Applied Mathematics E-Notes, 21(2021), 163-171 ©
Available free at mirror sites of http://www.math.nthu.edu.tw/∼amen/
ISSN 1607-2510

On Computing Sparse Approximate Inverse Factor Of SPD Matrices*

Davod Khojasteh Salkuyeh†‡

Received 16 March 2020

Abstract

In this paper, we give an approach for computing a sparse approximate inverse factor for symmetric positive definite matrices. Each column of the computed factor contains at most two nonzero entries. The computed inverse factor can be applied as a pre-scaling for symmetric positive definite linear systems. Numerical results are given to show the efficiency of the method.

1 Introduction

Consider the system of linear equations of the form

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

where the coefficient matrix $A \in \mathbb{R}^{n\times n}$ is large, sparse, and symmetric positive definite (SPD) and $x, b \in \mathbb{R}^n$. Iterative methods which combine preconditioning techniques are among the most efficient techniques for solving the system (1). More precisely, iterative methods usually involve a second matrix that transforms the coefficient matrix into one with a more favorable spectrum. The transformation matrix is called a preconditioner. Let us assume that A has the $\mathrm{L^TDL}$ factorization $A = L^TDL$, where L^T and D are lower unit-triangular and diagonal matrices, respectively. Since the matrix A is SPD, it is easy to see the diagonal entries of the matrix D are positive. Letting $M = L^{-1}$, we see that $M^TAM = D$. The matrix M is called the inverse factor of A. In this case, we have $D^{-\frac{1}{2}}M^TAMD^{-\frac{1}{2}} = I$, where I is the identity matrix. Now, let W be a sparse approximation of the matrix $MD^{-\frac{1}{2}}$. In this case, we have $W^TAW \approx I$, and solving the system

$$W^TAWy = W^Tb, \quad x = Wy, \tag{2}$$

using the conjugate gradient (CG) method is recommended.

Eq. (2) is called split-preconditioned system. Also, there are left- and right-preconditioned systems [1, 9].

There are many ways to compute sparse approximate inverse factors of an SPD matrix. Among them are the FSAI algorithm proposed by Kolotilina and Yeremin [6, 7], the AIB (approximate inverse factors via a bordering technique) algorithm proposed by Saad [3, 9] and the SAINV algorithm presented by Benzi et al. in [2]. In this paper, using a well-known iterative method for computing the solution of a SPD system of linear equations and the AIB algorithm we propose a method for computing a approximate inverse factor of an SPD matrix, where each column of the inverse factor contain at most two nonzero entries.

This paper is organized as follows. In Section 2, an algorithm for solving an SPD linear system and the AIB algorithm are presented. Section 3 is devoted to the proposed method. Numerical experiments are given in section 3. Concluding remarks are drawn in Section 5.

*Mathematics Subject Classifications: 65F10, 65F50.
†Faculty of Mathematical Sciences, University of Guilan, Rasht, Iran
‡Center of Excellence for Mathematical Modelling, Optimization and Combinational Computing (MMOCC), University of Guilan, Rasht, Iran

2 Solving SPD Systems and the AIB Algorithm

2.1 Solving SPD Linear Systems

In this section, we derive an approach for solving SPD linear systems of equations which is provided by a projection method. This approach will be useful for later applications. Let $\mathcal{L} = \mathcal{K} = \text{span}\{e_i\}$, where e_i is the i-th column of the identity matrix. Let also x be an approximate solution of Eq. (1). We look for an approximate solution x_{new} to the system (1) by imposing the conditions that x_{new} belongs to $x + \mathcal{K}$ and that the new residual vector be orthogonal to $\mathcal{L}$, i.e.,

$$\text{Find } x_{new} \in x + \mathcal{K}, \quad \text{such that } r_{new} := b - Ax_{new} \perp \mathcal{L}.$$

