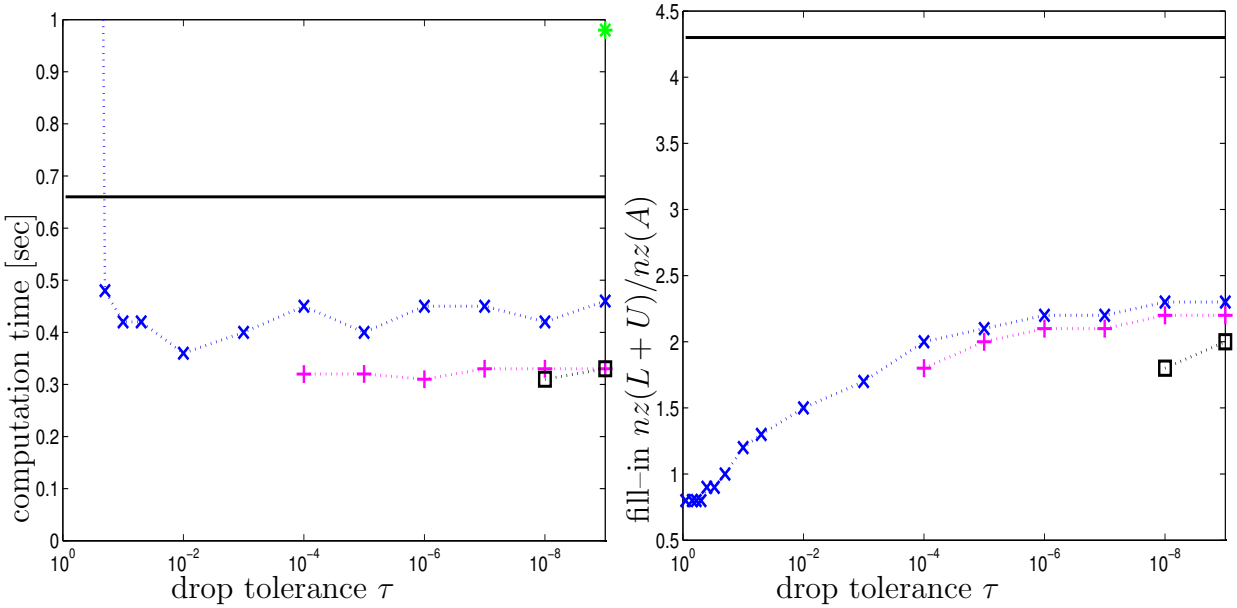
Figure 11: **Solvers for matrix lhr71c, computation time (legend see Table 1)**



Example 12 The final example is the matrix **bayer02** from the Davis collection. It arises from chemical process simulation ($n = 13935$, 63679 nonzeros.

In contrast to the matrices from CFD the matrix **bayer02** is strongly unsymmetrically structured. Here additional preprocessing accelerates all methods, but the MA50–based methods are less sensitive with respect to preprocessing. Preprocessing has a strong influence on ILUTP. This can be seen not only from the different performance with respect to the computation time but also with respect to the drop tolerance which has to be adjusted down to 10^{-9} without preprocessing (cf. Figure 12).

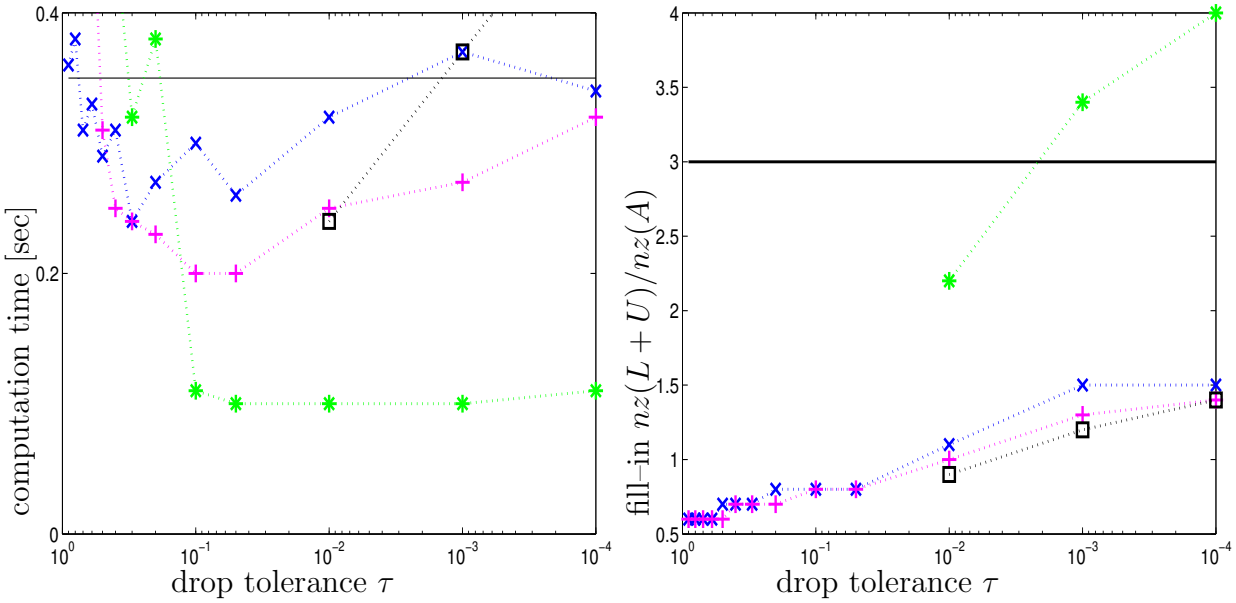
Figure 12: **Solvers for matrix bayer02, computation time (left) and, memory requirements (right). Legend see Table 1.**



