IMPROVED BALANCED INCOMPLETE FACTORIZATION [*]

RAFAEL BRU, JOSÉ MARÍN, JOSÉ MAS [†] AND MIROSLAV TŮMA[‡]

Abstract. In this paper we improve the BIF algorithm which computes simultaneously the LU factors (direct factors) of a given matrix, and their inverses (inverse factors). This algorithm was introduced in [R. Bru, J. Marín, J. Mas and M. Tůma, *SIAM J. Sci. Comput.*, 30 (2008), pp. 2302–2318]. The improvements are based on a deeper understanding of the Inverse Sherman-Morrison (ISM) decomposition and they provide a new insight into the BIF decomposition. In particular, it is shown that a slight algorithmic reformulation of the basic algorithm implies that the direct and inverse factors influence numerically each other even without any dropping for incompleteness. Algorithmically, the nonsymmetric version of the improved BIF algorithm is formulated. Numerical experiments show very high robustness of the incomplete implementation of the algorithm used for preconditioning nonsymmetric linear systems.

Key words. Preconditioned iterative methods, sparse matrices, incomplete decompositions, approximate inverses, Sherman-Morrison formula, nonsymmetric matrices.

1. Introduction.

Consider the system of linear algebraic equations written in the form

$$Ax = b, \tag{1.1}$$

where $A \in \mathbb{R}^{n\times n}$ is a large, sparse and regular nonsymmetric matrix. We are interested in a specific factorization strategy introduced in [12], which is called BIF. While its theoretical basis was originally described in [12] for general nonsymmetric case, algorithmically the paper [12] concentrated to the symmetric and positive definite matrix A. Here we study further properties of this factorization in order to understand more deeply the BIF algorithm and its connections to standard direct incomplete decompositions. In addition, we apply the factorization procedure incompletely as a preconditioner of a nonsymmetric Krylov space method and show that this method may be a useful tool for solving systems of linear algebraic equations.

The so-called BIF algorithm combines the simultaneous computation of both the LU factors and their inverses of a given matrix. In other words, the algorithm performs both the *direct* and the *inverse* decomposition. The fact, that in general, both decompositions are closely connected is well-known, and it was clearly demonstrated in some early papers. Hestenes and Stiefel described such connection in the SPD case in Section 12 of their seminal paper [26]. In particular, they demonstrated the connection of the Gaussian elimination and a specific form of construction of the mutually conjugate vectors. The contemporary form of the former process which was described in that paper is the LU decomposition, and the latter process is called biconjugation, cf. [4, 22], see also the nice survey [14]. A convenient equivalent form of biconjugation is the inverse decomposition, in which we get the inverse factors Z and W such that $A = W^{-T}DZ^{-1}$, directly from A. Let us comment on the biconjugation approaches more in detail, since, as we will see, they are very close to the subject of this paper. Biconjugation algorithms can be classified by the following two features: First, by

*This work was supported by Spanish grant MTM 2007-64477, by the project No. IAA100300802 of the Grant Agency of the Academy of Sciences of the Czech Republic and partially also by the international collaboration support M100300902 of AS CR.

†Institut de Matemàtica Multidisciplinar, Universitat Politècnica de València, 46022 València, Spain (rbru@imm.upv.es, jmarinma@imm.upv.es, jmasm@imm.upv.es)

‡ Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic, (tuma@cs.cas.cz).

the organization of the outermost loop of the biconjugation process (*left-looking* or *right-looking*, analogously to the organization of the outer loop of the LU decomposition). Second, by the way of computation of the diagonal entries in the biconjugation (*one-sided* or *stabilized*, see, e.g., [1]). One-sided computation of the diagonal entries means that each such entry is computed as a dot product of two vectors, whereas the stabilized computation means that the full bilinear form with the system matrix is used. Note that the algorithms can still differ by some other features. For example, the factors Z and W can be evaluated either *successively* one after another or their computation can be *interleaved*. In the latter case the interleaved processes to get Z and W may share some intermediate quantities, but the computed factors are still the same in exact arithmetic.

The biconjugation scheme described in the paper by Hestenes and Stiefel is left-looking and stabilized as well as the algorithm given by Fox, Huskey and Wilkinson [22]. The escalator method by Morris [31] is the left-looking biconjugation process with one-sided computation of diagonal entries, Purcell [33] came with the one-sided and right-looking algorithm. A specific feature of many of the early descriptions is that they did not consider the computational aspects which are very important from a contemporary point of view. To make biconjugation algorithms useful for solving large problems, e.g. in the form of preconditioners, the implementations which fully exploit the matrix sparsity are absolutely crucial. Just the recent progress in exploiting sparsity inspired reconsideration of some classical algorithms including biconjugation. Consequently, these new efforts shed a new light on standard direct decompositions.

The fact that biconjugation can be a practical tool to get an incomplete direct factorization was shown in [5] for symmetric and positive definite matrices. Since then, several nonsymmetric generalizations were proposed, but as far as we know, none of them was shown to solve efficiently difficult large problems. Nevertheless, from the theoretical point of view, there is a nice collection of ideas related to generalizations of direct and inverse decompositions in [9]. The authors in [11] introduced the $(s^{-1}I - A^{-1})^{-1}$ biconjugation process which can be also described in a factorized form. To derive the underlying algorithm, the Sherman-Morrison formula [35] is used repeteadly, and the factorized form is obtained using a similar process to that described in [14]. It was shown in [12] that the resulting BIF factorization contains both a direct and inverse factor of the system matrix. A slight generalization of the new biconjugation is given in [37].

One particular purpose of this paper is to show that the BIF factorization can be modified in such a way that the resulting direct and inverse factors influence numerically each other even when the computation is exact. This result is obtained by a careful analysis of the new biconjugation process, and we hope that it leads to a better understanding of the algorithm. The numerical experiments show that the incomplete computation of the factors leads to a useful preconditioner for solving the nonsymmetric systems that will be called here NBIF. Nevertheless, our goal is much broader. We believe that development of new algorithms, like this, will lead to a better understanding of matrix decompositions in general, and not necessarily only for preconditioning purposes.

The paper is organized as follows. In Section 2 we present the main theoretical results which provide a better understanding of the BIF algorithm. The resulting improved nonsymmetric algorithm is described in Section 3. In Section 4 we give some theoretical results related to scaling of the corresponding biconjugation model.

Implementation details and results of numerical experiments are discussed in Section 5. Our experiments suggest that good rates of convergence can be achieved with the NBIF preconditioner, comparable to those insured by the ILU preconditioner with the inverse-based dropping. At the same time, the new preconditioner seems to be significantly more robust with respect to changes of its size (i.e., in the number of its nonzero entries). Conclusions are given in Section 6.

2. Improving the BIF algorithm.

Consider the exact ISM decomposition from [12] for columns z_i and v_i of Z, V, respectively, and for the matrix $D_s = \operatorname{diag}(r_1, \dots, r_n)$ and scalar $s > 0$. It is given by formulas

$$z_k = e_k - \sum_{i=1}^{k-1} \frac{v_i^T e_k}{s r_i} z_i \quad \text{and} \quad v_k = y_k - \sum_{i=1}^{k-1} \frac{y_k^T z_i}{s r_i} v_i, \tag{2.1}$$

for $k = 1, 2, \dots, n$, where $r_k = 1 + y_k^T z_k / s = 1 + v_k^T e_k / s$, and where e_k and y_k are the columns of I and $Y = A^T - sI$, respectively. The matrices Z_s, V_s and D_s represent the inverse Sherman-Morrison (ISM) decomposition

$$s^{-1} I - A^{-1} = s^{-2} Z_s D_s^{-1} V_s^T \tag{2.2}$$

which can be seen, as mentioned above, as a specific biconjugation. It was shown in [11] that Z_s does not depend on the parameter s, so we will denote it by Z. In addition it was proved that $sD_s = tD_t$, for every two different values of the parameter, as well as an explicit relation between the matrices V_s and V_t for different parameters which is not used in this paper. The matrices Z, D_s and V_s are the factors of the ISM decomposition for a given s.

The relation between the LDU factorization of A, $A = LDU$, and the ISM decomposition

$$D = s^{-1} D_s, \quad U = Z^{-1}, \quad V_s = U^T D - sL^{-T}, \tag{2.3}$$

is proved in [12].

The last relation becomes $V_s = L^T D - sL^{-T}$ in the symmetric and positive definite case. The ISM decomposition then uses both the direct and the inverse factors which are equivalent in exact arithmetic to the Choleski factors and their inverses. Moreover, dropping strategies proposed in [7, 8, 9] can be used to compute the incomplete decomposition, see [12], where the BIF preconditioner based on the direct factors is introduced.

Here we give an alternative proof for the last two relations of (2.3) based on the entries of the matrix. This "entrywise" proof reveals both theoretical and practical consequences that will be used later in the paper. We denote the matrix rows by superscripts. For example $a^k, k = 1, \dots, n$, are rows of A. For simplicity, entries of V_s are denoted by v_{ik} without stating their dependency on s.

THEOREM 2.1. *Let $A = LDU$ be the LDU decomposition of A, and let $s^{-1}I - A^{-1} = s^{-2} Z D_s^{-1} V_s^T$ be the ISM decomposition (2.2). Then*

$$U = Z^{-1}, \quad \textit{and} \quad V_s = U^T D - sL^{-T}.$$

Proof. First, from (2.1) and the definition of vectors y_k observe that

$$y_k^T z_i = a^k z_i \quad \text{for} \quad k > i, \tag{2.4}$$

since Z is upper triangular.