This framework is known as the Petrov-Galerkin condition. Hence the new approximate solution of the system (1) takes the form $x_{new} = x + \delta$, where $\delta \in \mathcal{K}$, i.e.,

$$x_{new} = x + \alpha e_i, \tag{3}$$

for some $\alpha \in \mathbb{R}$. Now, we have

$$r_{new} = b - Ax_{new} = r - \alpha A e_i, \tag{4}$$

where the vector r denotes the initial residual vector $r = b - Ax$. Then the Petrov-Galerkin condition $r_{new} \perp \mathcal{L}$ yields

$$\alpha = \frac{1}{a_{ii}} e_i^T r = \frac{1}{a_{ii}} r_i, \tag{5}$$

where r_i is the i-th entry of r. Therefore, we have

$$\begin{aligned} (x_{new})_i &= x_i + \frac{1}{a_{ii}} r_i, \\ (x_{new})_j &= x_j, \quad j \neq i. \end{aligned}$$

Theorem 1 ([9]) *Assume that i is selected at each projection step to be the index of a component of largest absolute value in the current residual vector $r = b - Ax$. Then*

$$\|d_{new}\|_A \leq (1 - \frac{1}{n\kappa_2(A)})^{1/2} \|d\|_A, \tag{6}$$

where $d_{new} = A^{-1}b - x_{new}$, $d = A^{-1}b - x$, and $\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2$ is the spectral condition number of A. Here, for a vector $v \in \mathbb{R}^n$, $\|v\|_A = \sqrt{v^T A v}$. Eq. (6) shows that the method converges for any initial guess.

2.2 The AIB Algorithm

The AIB algorithm is among the algorithms for computing the inverse factors of a matrix. Here is a brief description of the algorithm in the case that the matrix is SPD. In the AIB algorithm, the sequence of matrices

$$A_{k+1} = \begin{pmatrix} A_k & v_k \\ v_k^T & \alpha_{k+1} \end{pmatrix}, \quad k = 1, \ldots, n-1,$$

where A_k is the kth leading principle sub-matrix of A, $v_k = (a_{1,k+1}, \ldots, a_{k,k+1})^T$ and $\alpha_{k+1} = a_{k+1,k+1}$. If the inverse factor U_k is available for A_k, i.e., $U_k^T A_k U_k = D_k$, then the inverse factor U_{k+1} for A_{k+1} will be obtained by writing

$$\begin{pmatrix} U_k^T & 0 \\ -z_k^T & 1 \end{pmatrix} \begin{pmatrix} A_k & v_k \\ v_k^T & \alpha_{k+1} \end{pmatrix} \begin{pmatrix} U_k & -z_k \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} D_k & 0 \\ 0 & \delta_{k+1} \end{pmatrix},$$

in which

$$A_k z_k = v_k, \tag{7}$$

$$\delta_{k+1} = \alpha_{k+1} - z_k^T v_k. \tag{8}$$

Eq. (8) can be exploited if the system (7) is solved exactly. Otherwise, we should use

$$\delta_{k+1} = \alpha_{k+1} - v_k^T z_k - z_k^T (v_k - A_k z_k) = \alpha_{k+1} - z_k^T (v_k + r_k), \tag{9}$$

instead of Eq. (8) where $r_k = v_k - A_k z_k$. Starting from $k = 1$, this procedure suggests an algorithm for computing the inverse factor of A. If a sparse approximate solution of Eq. (7) is computed, then an approximate factorization of A^{-1} is obtained. This scheme can be summarized as follows.

Algorithm 2 ***AIB algorithm***

1. *Set* $A_1 = [a_{11}]$, $U_1 = [1]$ *and* $\delta_1 = a_{11}$.
2. *For* $k = 1, \ldots, n-1$ *Do: (in parallel).*
3. *Compute a sparse approximate solution to* $A_k z_k = v_k$ *and return the residual* $r_k = v_k - A_k z_k$
4. *Compute* $\delta_{k+1} = \alpha_{k+1} - z_k^T (v_k + r_k)$.
5. *Form* U_{k+1} *and* D_{k+1}
6. *EndDo*
7. $U := U_n$ *and* $D := D_n$.