6.6 Observations

We have seen from several numerical examples that incomplete LU decompositions that simply drop small entries do not necessarily result in good preconditioners. Other additional techniques are required. For ILUTP this is often achieved using MC64 and confirms previous observations [2, 16]. In this case ILUTP often is the fastest method. But this does not necessarily mean that it is also the most efficient method with respect to memory savings. For MA50 it has turned out that simply dropping small entries is not really helpful which also confirms similar observations on ILUs that are derived from a direct solver [24]. A significant improvement are the dropping strategies of ILUSTAB. ILUSTAB-T usually is less sensitive with respect to the drop tolerance τ, ILUSTAB-S is often faster. The robustness with respect to τ makes it much easier to tune the parameters. Quite often a relatively big τ is sufficient. In addition ILUSTAB inherits the benefits of incomplete LU decompositions to save memory in those cases where direct solvers require a large amount of memory. Its performance in general does not significantly vary when preprocessing like MC64 is added, since it is based on a direct method that performs pivoting with respect to the sparsity pattern. When switching to smaller drop tolerances it turns more and more

Figure 13: **Solvers for matrix bayer02 (after applying MC64), computation time (left) and, memory requirements (right). Legend see Table 1.**



to a direct method. For those cases where the direct solver performs excellent, the gains of ILUSTAB are smaller and may reduce to memory savings. The additional administration together with some changes that were necessary for the incomplete factorization process slow down ILUSTAB slightly but still time and memory requirement are often comparable with the direct solver.

7 Conclusions

We have presented several numerical examples for ILUSTAB. The main technique of this ILU is that the row/column norms of the inverse triangular factors are estimated using a condition estimator. The implementation is based on an existing direct solver (MA50). Numerical experiments have shown the improved robustness of this approach. The method significantly gains with respect to computational time and even more with respect to memory savings. Parameters like the drop tolerance are relatively robust with respect to the variation of the problem. While using a rough drop tolerance one often saves memory and time, already for a moderate drop tolerance ILUSTAB solves most of the sample matrices. Extremely small drop tolerances are rare in practice, although possible. Even in the case of a small drop tolerance the algorithm gains from the advantages of the original direct solver. This makes this approach attractive as alternative to direct solvers. So far there is only little theory based on links to approximate inverse techniques. These are currently under investigation.

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.*, 22:1333–1353, 2000.

[3] 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.

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

[5] M. Benzi and M. Tůma. A robust incomplete factorization preconditioner for positive definite matrices. *Numer. Lin. Alg. w. Appl.*, 2001. To appear.

[6] M. Bollhöfer. A robust ILU with pivoting based on monitoring the growth of the inverse factors. *Linear Algebra Appl.*, 338(1–3):201–218, 2001.

[7] M. Bollhöfer and Y. Saad. On the relations between ILUs and factored approximate inverses. *SIAM J. Matrix Anal. Appl.*, 24(1):219–237, 2002.

[8] A. Cline, C. B. Moler, G. Stewart, and J. Wilkinson. An estimate for the condition number of a matrix. *SIAM J. Numer. Anal.*, 16:368–375, 1979.

[9] T. Davis. Sparse matrix collection. available online at http://www.cise.ufl.edu/∼davis/sparse/.

[10] T. A. Davis. Algorithm 8xx: Umfpack v3.2, an unsymmetric-pattern multifrontal method with a column pre-ordering strategy. Technical Report TR-02-002, Univ. of Florida, 2002.

[11] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999.

[12] I. S. Duff. Private communication. CERFACS, 2001.

[13] I. S. Duff, A. Erisman, and J. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 1986.

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

[15] I. S. Duff, R. G. Grimes, and J. G. Lewis. Users' guide for the Harwell–Boeing sparse matrix collection (release 1). Technical Report RAL–TR–92–086, Rutherford Appleton Laboratory, Oxfordshire, England, 1992.

[16] 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.

[17] I. S. Duff and J. Reid. The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Trans. Math. Software*, 22:187–226, 1996.

[18] S. Eisenstat and J. W. H. Liu. Exploiting structural symmetry in a sparse partial pivoting code. *SIAM J. Sci. Comput.*, 14:253–257, 1993.

[19] R. Freund and N. Nachtigal. QMR: A quasi-minimal residual method for non-hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.

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

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

[22] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13:333–356, 1992.

[23] J. R. Gilbert and T. Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Statist. Comput.*, 8:862–874, 1988.

[24] J. R. Gilbert and S. Toledo. An assessment of incomplete-lu preconditioners for nonsymmetric linear systems. *Informatica*, 24:409–425, 2000.

[25] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, third edition, 1996.

[26] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* Frontiers in Applied Mathematics. SIAM Publications, 1997.

[27] A. Gupta. Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices. *SIAM J. Matrix Anal. Appl.*, 24:529 – 552, 2002.

[28] 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.

[29] 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.

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

[31] National Institute of Standards. Matrix market. available online at http://math.nist.gov/MatrixMarket/.

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

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

[34] 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.

[35] The MathWorks Inc. MATLAB version 6.1 release 12, 2001. a software program.

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