Further, it is well-known that the identity matrix can be obtained by multiplying the matrix

$$U = \begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ & 1 & u_{23} & \cdots & u_{2n} \\ & & \ddots & & \vdots \\ & & & \ddots & \vdots \\ & & & & 1 \end{bmatrix}$$

by the matrices $G_2, \ldots, G_n$ successively from the right, where

$$G_2 = \begin{bmatrix} 1 & -u_{12} & \cdots & 0 \\ & 1 & \cdots & 0 \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}, \ \ldots, \ G_n = \begin{bmatrix} 1 & 0 & \cdots & -u_{1n} \\ & 1 & \cdots & -u_{2n} \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}$$

are the Gauss transformations [24] with reversed ordering $n, \ldots, 1$ of their columns and rows. The same sequence of operations, applied to the identity matrix, gives U^{-1}. If b_k denotes, for simplicity, the k-th column of U^{-1}, we have

$$b_k = e_k - u_{1k} b_1 - \cdots - u_{k-1,k} b_{k-1}. \tag{2.5}$$

In the following we are going to prove that the relation

$$v_k = (a^k - se^k)^T - \sum_{i=1}^{k-1} l_{ki} v_i \tag{2.6}$$

is valid for all $k = 1, \ldots, n$, using a straightforward notation for the entries of L. Simultaneously we will show that the vectors z_k can be expressed as b_k in (2.5), and that

$$v_i^T e_k \equiv v_{ki} = u_{ik} d_i \qquad \text{for all } i < k. \tag{2.7}$$

We will proceed by induction on k.
The factorization $A = LDU$ implies $AU^{-1} = LD$, from which we deduce

$$a^k b_i = l_{ki} d_i, \qquad k > i. \tag{2.8}$$

Consider first $k = 1$. Clearly $z_1 = e_1 = b_1$. On the other hand, from the equation (2.1) we have $v_1 = (a^1 - se^1)^T$; observe also that a^1 is the first row of DU, and that $e_1 = (L^{-T})_1$. Also note that the first column of V_s coincides with the first column of $UD - sL^{-T}$.

For $k = 2$, we have $z_2 = e_2 - \dfrac{v_1^T e_2}{d_1} z_1 = e_2 - u_{12} b_1$, since $v_1^T e_2 \equiv v_{21} = a^1 e_2 = a_{12} = l^1 D u_2 = u_{12} d_1$, which is (2.7) for $i = 1$. Further, from (2.1) and (2.4)

$$v_2 = (a^2 - se^2)^T - \frac{a^2 z_1}{d_1} v_1 = (a^2 - se^2)^T - \frac{a^2 b_1}{d_1} v_1 = (a^2 - se^2)^T - l_{21} v_1$$

since $a^2 b_1 = l_{21} d_1$ by (2.8).

Assume now that the induction hypothesis expressed above is valid for all $i < k$. Since in particular, (2.7) is valid for all $i < k$, we obtain

$$z_k = e_k - \sum_{i=1}^{k-1} \frac{v_i^T e_k}{d_i} z_i = e_k - \sum_{i=1}^{k-1} u_{ik} z_i$$

which coincides with the expression (2.5) from the induction hypothesis. Further, from (2.1), (2.4) and (2.8) we have

$$\begin{aligned} v_k &= (a^k - se^k)^T - \sum_{i=1}^{k-1} \frac{y_k^T z_i}{d_i} v_i = (a^k - se^k)^T - \sum_{i=1}^{k-1} \frac{a^k b_i}{d_i} v_i \\ &= (a^k - se^k)^T - \sum_{i=1}^{k-1} l_{ki} v_i. \end{aligned}$$

Then, for $m > k$ we have

$$v_k^T e_m = (a^k - se^k) e_m - \sum_{i=1}^{k-1} l_{ki} v_i^T e_m = a_{km} - \sum_{i=1}^{k-1} l_{ki} d_i u_{im}.$$

Substituting $a_{km} = \sum_{i=1}^{k} l_{ki} d_i u_{im}$ that comes from the LDU factorization of A, we obtain the relation (2.7). Summarizing the obtained results in matrix form, we get

$$V_s^T = L^{-1}(A - sI) = L^{-1}(LDU - sI) = DU - sL^{-1},$$

which, in addition to $U = Z^{-1}$, completes the proof. $\square$

Note that (2.8) is a variant of the relation $L = AZD^{-1}$ which was used, e.g. in [26] to show an interconnection of the conjugate direction methods and Gaussian elimination, and recently exploited, for example in [5], to compute the RIF decomposition. Also recall that Theorem 2.1 implies that the structure of V_s from [12] can be written in terms of the entries of the LDU factorization $A = LDU$ and of its inverse as

$$v_{ij} = \begin{cases} -s\ell_{ji} & i < j \\ d_i - s & i = j \\ u_{ji} d_j & i > j, \end{cases} \tag{2.9}$$

where ℓ_{ij} is the (i, j)-entry of the matrix L^{-1} .

A careful look at the expression (2.6) leads to the simplification in evaluations of some entries v_{pk} of V_s for $p < k$ which is described in Theorem 2.2.

Theorem 2.2. *With the assumptions and notations of Theorem 2.1, one can simplify the computations of the vectors v_k. In particular, for a row index p, $p < k$ we have,*

$$v_{pk} = sl_{kp} - \sum_{i=p+1}^{k-1} l_{ki} v_{pi}. \tag{2.10}$$

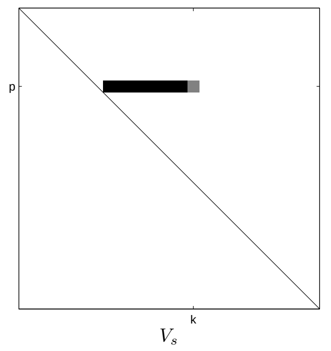

FIG. 2.1. *Dependency of the entry v_{pk} on other entries of the factor V_s after skipping some updates as described in Remark 2.1.*

Proof. Consider v_{pk} for some $p < k$. We have from (2.6) and (2.9)

$$\begin{aligned} v_{pk} &= a_{kp} - l_{k1}v_{p1} - \cdots - l_{k,p-1}v_{p,p-1} - l_{kp}v_{pp} \\ &\quad - l_{k,p+1}v_{p,p+1} - \cdots - l_{k,k-1}v_{p,k-1} \\ &= a_{kp} - l_{k1}(d_1u_{1p}) - \cdots - l_{k,p-1}(d_{p-1}u_{p-1,p}) - l_{kp}(d_p - s) \\ &\quad - l_{k,p+1}v_{p,p+1} - \cdots - l_{k,k-1}v_{p,k-1}. \end{aligned} \tag{2.11}$$

In addition, from definition of the LDU decomposition we have

$$a_{kp} = \sum_{i=1}^{p-1} l_{ki}(d_iu_{ip}) + l_{kp}d_p. \tag{2.12}$$

Substituting the last identity into (2.11) we obtain (2.10) and the proof is complete. □

Theorem 2.2 implies the following remark which emphasizes that computation of some entries of V_s can be simplified. In Section 3 we will observe that their dependence on other entries of the factor can be significantly changed after another computational reformulation.

REMARK 2.1. *In order to get V_s using* (2.1)*, it is possible to skip some evaluations which update v_{pk} for $p < k$, namely those which cancel each other out due to the relation* (2.10)*. In particular, in order to compute v_{pk} we can skip all contributions from columns $1, \ldots, p$. Consequently, the entries of the strict upper triangular part of V_s do not directly depend on entries of the factor U. Note that the entries of the strict upper triangular part of V_s belong to the scaled factor L^{-T}.*

Figure 2.1 depicts the dependencies during the construction of V_s which were mentioned in Remark 2.1. Namely, only the matrix entries in the filled part of the p-th row of V_s are involved in the computation of the entry v_{pk}.

The issue of actual dependency of the factor entries is not only of theoretical interest. For example, it may be important if an incomplete decomposition is computed. For simplicity, let us recall the BIF algorithm in [12] for a symmetric and positive definite matrix A. There, the factors L and L^{-1} embedded in V_s mutually influence each other during the incomplete computation since their computation is *coupled by dropping rules.* That is, actual dropping to compute the incomplete factor L depend on the entries of L^{-1} and vice versa. If we give a closer look at the mechanism of the updates after skipping the unnecessary operations as explained in Remark 2.1, we can see that the entries of the scaled matrix L^{-1} are evaluated in the symmetric and positive definite case with the explicit use of the *computed* entries of L but *not* vice versa. The latter statement is caused by the fact that the entries of the lower triangular part of V_s which belong to L are not used to update the upper triangular part of V_s since they cancel each other out. Generalization of this discussion to the nonsymmetric case is straightforward.

3. The Nonsymmetric Balanced Incomplete Factorization (NBIF) Algorithm. In this section we introduce the nonsymmetric version of BIF, that we call Nonsymmetric Balanced Incomplete Factorization (NBIF).