This algorithm returns U and D such that $U^T A U \approx D$. For SPD matrices, it can be easily seen that δ_{k+1} is always positive independently of the accuracy with which the system (7) is solved [9]. Hence the AIB algorithm is well-defined for SPD matrices.

3 A Sparse Approximate Inverse

We consider the projection method studied in Subsection 2.1 and implement it for computing an approximate solution of the system $A_k z_k = v_k$ in step 4 of Algorithm 2. First, let $z_k = 0$ be an approximate solution of $A_k z_k = v_k$. In this case

$$\begin{cases} \delta_1 = a_{11}, \\ \delta_{k+1} = \alpha_{k+1} = a_{k+1,k+1}, \quad k = 1, \ldots, n-1. \end{cases}$$

Hence, $U = I$ and $D = \mathrm{diag}(A)$. Note that $a_{ii} > 0$, since A is SPD. Therefore we have

$$D^{-1/2} A D^{-1/2} \approx I. \tag{10}$$

Hence, $D^{-1/2}$ can be used as a diagonal preconditioner for the system (1). Now, we try to improve the preconditioner. To do this, we improve the solution $z_k = 0$ of $A_k z_k = v_k$. We use the projection method described in Subsection 2.1. Obviously $r_k = v_k - A_k z_k = v_k$. Hereafter, let i be the index of a component of largest absolute value in r_k. This choice provides the assumptions of Theorem 3. In this case, Eq. (5) results in

$$\alpha = \frac{1}{a_{ii}} e_i^T r_k = \frac{a_{i,k+1}}{a_{ii}},$$

and Eq. (3) gives the new approximate solution $z_{k,new} = z_k + \alpha e_i = \alpha e_i$, of $A_k z_k = v_k$. Let $r_{k,new}$ be the residual vector of $z_{k,new}$. Then, we have

$$e_i^T r_{k,new} = e_i^T (v_k - A_k z_{k,new}) = a_{i,k+1} - \alpha e_i^T A_k e_i = a_{i,k+1} - \alpha a_{ii} = 0.$$

Therefore

$$\begin{aligned}\delta_{k+1} &= a_{k+1,k+1} - \alpha e_i^T(v_k + r_k) = a_{k+1,k+1} - \alpha e_i^T v_k = a_{k+1,k+1} - \frac{a_{i,k+1}^2}{a_{i,i}} \\ &= \frac{1}{a_{ii}} \begin{vmatrix} a_{ii} & a_{i,k+1} \\ a_{k+1,i} & a_{k+1,k+1} \end{vmatrix} \qquad (\text{from } a_{i,k+1} = a_{k+1,i}) \\ &= \frac{1}{a_{ii}} \det(E^T A E) > 0,\end{aligned}$$

where $E = (e_i, e_{k+1})$. Note that, the matrix $E^T A E$ is SPD and hence $\det(E^T A E) > 0$. Therefore, the approximate inverse factor U computed in step 7 of the AIB algorithm has at most two nonzero entries in each column and

$$\begin{aligned} U_{kk} &= 1, \qquad & k = 1, \ldots, n, \\ U_{ik} &= -\frac{a_{ik}}{a_{ii}}, \qquad & k = 2, \ldots, n, \end{aligned}$$

and

$$\begin{aligned} \delta_1 &= a_{11}, \\ \delta_k &= a_{kk} - \frac{a_{ik}^2}{a_{ii}}, \qquad & k = 2, \ldots, n. \end{aligned}$$

Hence $U^T A U \approx S$, where $S = \mathrm{diag}(\delta_1, \ldots, \delta_n)$. Defining $W = US^{-1/2}$, we get $W^T A W \approx I$. It can be easily seen that

$$W_{kk} = \frac{1}{\sqrt{\delta_k}}, \qquad k = 1, \ldots, n, \tag{11}$$

$$W_{ik} = -\frac{a_{ik}}{a_{ii}\sqrt{\delta_k}}, \qquad k = 2, \ldots, n. \tag{12}$$

Theorem 3 *Let W be the approximate inverse factor computed by the Algorithm 2. Then* $\mathrm{diag}(W^T A W) = I$.