Theorem 2.1 (see also Corollary 2.4 in [12]) shows that the factors U and D of the LDU factorization of A as well as the inverse factor L^{-1} can be recovered from the ISM decomposition of A. To obtain the remaining direct factor L it is necessary to compute the ISM decomposition of A^T. This observation is the key to generalize the BIF strategy for nonsymmetric matrices. Let us denote by $\tilde{Z}$, $\tilde{V}_s$ and $\tilde{D}_s = \mathrm{diag}(\tilde{r}_1, \dots, \tilde{r}_n)$ the factors of the ISM decomposition of A^T, that is

$$s^{-1}I - A^{-T} = s^{-2}\tilde{Z}\tilde{D}_s^{-1}\tilde{V}_s^{T}. \tag{3.1}$$

The subsequent lemma gives a basis for a substantial modification of the computation of V_s and $\tilde{V}_s$ that changes the dependence of factor entries which we discussed in the previous section after Remark 2.1.

LEMMA 3.1. *Consider the computation of v_{pk} for some $p < k$. The multiplier l_{ki} used in the equation (2.10) for $p+1 \le i \le k-1$, can be alternatively replaced by $\tilde{v}_{ki}/d_i$. That is*

$$v_{pk} = s\frac{\tilde{v}_{kp}}{d_k} - \sum_{i=p+1}^{k-1} \frac{\tilde{v}_{ki}}{d_i} v_{pi}.$$

Proof. The result follows directly from the relation (2.7) applied to the decomposition (3.1). □

As a consequence of Lemma 3.1 and equation (2.8) the entries v_{pk} with $p < k$ can be computed via

$$v_{pk} = s\frac{a^k z_p}{d_p} - \sum_{j=p+1}^{k-1} \frac{\tilde{v}_{kj}}{d_j} v_{pj}, \quad \text{for} \quad p < k, \tag{3.2}$$

instead of the direct use of (2.1), possibly modified as pointed out by Theorem 2.2.

Note that the modification given in the above lemma significantly changes evaluation of the strict upper triangular part of V_s (and $\tilde{V}_s$). We can see that the computed factors L and L^{-1} then do *influence each other numerically.* That is, entries of L are directly used to compute L^{-1} and vice versa. Similarly, the factors U and U^{-1} influence each other numerically as well in the decomposition (3.1). Let us emphasize that the entries of the factors and their inverses influence each other in the NBIF algorithm *independently* of a chosen dropping strategy. This numerical influence can be seen graphically in Figure 3.1 that depicts the construction of the column v_k. The fully filled part of V_s and $\tilde{V}_s$ are used and not changed in the computation of the first

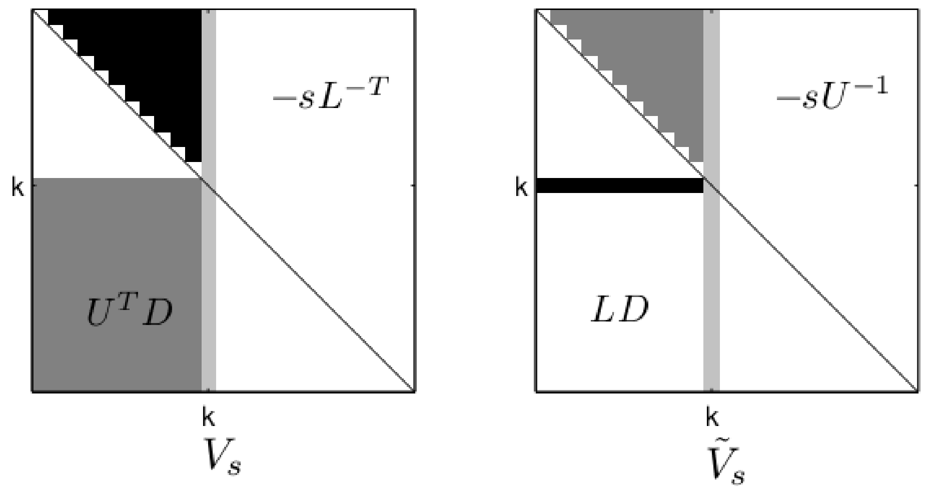

FIG. 3.1. *Dependency of the entries in the column v_k on the other entries of the factors V_s and $\tilde{V}_s$, which were computed so far as described in the text.*

$p-1$ components of v_k and the grey parts in V_s and $\tilde{V}_s$ are used to compute the last $n-p$ entries. Note that here we do not discuss computation of the diagonal entries.

The improvement given in (3.2) is used in Algorithm 3.1, which computes the incomplete LDU factorization of a nonsingular matrix via the ISM decomposition, and gives rise to the Nonsymmetric Balanced Incomplete Factorization (NBIF). The algorithm is written in MATLAB and includes a generalization of the basic algorithm considering a diagonal matrix S instead of a single scalar s as a parameter. This generalization is further discussed in Section 4.

The names of all variables used to compute the ISM factorization of A^T end with a `t` in the algorithm, in particular, matrix $\tilde{V}_s$ is denoted by `Vt` in Algorithm 3.1. To compute the upper triangular part of V_s equation (3.2) is used. The first term of (3.2) is computed in line 40 and the sum term in line 39. The lower triangular part is computed from the second equation of (2.1) (see line 41). Matrix $\tilde{V}_s$ is computed analogously.

ALGORITHM 3.1. The nonsymmetric BIF algorithm (NBIF)

```
1  function [L, D, U, V, Vt] = nbif(A, S, dropv, withZ, dropz)
2  %
3  % Input parameters
4  % A % input matrix, S % matrix of scaling factors
5  % dropv, dropz % drop tolerances for V and Z, respectively
6  % withZ 1 if Z is used, 0 otherwise
7  %
8  % Output parameters
9  % L, D, U % LDU factors of A
10
11           % Initializations
12 n = size(A, 1);
```

```
if withZ
   Z = eye(n);
   Zt = eye(n);
end
V = tril(A') - diag(S); % Matrix V
Vt = tril(A) - diag(S); % Matrix Vt
nrm_L = zeros(n,1); % Norms of rows of L
nrm_invL = zeros(n,1); % Norms of rows of inv(L)
nrm_U = zeros(n,1); % Norms of rows of L of A'
nrm_invU = zeros(n,1); % Norms of rows of inv(L) of A'
D = zeros(n,1); % Matrix D
Dt = zeros(n,1); % Matrix D of A'
          % Main loop
for k=1:n
   for i=1:k-1 % Update the k-th column of V and Vt
      if withZ
         mult_invU=(A(k,:)*Z(:,i))/(D(i)*S(i));
         mult_invL = (A(:,k)'*Zt(:,i))/(Dt(i)*S(i));
      else
         mult_invU = (A(k,:)*[-Vt(1:i-1,i)./S(1:i-1); ...
            1.0; zeros(n-i,1)])/(D(i)*S(i));
         mult_invL=(A(:,k)'*[-V(1:i-1,i)./S(1:i-1);...
            1.0; zeros(n-i,1)])/(Dt(i)*S(i));
      end
      mult_L = Vt(k,i)/(Dt(i)*S(i));
      mult_U = V(k,i)/(D(i)*S(i));
      V(1:i-1,k) = V(1:i-1,k) - mult_L*V(1:i-1,i); % (3.2)
      V(i,k) = V(i,k) + S(i)*mult_invU; % (3.2)
      V(k:n,k)=V(k:n,k) - mult_invU*V(k:n,i); % (2.1)
      Vt(1:i-1,k) = Vt(1:i-1,k) - mult_U*Vt(1:i-1,i);
      Vt(i,k) = Vt(i,k) + S(i)*mult_invL;
      Vt(k:n,k)=Vt(k:n,k) - mult_invL*Vt(k:n,i);
      if withZ
         Z(:,k)=Z(:,k)-(V(k,i)/(D(i)*S(i)))*Z(:,i);
         Zt(:,k)=Zt(:,k)-(Vt(k,i)/(Dt(i)*S(i)))*Zt(:,i);
      end
   end
   D(k) = V(k,k)/S(k) + 1.0;
   Dt(k) = Vt(k,k)/S(k) + 1.0;
   nrm_invL(k) = sqrt(1.0+norm(V(1:k-1,k)+S(k)*S(k)))/S(k);
   nrm_invU(k) = sqrt(1.0+norm(Vt(1:k-1,k)+S(k)*S(k)))/S(k);
   temp = 1.0/(D(k)*D(k)*S(k)*S(k));
   tempt = 1.0/(Dt(k)*Dt(k)*S(k)*S(k));
   nrm_L(k+1:n) = nrm_L(k+1:n)+temp*V(k+1:n,k).*V(k+1:n,k);
   nrm_U(k+1:n) = nrm_U(k+1:n)+tempt*Vt(k+1:n,k).*Vt(k+1:n,k);
   nrm_L(k) = sqrt(nrm_L(k) +1.0);
   nrm_U(k) = sqrt(nrm_U(k) +1.0);
         % Standard dropping in Z and Zt
   if withZ
```

```
62     Z(1:k-1,k) = Z(1:k-1,k).* (abs(Z(1:k-1,k)) > dropz);
63     Zt(1:k-1,k) = Zt(1:k-1,k).* (abs(Zt(1:k-1,k)) > dropz);
64     end
65            % Bollhöfer dropping in V and Vt
66     V(1:k-1,k) = V(1:k-1,k) .*((abs(V(1:k-1,k))...
67        > dropv./nrm_U(1:k-1))); % L --> L^-1
68     V(k+1:n,k) = V(k+1:n,k) .*((abs(V(k+1:n,k))...
69        > dropv*D(k)/nrm_invU(k))); % U^-1 --> U
70     Vt(1:k-1,k) = Vt(1:k-1,k).*((abs(Vt(1:k-1,k))...
71        > dropv./nrm_L(1:k-1))); % U --> U^-1
72     Vt(k+1:n,k) = Vt(k+1:n,k).*((abs(Vt(k+1:n,k))...
73        > dropv*D(k)/nrm_invL(k))); % L^-1 --> L
74 end
75 % Results
76 D = diag(V)+S;
77 Dt = diag(Vt)+S;
78 U = (tril(V)+diag(S))/diag(D);
79 U = U';
80 L = (tril(Vt)+diag(S))/diag(Dt);
```

Observe that Algorithm 3.1 interleaves two processes. One process computes V_s (lines 39–41), and the other computes $\tilde{V}_s$ (lines 42–44). The need to interleave the processes instead of running them successively is implied by adopting the changes introduced in Lemma 3.1 and by using the dropping rules in lines 67–73 (see [7, 8, 9]). Note that this generalization of the BIF algorithm to the nonsymmetric case is similar to generalizations of other biconjugation-based algorithms to get LDU factorization or its inverse. For instance while AINV ([3, 4]) runs the processes to compute its factors successively, straightforward nonsymmetric generalizations of SAINV [1] and RIF [5] and similar algorithms [10], [28] do interleave the computation.

The dropping which we propose is relative to the norms of the inverses of rows of the direct factor L and the norms of the inverses of columns of the direct factor U. This strategy was developed by Bollhöfer and Saad and it is presented in their papers which we quote here. In particular, it was shown both theoretically (by perturbation arguments) and experimentally that the preconditioners based on this dropping are very reliable. A strategy for using these rules in the ISM decomposition was shown in [12].

Since the ISM factor Z is in exact arithmetic equal to the inverse U^{-1} of the LDU factor U, and this factor is embedded in $\tilde{V}_s$ it does not seem to be necessary to compute explicitly the ISM factor Z that can be retrieved from $\tilde{V}_s$ (as is done in lines 32–34 of Algorithm 3.1). The same applies to the factor $\tilde{Z}$. Nevertheless, in some cases, especially in incomplete decompositions, the rowwise information stored in V_s can be quite different of that stored in Z due to *data structures* we use (see [12] and Section 5). Then it is advisable to compute Z and $\tilde{Z}$ as well. Let us recall that rowwise storage of V_s contains only a fixed preset number of largest nonzero entries for each row.

The relation between Z and V_s which was explained above, allows some further changes in the computation of Z, as can be seen in the next lemma.

LEMMA 3.2. *Using the same assumptions and notations of Theorem 2.1, we have*

$$z_k = e_k - \sum_{i=1}^{k-1} \frac{e_k^T v_i}{s r_i} z_i = e_k - \sum_{i=1}^{k-1} \frac{\tilde{y}_k^T \tilde{z}_i}{z_i^T A z_i} z_i, \tag{3.3}$$

Proof. Note first that $\tilde{y}_k^T \tilde{z}_i = a_k^T \tilde{z}_i$ for $k > i$, so for $i \leq k$. Connecting (2.7) and the relation $A^T \tilde{Z} = U^T D$, one has $v_i^T e_k = \tilde{y}_k^T \tilde{z}_i$. If $i = k$, the last relation follows directly from (2.1) if we take into account the fact that V_s and $\tilde{V}_s$ have the same diagonal entries, that is, $v_k^T e_k \equiv \tilde{v}_k^T e_k$.

Further $d_i = e_i^T A z_i \equiv z_i^T A z_i$ follows from the fact that z_k can be expressed using (2.7) as a sum of $e_k^T A z_k$ and a linear combination of terms $z_i^T A z_k$ for $i < k$. Since Z is upper triangular and AU is lower triangular, we are done. Then, from (2.1), relation (3.3) follows. ☐

Formula (3.3) uses entries of Z and $\tilde{Z}$ instead of entries of V_s and $\tilde{V}_s$, and it reminds the AINV decomposition with diagonal entries modified as in SAINV ([1]). Even when we do not use the equivalence from this lemma in our codes, we intended to show that links of NBIF to standard biconjugation techniques are very tight and future computer implementations may exploit this fact.

Last, but not least, simultaneous computation of both direct and inverse factors in the presented algorithm can be used in other techniques of computational linear algebra. Let us mention, for example, its direct use in condition estimators which may combine estimates based on incrementing the direct factor by a row and the inverse factor by a column, see [6], [18]. Choosing the best of the two estimates which can be easily enabled via the BIF or NBIF algorithm is a surprisingly good practical strategy [19].

4. Choice of the scaling factors.

In this section we will discuss the role of the parameter s that can play a scaling role in the biconjugation. The general ISM factorization framework is given in [11] as:

$$A_0^{-1} - A^{-1} = A_0^{-1} Z D^{-1} V^T A_0^{-1}. \tag{4.1}$$

Instead of $A_0 = sI$ as in equation (2.2), we will consider a more general case, $A_0 = S$, where S is a diagonal matrix, i.e.,

$$A_0 = S = \operatorname{diag}(s_1, s_2, \dots, s_n).$$

Note that in this section we do not use the subscript s in the notation of ISM factors.

First we will relate the ISM factorization (4.1) of A with $A_0 = S$ and that of AS^{-1} with $A_0 = I$. As can be seen in equations (4.2) and (4.3) some factors remain unchanged while the factor V is scaled by the matrix S^{-1}. This result can be obtained by formal multiplication of (4.2) by S from the left and embedding the scaling factor just in V. However, having no other result related to the uniqueness of the ISM factorization we give a direct proof.

THEOREM 4.1. *Let*

$$S^{-1} - A^{-1} = S^{-1} Z D^{-1} V^T S^{-1}, \tag{4.2}$$

be the ISM decomposition of A from (4.1) with $A_0 = S$, a nonsingular diagonal matrix, and $y_k = (a^k - s_k e^k)^T$. Then the ISM decomposition of the matrix AS^{-1} with $A_0 = I$, *and* $y_k = (a^k - e^k)^T$ *is*

$$I - (AS^{-1})^{-1} = ZD^{-1}(V^T S^{-1}). \tag{4.3}$$

Proof. Let us write

$$A = S + \sum_{k=1}^{n} e_k y_k^T, \tag{4.4}$$

where $S = \operatorname{diag}(s_1, \dots, s_n)$ and $y_k = (a^k - s_k e^k)^T$, then the ISM factors of A from (4.4) are obtained from the expressions (cf. (2.1))

$$z_k = e_k - \sum_{i=1}^{k-1} \frac{v_i^T S^{-1} e_k}{r_i} z_i, \qquad v_k = y_k - \sum_{i=1}^{k-1} \frac{y_k^T S^{-1} z_i}{r_i} v_i, \qquad r_k = 1 + y_k^T S^{-1} z_k,$$

which are the columns of the matrices Z, V and entries of D, respectively. Now multiplying (4.4) from the right by S^{-1} we obtain

$$\begin{aligned} AS^{-1} &= I + \sum_{k=1}^{n} e_k (y_k^T S^{-1}) \\ &= I + \sum_{k=1}^{n} e_k z_k^T, \end{aligned} \tag{4.5}$$

where $z_k^T = y_k^T S^{-1}$. Note that

$$z_k^T = y_k^T S^{-1} = (a^k - s_k e^k) S^{-1} = a^k S^{-1} - s_k e^k S^{-1} = a^k S^{-1} - e^k,$$

since $e^k S^{-1} = s_k^{-1} e^k$. Then from the equation (4.5) we obtain the ISM factorization of AS^{-1} with $A_0 = I$ using (2.1). Then the columns of the factors and diagonal entries of the last factorization are computed by

$$\check{z}_k = e_k - \sum_{i=1}^{k-1} \frac{\check{v}_i^T e_k}{\check{r}_i} \check{z}_i, \qquad \check{v}_k = z_k - \sum_{i=1}^{k-1} \frac{z_k^T \check{z}_i}{\check{r}_i} \check{v}_i, \qquad \check{r}_k = 1 + z_k^T \check{z}_k.$$

To finish the proof we need to verify that $Z = \check{Z}$, $D = \check{D}$ and $V = S\check{V}$. We proceed by induction.

It is clear for $k = 1$ that $z_1 = \check{z}_1 = e_1$, and hence that $r_1 = \check{r}_1$. On the other hand $v_1 = y_1 = S z_1 = S \check{v}_1$.