Proof. We have

$$\begin{aligned} W_{kk} &= W_{ik}(a_{ii}W_{ik} + a_{ik}W_{kk}) + W_{kk}(a_{ki}W_{ik} + a_{kk}W_{kk}) = a_{ii}W_{ik}^2 + 2a_{ik}W_{ik}W_{kk} + a_{kk}W_{kk}^2 \\ &= a_{ii} \times \frac{a_{ik}^2}{a_{ii}^2\delta_k} - 2a_{ik}\frac{a_{ik}}{a_{ii}\sqrt{\delta_k}}\frac{1}{\sqrt{\delta_k}} + a_{kk}\frac{1}{\delta_k} = \frac{1}{\delta_k}(a_{kk} - \frac{a_{ik}^2}{a_{ii}}) = \frac{1}{\delta_k} \times \delta_k = 1, \end{aligned}$$

which completes the proof. ■

Here, it is mentioned that the diagonal scaling has the property obtained in Theorem 3.

We now consider the special case when the matrix A is tridiagonal. Let

$$A = \begin{pmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & b_n \\ & & & b_n & a_n \end{pmatrix}.$$

In this case the matrix U can be written as

$$U = \begin{pmatrix} 1 & -\frac{b_2}{a_1} & & & \\ & 1 & -\frac{b_3}{a_2} & & \\ & & \ddots & \ddots & \\ & & & 1 & -\frac{b_n}{a_{n-1}} \\ & & & & 1 \end{pmatrix}.$$

and $D = \text{diag}(\delta_1, \delta_2, \ldots, \delta_n)$, with $\delta_1 = a_1$ and $\delta_k = a_k - \frac{b_k^2}{a_{k-1}}$, $k = 2, \ldots, n$. Obviously, the approximate inverse factor $W = UD^{\frac{1}{2}}$ is an upper bidiagonal matrix.

4 Applications

The proposed preconditioner can be directly used as a preconditioner for linear system of equations. But as we see in the next section the effectiveness of the proposed preconditioner for large matrices is limited. Hence combining it with other preconditioning techniques may be useful. The direct computation W^TAW is not economical. But, the preconditioning techniques which uses only matrix-vector multiplication such as the SAINV algorithm [2] may benefit from it. In fact W can be used as a scaling for the matrices.

Let A be a SPD block-tridiagonal matrix blocked in the form

$$A = \begin{pmatrix} G_1 & E_2 & & & \\ E_2^T & G_2 & E_3 & & \\ & \ddots & \ddots & \ddots & \\ & & E_{l-1}^T & G_{l-1} & E_l \\ & & & E_l^T & G_l \end{pmatrix}. \tag{13}$$

Preconditioning of the systems with the coefficient matrices of the above form have been presented in several papers (for example see [4, 5, 8, 10]). Let G be block-diagonal matrix consisting of the diagonal blocks G_i and Q the block strictly-upper triangular matrix consisting of the super-diagonal blocks E_i. This kind of matrices usually arise from discretization of the partial differential matrices. Then, A is of the form $A = Q^T + G + Q$. Let

$$\Lambda_1 = G_1, \qquad \Lambda_{k+1} = G_{k+1} - E_{k+1}^T \Lambda_k^{-1} E_{k+1}, \qquad k = 1, \ldots, l-1.$$

By the above notations and expressions we have $A = (L + \Lambda)\Lambda^{-1}(\Lambda + U)$ where $\Lambda = \text{blkdiag}(\Lambda_1, \ldots, \Lambda_l)$. Let Ω_k be an approximation of Δ_k^{-1}, where Δ_i are defined as

$$\Delta_1 = G_1, \qquad \Delta_{k+1} = G_{k+1} - E_{k+1}^T \Omega_k E_{k+1}, \quad k = 1, \ldots, l-1,$$

(see [10, 8]). Matrices G_i, $i = 1, \ldots, l-1$ are usually sparse (for PDEs, are usually tridiagonal) and of small size. Let W_k be the computed approximate bidiagonal inverse factor of Δ_k. Then we have $W_k^T \Delta_k W_k \approx I$. Hence, $\Delta_k^{-1} \approx W_k W_k^T$. Hence we can set $\Omega_k = W_k W_k^T$. In this case the block incomplete LU factorization of A takes the form $M = (\Delta + Q^T)\Delta^{-1}(\Delta + Q)$. For more details see [8].