Assume now that for $i = 1, 2, \dots, k-1$, the relations $z_i = \check{z}_i$, $r_i = \check{r}_i$, and $v_i = S\check{v}_i$, hold. It is clear that, with these assumptions, the expressions for computing z_k and $\check{z}_k$, and to compute r_k and $\check{r}_k$ are equivalent. Moreover

$$v_k = y_k - \sum_{i=1}^{k-1} \frac{y_k^T S^{-1} z_i}{r_i} v_i = S z_k - \sum_{i=1}^{k-1} \frac{z_k^T S S^{-1} z_i}{r_i} S\check{v}_i = S\left(z_k - \sum_{i=1}^{k-1} \frac{z_k^T \check{z}_i}{\check{r}_i} \check{v}_i\right) = S\check{v}_k,$$

and the proof is complete. ☐

The relation between both ISM factorizations of Theorem 4.1 allows the use of scaling strategies of the matrix A by simple embedding the scaling on the already computed entries of factor V. In addition, the entries of the scaling matrix S can be computed on the fly, according to a particular strategy based on the previous computed vectors z_i and v_i and the scalars r_i. The scaling may be embedded by adding a convenient code to compute `S(k)` immediately after the `i` loop in Algorithm 3.1.

Further, let us study the relation among factors obtained from two different ISM decompositions of the same matrix A with different parameters, that is, $A_0 = I$ and $A_0 = S$. One of the relations we are going to prove generalizes equation (3.8) of [11], (see also the Proposition 3.1 of [12]).

THEOREM 4.2. *Assume that there exists the ISM factorization of A with $A_0 = I$, $x_k = e_k$ and $y_k = (a^k - e^k)^T$. Then there exists the ISM factorization of A with $A_0 = S = \operatorname{diag}(s_1, s_2, \ldots, s_n)$, $x_k = e_k$ and $y_k = (a^k - s_k e^k)^T$.*

Moreover, if Z, V, D and $\check{Z}$, $\check{V}$ and $\check{D}$ are the factors of the former and the latter ISM factorization of A, respectively, then

$$\check{z}_{ij} = z_{ij} s_i / s_j \tag{4.6}$$

$$\check{v}_{ij} = v_{ij} s_i, \qquad i < j \tag{4.7}$$

$$\check{v}_{ij} = v_{ij}, \qquad i > j \tag{4.8}$$

$$\check{v}_{jj} = v_{jj} - s_j + 1 \tag{4.9}$$

$$\check{r}_j = r_j / s_j \tag{4.10}$$

for i, $j = 1, 2, \ldots, n$.

Proof. We proceed by induction on k, that is, on columns of the matrices. For $k = 1$, it is clear that $z_1 = \check{z}_1 = e_1$, then the equality (4.6) holds for $j = 1$ and $i = 1, 2, \ldots, n$. Further, since $v_1 = (a^1 - e^1)^T$, and $\check{v}_1 = (a^1 - s_1 e^1)^T$, then the equalities (4.8) and (4.9) hold for $j = 1$. Observe that (4.7) does not apply. Finally, $\check{r}_1 = 1 + \check{v}_1^T S^{-1} e_1 = 1 + \check{v}_1^T e_1 / s_1 = 1 + \check{v}_{11}/s_1 = 1 + (a_{11} - s_1)/s_1 = a_{11}/s_1$, since $v_{11} = a_{11} - 1$, $\check{r}_1 = (v_{11} + 1)/s_1 = r_1/s_1$, which proves (4.10) for $j = 1$.

Let us assume that the relations (4.6)–(4.10) are valid for $j = 1, 2, \ldots, k-1$, and the corresponding indices i.

Then, applying the inductive assumption to (4.6), we obtain

$$\begin{aligned} \check{z}_k &= e_k - \sum_{j=1}^{k-1} \frac{\check{v}_j^T S^{-1} e_k}{\check{r}_j} \check{z}_j = e_k - \sum_{j=1}^{k-1} \frac{\check{v}_j^T e_k}{s_k \check{r}_j} \check{z}_j \\ &= e_k - \frac{1}{s_k} \sum_{j=1}^{k-1} \frac{\check{v}_{kj}}{\check{r}_j} \check{z}_j = e_k - \frac{1}{s_k} \sum_{j=1}^{k-1} \frac{v_{kj}}{r_j / s_j} \check{z}_j, \end{aligned}$$

where (4.8) was used in the sums since $k > j$. Now, for $l < k$ using the relation (4.6), we have

$$\check{z}_{lk} = -\frac{1}{s_k} \sum_{j=1}^{k-1} \frac{v_{kj} s_j}{r_j} \check{z}_{lj} = -\frac{1}{s_k} \sum_{j=1}^{k-1} \frac{v_{kj} s_j}{r_j} \frac{s_l}{s_j} z_{lj} = -\frac{s_l}{s_k} \sum_{j=1}^{k-1} \frac{v_{kj}}{r_j} z_{lj} = \frac{s_l}{s_k} z_{lk}.$$

Obviously $\check{z}_{kk} = 1 = z_{kk} = s_k z_{kk}/s_k$, and for $l > k$ one has $\check{z}_{lk} = z_{lk} = 0 = s_l z_{lk}/s_k$. Consequently, the relation (4.6) holds for $j = 1, \ldots, k$ and $i = 1, \ldots, n$.

Now from the equation (4.6) we have that the l-th component of $S^{-1}\check{z}_j$ is $\check{z}_{lj}/s_l = z_{lj} s_l/(s_j s_l) = z_{lj}/s_j$, and we have

$$S^{-1} \check{z}_j = s_j^{-1} z_j.$$

Taking into account that $\check{y}_k = a^k - s_k e_k = y_k - (1-s_k)e_k$, the expression for the vectors $\check{v}_k$ becomes

$$\begin{aligned}\check{v}_k &= \check{y}_k - \sum_{j=1}^{k-1} \frac{\check{y}_k^T S^{-1} \check{z}_j}{\check{r}_j} \check{v}_j = y_k + (1-s_k)e_k - \sum_{j=1}^{k-1} \frac{\check{y}_k^T z_j}{s_j \check{r}_j} \check{v}_j \\ &= y_k + (1-s_k)e_k - \sum_{j=1}^{k-1} \frac{y_k^T z_j}{r_j} \check{v}_j - \sum_{j=1}^{k-1} \frac{(1-s_k)e_k^T z_j}{r_j} \check{v}_j \qquad (4.11) \\ &= y_k + (1-s_k)e_k - \sum_{j=1}^{k-1} \frac{y_k^T z_j}{r_j} \check{v}_j,\end{aligned}$$

where the latest equality is implied by the relation $e_k^T z_j = 0$ for $k > j$. Consider now $l < k$, then, by (4.11) and (2.6)

$$\check{v}_{lk} = y_{lk} - \sum_{j=1}^{k-1} \frac{y_k^T z_j}{r_j} \check{v}_{lj} = y_{lk} - \sum_{j=1}^{k-1} l_{kj} \check{v}_{lj} = a_{kl} - \sum_{j=1}^{l-1} l_{kj} \check{v}_{lj} - l_{kl} \check{v}_{ll} - \sum_{j=l+1}^{k-1} l_{kj} \check{v}_{lj}.$$

Now from (4.8), (4.9) and (4.7), we have

$$\check{v}_{lk} = a_{kl} - \sum_{j=1}^{l-1} l_{kj} v_{lj} - l_{kl}(v_{ll} - s_l + 1) - \sum_{j=l+1}^{k-1} l_{kj} v_{lj} s_l,$$

now, applying (2.9) and recalling that $s = 1$, (2.12) and (2.10) we obtain

$$\check{v}_{lk} = a_{kl} - \sum_{j=1}^{l-1} l_{kj} v_{lj} - l_{kl}(d_l - 1 - s_l + 1) - s_l \sum_{j=l+1}^{k-1} l_{kj} v_{lj} = l_{kl} s_l - s_l \sum_{j=l+1}^{k-1} l_{kj} v_{lj} = s_l v_{lk},$$

which proves (4.7) for $j = 1, \ldots, k$. Observe that here we have used in the assumption that (4.7) is clearly true for $k = 2$. For the k-th diagonal entry, it is clear from (4.11) that

$$\check{v}_{kk} = y_{kk} + (1-s_k) - \sum_{i=1}^{k-1} \frac{y_k^T z_i}{r_i} \check{v}_{ki} = (1-s_k) + y_{kk} - \sum_{i=1}^{k-1} \frac{y_k^T z_i}{r_i} v_{ki} = v_{kk} - s_k + 1$$

which proves the relation (4.9) for $j = 1, \ldots, k$. When $l > k$, from (4.11) it is straightforward that

$$\check{v}_{lk} = y_{lk} - \sum_{i=1}^{k-1} \frac{y_k^T z_i}{r_i} \check{v}_{li} = y_{lk} - \sum_{i=1}^{k-1} \frac{y_k^T z_i}{r_i} v_{li} = v_{lk}.$$

which proves the relation (4.8) for $j = 1, \ldots, k$. Finally it is clear that

$$r_k = 1 + v_{kk} = 1 + \check{v}_{kk} + s_k - 1 = \check{v}_{kk} + s_k = s_k \check{r}_k.$$

which proves the relation (4.10) for $j = 1, \ldots, k$, completing the induction.

The latest relation assures the existence of the ISM decomposition of A for $A_0 = S$, and we are done. ◻

COROLLARY 4.3. *Let Z, V and D the ISM factors of A for $A_0 = I$, and let $\check{Z}$, $\check{V}$ and $\check{D}$ the ISM factors of A for $A_0 = S = \mathrm{diag}(s_1, s_2, \ldots, s_n)$, and let $A = LDU$ the LDU factorization of A. Then*

$$\check{Z} = SZS^{-1} = (SUS^{-1})^{-1}, \qquad \check{V} = U^T D - SL^{-T} \qquad (4.12)$$

Let us emphasize that the second part of (4.12) generalizes that of Theorem 2.1. This proves that the LDU factorization of A can be retrieved from the ISM factorization of A with any nonsingular diagonal S.

Figure 4.1 shows the behavior of the matrix OLM100, an ill-conditioned nonsymmetric matrix from the University of Florida Sparse Matrix Collection [15] for different values of the scaling parameter s. It illustrates also that in some cases the behavior of the NBIF algorithm, even without dropping, can be very different when Z is computed (`withZ=1` in Algorithm 3.1, denoted 'with Z' in Figure 4.1), and when Z is not computed (`withZ=0` in Algorithm 3.1, denoted 'without Z'). If differences between the two options are not negligeable then the computation using Z explicitly is more accurate than without it. In addition one can see in the figure that the irregularities caused by changes of s are more significant when Z is not computed. For example, results for $s = 10^3$ and $s = 10^{-3}$ are considerably better than others. Bigger values of s give poor results in both cases. More research is needed to understand the role of the parameter s (or in general of S) in the stability of the process.

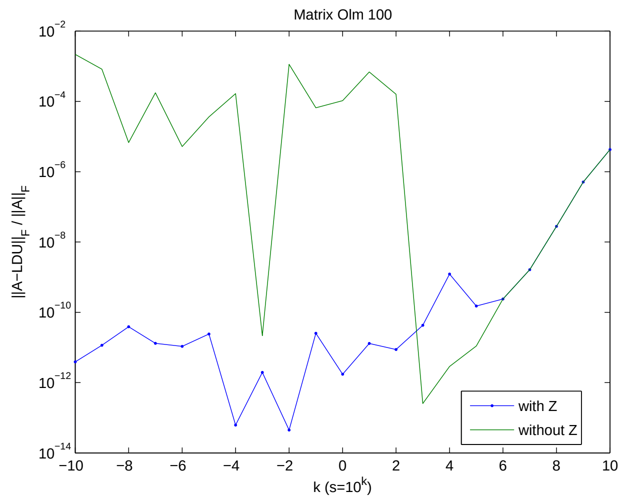

FIG. 4.1. *Relative error, measured in Frobenius norm, of the LDU factorization of the matrix OLM100 in function of s, computed with and without Z.*

We end this section with a comment on the reliability of the (incomplete) ISM decomposition with general scaling matrix S.

THEOREM 4.4. *The (incomplete) ISM process for any diagonal matrix S is breakdown-free for M and H-matrices.*

Proof. The result was stated for the choice $A_0 = sI$ in Theorem 3.3 of [13]. The equation (4.10) shows that it is also true in the more general case $A_0 = S$. $\square$

As a consequence, we have that the breakdown-free condition for ISM decomposition with general scaling matrix S is strong non-singularity of the system matrix. However, note that the factors of the ISM factorization correspond to the shifted matrix $A_0^{-1} - A^{-1}$ (see (4.1)), which may be singular. It happens, for instance if $A_0 = sI$, where s is an eigenvalue of A. Since the matrices Z and D, and A_0 are nonsingular, the ISM factor V must be singular in this case. In fact, its rank is equal to the rank of $A_0^{-1} - A^{-1}$. But, even in this case we can still recover the nonsingular factors of the LDU factorization from V_s because of the specific structure of its diagonal.

5. Numerical experiments. In this section we study the numerical performance of the enhanced BIF algorithm used as the preconditioner of an iterative method. We concentrate on solving large nonsymmetric problems using the NBIF algorithm. Our main goal is to show that the new algorithm is not only of theoretical interest, but it has a potential to be investigated and applied to solve practical problems. In particular, our numerical experiments point out that the new algorithm is very robust.

The incomplete NBIF method from Algorithm 3.1 is compared to the ILU decomposition with inverse-based dropping which we denote here by ILU-ID. In particular, dropping in ILU with an a priori given dropping parameter is based on the norm estimation strategy described in [8]. Our implementation of this strategy was made efficient after a couple of straightforward changes and it uses only two additional vectors for the estimated norms. The factor L is computed by columns and the factor U by rows. The row and column updates are controlled via the mechanism developed in [20, 21], see also [23] and [30]. In this way, both NBIF and ILU algorithms exploit dropping with variable drop tolerances and the comparison then transparently expresses actual power of algorithms combined with an iterative method without any additional feature that could bias the comparison. Later, in order to show very high robustness of the NBIF algorithm also with respect to other ILU techniques, we use in three tested problems also the ILU(τ) preconditioner with an a priori given drop tolerance to drop matrix entries and other intermediate quantities with magnitudes smaller than τ. It is well-known that this algorithm may suffer from strong instabilities and our experiments confirm this. At the same, if ILU(τ) works, it is one of the best algorithms. Note that, in practice, because of its memory demands, it is often replaced by the dual threshold ILU decomposition called ILUT [34] with predictable memory demands and simpler implementation. Both ILU algorithms use memory reallocation inside the codes.

Initially, we intended to use as other possible competitors the nonsymmetric version of the RIF method [5] and the nonsymmetric FSAI method [36]. In contrast to the SPD case, we found both of these preconditioners very unreliable or costly for our test problems, and we do not present their results here. Note that while for the nonsymmetric RIF method we have a fully sparse implementation as explained in [5], our nonsymmetric FSAI implementation uses dense row subproblems as proposed in [29].

The implementation of Algorithm 3.1 shares its basic features with the implementation of BIF described in [12]. Data structures are based on the same principles and they are used to store the incomplete factors Z and V of the ISM factorization of A and the corresponding factors $\tilde{Z}$ and $\tilde{V}$ of A^T, at the same time. From them, the incomplete factors L, D and U used in the preconditioned iterative method are obtained, as can be seen in the last part of the Algorithm 3.1. Note that the nonzero entries in these factors are stored by columns. An additional space of size $10 * n$ is used to store the factors by rows as well, keeping there only their entries with the largest magnitudes. Let us emphasize here that we also store the factors Z and $\tilde{Z}$ explicitly since they seem to provide better information on the rows of V and $\tilde{V}$ used in the dot products. This fact may be caused by the chosen incomplete representation in the data structures, but a future floating-point analysis of the complete algorithm may reveal other important reasons. In practice, overall memory consumption of our implementation of NBIF is approximately two to three-times of that of the ILU-ID

TABLE 1
Test problems

Matrix	n	nz	Application
CHEM_MASTER1	40,401	201,201	chemical engineering 2D problem
EPB3	84,617	463,625	thermal problem
POISSON3DB	85,623	2,374,949	computational fluid dynamics
RAJAT20	86,916	604,299	circuit simulation problem
HCIRCUIT	105,676	513,072	circuit simulation problem
TRANS4	116,835	749,800	circuit simulation
CAGE12	130,228	2,032,536	directed weighted graph
FEM_3D_THERMAL2	147,900	3,489,300	thermal problem
XENON2	157,464	3,866,668	materials problem
CRASHBASIS	160,000	1,750,416	optimization problem
MAJORBASIS	160,000	1,750,416	optimization problem
STOMACH	213,360	3,021,648	3D model of a duodenum
TORSO3	256,156	4,429,042	3D model of human torso
ASIC_320KS	321,671	1,316,085	circuit simulation problem
LANGUAGE	399,130	1,216,334	directed weighted graph
CAGE13	445,315	7,479,343	directed weighted graph
RAJAT30	643,994	6,175,244	circuit simulation problem
ASIC_680K	682,862	2,638,997	circuit simulation problem
CAGE14	1,505,785	27,130,439	directed weighted graph

algorithm for generating similarly-sized incomplete decompositions. In particular, we additionally store the factors Z and $\tilde{Z}$ and use the additional space of restricted size to store V and $\tilde{V}$ by rows, which is partially compensated by additional vectors needed for the inverse-based dropping and for keeping track of the columnwise decomposition of L and rowwise decomposition of U. We are persuaded that this memory consumption is more than compensated for by the results of NBIF presented in this section, noting that rather sparse preconditioners can be reasonably powerful. In general, finding a possible combination of data structures for the four involved factors which would further decrease the memory consumption seems to be an open problem. Note that in [12] we rarely found differences in the complete factorizations of a large set of small problems if the appropriate parts of V and $\tilde{V}$ were used to replace Z and $\tilde{Z}$, but see also Figure 4.1 which may represent an infrequent case.

The test matrices are shown in Table 1. All of them were taken from the University of Florida Sparse Matrix Collection [15]. For each matrix we provide its dimension n, the number of its nonzero entries nz, and its application field as reported in [15].

Each problem was solved by the preconditioned BiCGStab method for a relative decrease 10^{-8} of the system backward error, allowing a maximum of 2000 iterations. This strong criterion was changed to keep some uniformity in the presentation of the experiments in some cases: the allowed relative decrease was set for the matrices STOMACH and TORSO3 to 10^{-3} and to $2 \cdot 10^{-3}$ respectively. Note that a specific feature of comparisons of preconditioned iterative methods for solving large test cases is that each problem may have completely different final attainable accuracy of the method, see, e.g., [25].

For the experiments we used an artificial right-hand side b computed as $b = Ae$, where e is the vector of all ones. The initial approximation to the solution x was the vector of all zeros. All the codes developed for the tests were written in Fortran 90, and have been compiled with Compaq Visual Fortran 6.6c. The computations have

Table 2
A comparison of the NBIF and ILU-ID preconditioners.

Matrix	NBIF				ILU-ID			
	rlsize	*t_p*	*its*	*t_it*	*rlsize*	*t_p*	*its*	*t_it*
CHEM_MASTER1	0.53	0.22	169	0.73	0.70	0.05	168	0.83
EPB3	0.99	0.72	83	1.20	1.06	0.08	120	1.50
POISSON3DB	0.11	0.69	126	3.48	0.09	0.39	199	5.20
RAJAT20	0.17	0.14	8	0.09	0.15	0.13	9	0.09
HCIRCUIT	0.40	0.14	182	2.56	0.21	0.11	203	2.31
TRANS4	0.45	0.23	3	0.06	0.48	1.13	5	0.09
CAGE12	0.31	0.55	5	0.14	0.35	0.38	9	0.22
FEM_3D_THERMAL2	0.06	0.52	20	0.61	0.06	0.43	26	0.80
XENON2	0.05	0.60	539	19.5	0.05	0.44	690	27.1
CRASHBASIS	0.18	0.39	29	0.71	0.20	0.28	14	0.34
MAJORBASIS	0.36	0.73	15	0.42	0.33	0.27	15	0.41
STOMACH	0.07	0.53	20	0.66	0.12	0.42	21	0.67
TORSO3	0.06	0.78	6	0.28	0.07	0.55	6	0.30
ASIC_320KS	0.26	0.47	20	0.94	0.18	0.28	20	0.88
LANGUAGE	0.53	0.70	9	0.55	0.33	0.36	20	1.06
CAGE13	0.06	1.41	6	0.55	0.06	1.05	7	0.61
RAJAT30	0.11	1.56	3	0.34	0.11	1.17	3	0.34
ASIC_680KS	0.36	0.97	5	0.22	0.43	0.59	5	0.55
CAGE14	0.07	12.2	6	2.02	0.07	4.30	8	2.81

been performed using one processor Intel Core2 Q6700 (2.67GHz, 3.24GB RAM).

The problems were initially reordered and scaled by the MC64 code of the HSL library [27]. We used the option which implements the maximum product transversal with row and column scalings. This code represents a sparse implementation of the ideas from [32], see [16], [17] and [2].

Table 2 presents the results obtained with the NBIF and the ILU-ID preconditioners. For each preconditioner we report the ratio between the number of nonzero entries of the incomplete decomposition and the number of nonzero entries in the system matrix. The ratio is denoted by *rlsize*. Further we show the time to construct the preconditioner (*t_p*), the number of BiCGStab iterations (*its*) and the time for the iterations (*t_it*). The parameters to apply the dropping rules in both preconditioners were chosen so that the preconditioners have similar size. In this way we can easily check their relative efficiency, but in general algebraic preconditioning it is not guaranteed that if we increase its size by changing its input parameters a more efficient preconditioner is obtained. It often happens that the dependence of the number of iterations on the size of the preconditioner is rather erratic, as we also show below, and such behavior is more likely to be observed when solving nonsymmetric problems. We are also interested in sparse preconditioners since we believe that only the resulting preconditioned iterative methods have most of its potential to be routinely used for solving large problems. We tend to believe that their future role will be in their coupling with an appropriate outer, possibly hierarchical, framework and, maybe, with an additional pivoting. This fact also motivates our interest in robustness of the algebraic preconditioners.

The results in Table 2 show that the NBIF algorithm is able to produce sparse

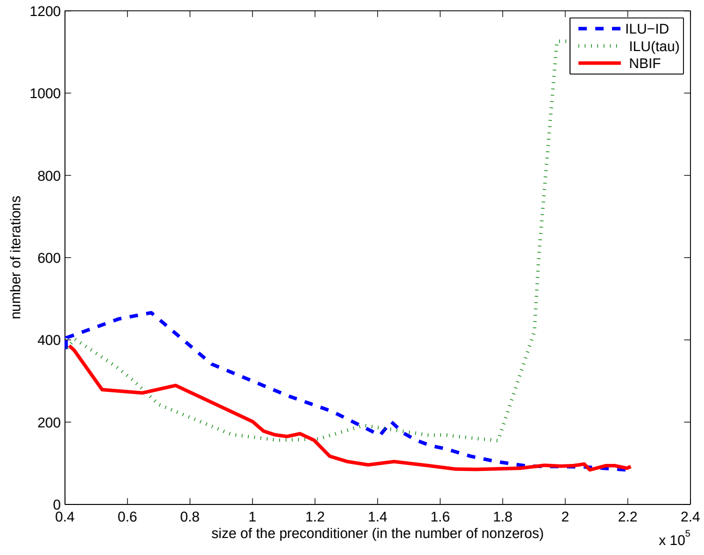

Fig. 5.1. *The sizes of NBIF, ILU-ID and ILU(τ) preconditioners (in numbers of their nonzeros) versus iteration counts of the preconditioned BiCGStab method for the matrix CHEM_MASTER1.*

preconditioners which are similarly efficient as the ILU-ID preconditioners of similar size. The time to compute the new preconditioner is larger, but it scales well with matrix size and it does not seem to be prohibitive, see the comparison of timings in the SPD case with the timings for the RIF preconditioner in [12]. The NBIF timings just reflects the fact that we approximate both direct and inverse factors of the system matrix. At this moment we would like to avoid any strong judgements based on Table 2 but, as we will see later, this standard form of presentation does not tell the whole story.

The four figures 5.1, 5.2, 5.3 and 5.4 reveal an actual potential of the new approach. They show a detailed comparison of the NBIF and the two ILU methods by generating preconditioners of various sizes. In particular, the figures show dependence of the number of iterations of the preconditioned BiCGStab method on the size of the preconditioner for the matrices used in our experiments. Note that for denser decompositions, the behavior of the iterative method preconditioned by ILU(τ) of CHEM_MASTER1 is rather erratic. The solver no longer converges for the sizes between 220,000 and 470,000, again converges with the number of nonzeros between 470,000 and 755,000 (typically around 200 iterations), and, it finally converges, for very dense decompositions with the number of nonzeros larger than approximately 5,000,000. For the sizes in between the BiCGStab with ILU(τ) does not converge. The ILU-ID decomposition is for this matrix very stable and its iteration count further decreases with the decomposition size. Our implementation of the NBIF method was not able to generate incomplete decomposition denser than those depicted in Figure 5.1. This is probably caused by the restricted size of data structures for rows of V and $\tilde{V}$ and it also restricts its memory demands. In addition, the NBIF method seems to be perfectly robust over a wide spectrum of decomposition sizes and it is very good even when it is very sparse.

For the matrix POISSON3DB (see Figure 5.2, ILU-ID decompositions with more than approximately 250000 nonzero entries are unstable and the preconditioned iterative method does not converge. The iterative method then converges again for very

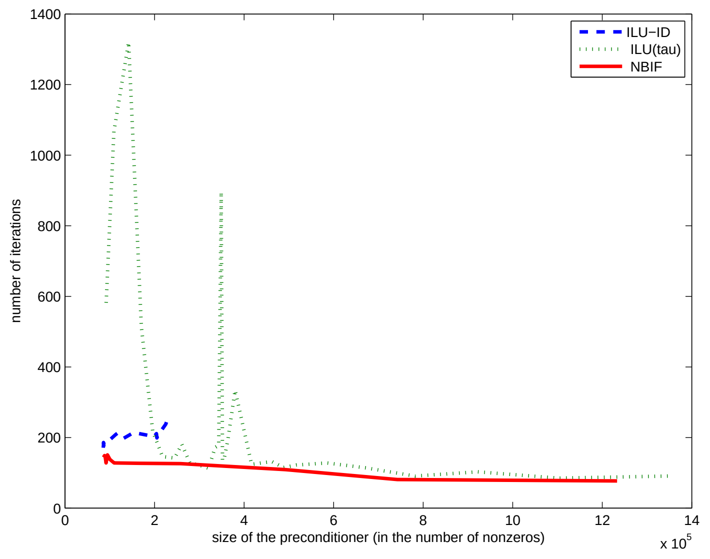

FIG. 5.2. *The sizes of NBIF, ILU-ID and ILU(τ) preconditioners (in numbers of their nonzeros) versus iteration counts of the preconditioned BiCGStab method for the matrix POISSON3DB.*

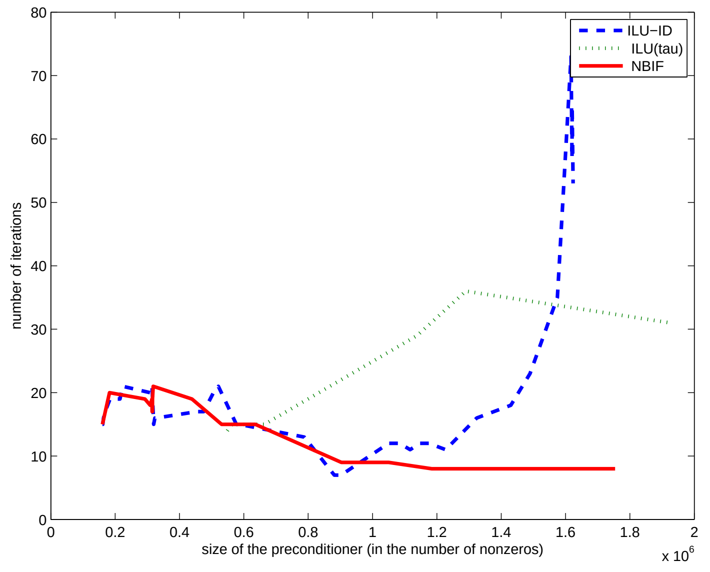

FIG. 5.3. *The sizes of NBIF, ILU-ID and ILU(τ) preconditioners (in numbers of their nonzeros) versus iteration counts of the preconditioned BiCGStab method for the matrix MAJORBASIS.*

dense preconditioners with approximately ten-times or more nonzeros having also prohibitive timings. Both ILU methods contrast with the behavior of the new approach which is very robust. High quality of the NBIF preconditioner is also clear from the Figure 5.3 where both ILU methods are unstable for decompositions which are denser than those depicted for the matrix MAJORBASIS. As for the matrix EPB3 on Figure 5.4, our implementation does not allow to generate the NBIF preconditioner of larger size, as we discussed above. Its ILU competitors are not unstable, but they are surpassed for most of the NBIF preconditioner sizes.

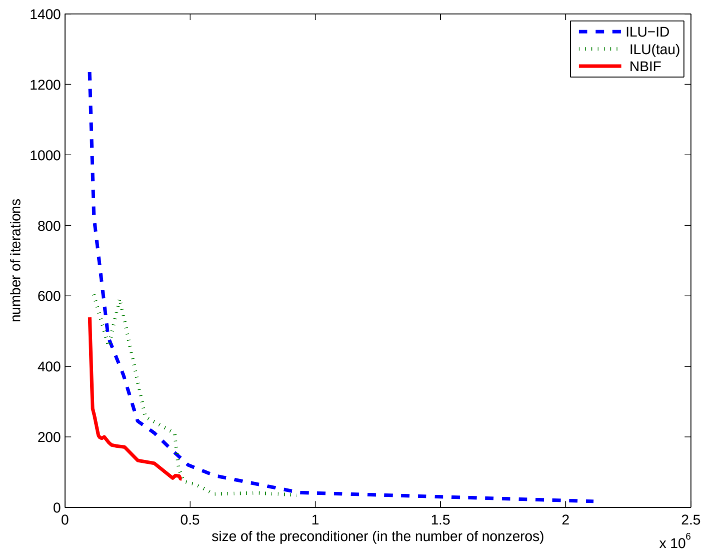

FIG. 5.4. *The sizes of NBIF, ILU-ID and ILU(τ) preconditioners (in numbers of their nonzeros) versus iteration counts of the preconditioned BiCGStab method for the matrix EPB3.*

6. Conclusions and future work. In this paper we give a new insight into the mutual dependence of the direct and inverse factors in the decomposition derived from the $(s^{-1}I - A^{-1})^{-1}$ biconjugation. Based on this dependence, we proposed improvements in the basic algorithmic scheme of the BIF algorithm, and a nonsymmetric version called NBIF. The last algorithm uses the entries of the direct factors to construct their inverses and vice versa even without an explicit connection to a chosen dropping scheme. In the experimental section, the resulting decomposition is used as a preconditioner for the BiCGStab iterative method. In spite of the specific choice of data structures, the new algorithm seems to be very robust and it has the potential to become a useful tool for the solution of large and sparse linear systems.

Some problems can be considered for a future work. One of the crucial tasks in the preconditioning is to explore more possibilities for the data structures and dropping. One way would be to abandon storing Z and V_s by rows completely, and storing them only by columns. Then fast evaluation of the involved dot products should be solved. Further, a floating-point analysis of more possible approaches, especially for small drop tolerances, but also for complete decompositions would be strongly desirable. In addition the role of the parameter s (or its generalization S) in the stability deserves a detailed study as well as possible strategies to compute these parameters on the fly. We believe that the new procedure may be useful also in further applications. For example, as mentioned above, joint computation of both direct and inverse factors may be useful for condition estimators. The straightforward columnwise approach of the BIF and NBIF algorithms may imply sufficiently efficient high-performance implementations. Our future work will be also concerned with enhancing the basic scheme by a block, and finally both block and multilevel implementations.

Acknowledgment. We would like to thank to the anonymous referees. Their comments helped to improve the paper significantly. Parts of this paper were written while the fourth author was a visitor at the Universitat Politècnica de València: the hospitality and support are greatly appreciated.

REFERENCES

[1] M. Benzi, J. K. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM J. Sci. Comput.*, 22(4):1318–1332, 2000.

[2] M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Sci. Comput.*, 22(4):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(5):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. Linear Algebra Appl.*, 10(5-6):385–400, 2003.

[6] C. H. Bischof. Incremental condition estimation. *SIAM J. Matrix Anal. Appl.*, 11:312–322, 1990.

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

[8] M. Bollhöfer. A robust and efficient *ILU* that incorporates the growth of the inverse triangular factors. *SIAM J. Sci. Comput.*, 25(1):86–103, 2003.

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

[10] R. Bridson. Multi-resolution approximate inverses. Master's thesis, Department of Computer Science, Waterloo University, Ontario, Canada, 1999.

[11] R. Bru, J. Cerdán, J. Marín, and J. Mas. Preconditioning sparse nonsymmetric linear systems with the Sherman-Morrison formula. *SIAM J. Sci. Comput.*, 25(2):701–715, 2003.

[12] R. Bru, J. Marín, J. Mas, and M. Tůma. Balanced incomplete factorization. *SIAM J. Sci. Comput.*, 30(5):2302–2318, 2008.

[13] J. Cerdán, T. Faraj, N. Malla, J. Marín, and J Mas. A block approximate inverse preconditioner for sparse nonsymmetric linear systems. *ETNA*, To appear, 2010.

[14] M. T. Chu, R. E. Funderlic, and G. H. Golub. A rank-one reduction formula and its applications to matrix factorizations. *SIAM Review*, 37:512–530, 1995.

[15] T. A. Davis. *University of Florida Sparse Matrix Collection.* available online at http://www.cise.ufl.edu/∼davis/sparse/, NA Digest, vol. 94, issue 42, October 1994.

[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.*, 20:889–901, 1999.

[17] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal.*, 22:973–996, 2001.

[18] I. S. Duff and C. Vömel. Incremental norm estimation for dense and sparse matrices. *BIT*, 42:300–322, 2002.

[19] J. Duintjer Tebbens, 2008. Personal communication.

[20] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. The Yale Sparse Matrix Package (YSMP) – II : The non-symmetric codes. Technical Report No. 114, Department of Computer Science, Yale University, 1977.

[21] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Yale Sparse Matrix Package (YSMP) – I : The symmetric codes. *Int. J. Numer. Meth. in Eng.*, 18:1145–1151, 1982.

[22] L. Fox, H. D. Huskey, and J. H. Wilkinson. Notes on the solution of algebraic linear simultaneous equations. *Quart. J. Mech. and Appl. Math.*, 1:149–173, 1948.

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

[24] G. H. Golub and C. F. Van Loan. *Matrix Computations. 3rd ed.* The Johns Hopkins University Press, Baltimore and London, 1996.

[25] A. Greenbaum. Estimating the attainable accuracy of recursively computed residual methods. *SIAM J. Matrix Anal. Appl.*, 18:535–551, 1997.

[26] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–435, 1952.

[27] HSL, a Collection of Fortran codes for large-scale scientific computation, 2007. `http://www.hsl.rl.ac.uk`.

[28] S. A. Kharchenko, L. Yu. Kolotilina, A. A. Nikishin, and A. Yu. Yeremin. A robust AINV-type method for constructing sparse approximate inverse preconditioners in factored form. *Numer. Linear Algebra Appl.*, 8(3):165–179, 2001.

[29] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings. I. Theory. *SIAM J. Matrix Anal. Appl.*, 14(1):45–58, 1993.

[30] N. Li, Y. Saad, and E. Chow. Crout versions of ILU for general sparse matrices. *SIAM J. Sci.*

Comput., 25(2):716–728, 2003.

[31] J. Morris. An escalator process for the solution of linear simultaneous equations. *Philos. Mag.*, 37:106–120, 1946.

[32] A. Neumaier and M. Olschowka. A new pivoting strategy for Gaussian elimination. *Lin. Algebra Appl.*, 240:131–151, 1996.

[33] E. W. Purcell. The vector method of solving simultaneous linear equations. *J. Math. Phys.*, 32:150–153, 1953.

[34] Y. Saad. ILUT: a dual threshold incomplete *LU* factorization. *Numer. Linear Algebra Appl.*, 1(4):387–402, 1994.

[35] J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Ann. Math. Stat.*, 21:124–127, 1950.

[36] A. Yu. Yeremin and A. A. Nikishin. Factorized sparse approximate inverse preconditioning of linear systems with nonsymmetric matrices. *Zap. Nauchn. Sem. S.-Peterburg. Otdel. Mat. Inst. Steklov. (POMI)*, 284(Chisl. Metody i Vopr. Organ. Vychisl. 15):18–35, 269, 2002.

[37] J. F. Yin. A class of preconditioners based on splitting for nonsymmetric system of linear equations, *preprint*, 2008.