In the implementation of the preconditioner M in a Krylov-subspace method like CG, in each iteration of the method two linear systems with matrices $\Delta + Q^T$ and $\Delta + Q$ should be solved which can be done easily, since these matrices are lower block and upper block triangular matrices, respectively. Furthermore, a matrix-vector multiplication with the matrix Δ should be done.

5 Numerical Examples

All the numerical experiments presented in this section were computed in double precision with some MATLAB (Release 13) codes on a Laptop 1.80GHz CPU and 6GB RAM.

For the first set of numerical experiments we choose two small-size matrices bcsstk03 ($n = 112$, $nnz = 640$, $\kappa_2(A) = 9.41 \times 10^6$) and bcsstm07 ($n = 420$, $nnz = 7252$, $\kappa_2(A) = 1.33 \times 10^4$) from the Harwell-Boeing collection (https://math.nist.gov). Figures 1 and 2 display the eigenvalue distribution of the matrices A, $D^{-1/2}AD^{-1/2}$ and W^TAW for the matrices bcsstk03 and bcsstm07, respectively. Here, $D = \text{diag}(A)$ and W is the approximate inverse computed by the proposed method. As we observe for both of the matrices the eigenvalue distribution of the matrix W^TAW are more clustered around $(1, 0)$ than those of the matrix $D^{-1/2}AD^{-1/2}$. This shows that the spectral condition number of W^TAW is less than that of $D^{-1/2}AD^{-1/2}$.

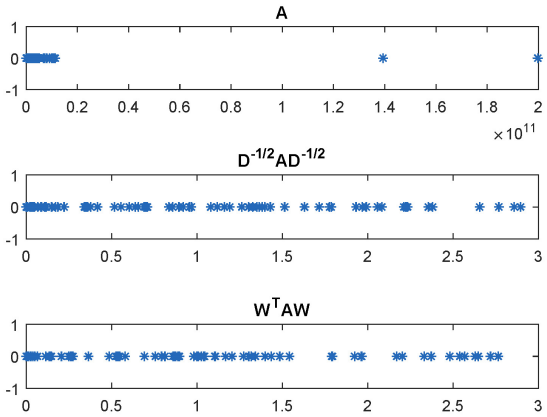

Figure 1: Eigenvalue distribution of A, $D^{-1/2}AD^{-1/2}$ and W^TAW for the matrix bcsstk03.

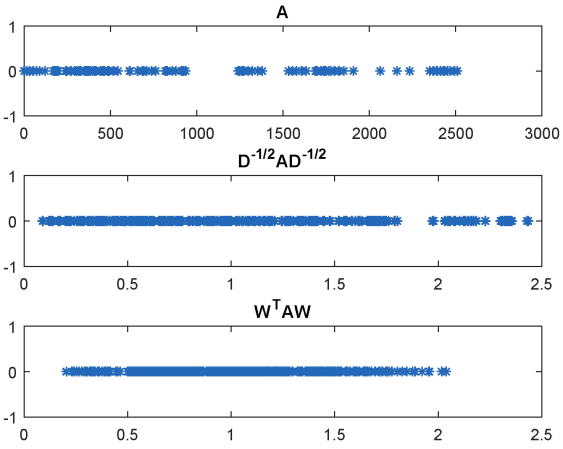

Figure 2: Eigenvalue distribution of A, $D^{-1/2}AD^{-1/2}$ and W^TAW for the matrix bcsstm07.

For the second set of the numerical experiments, some matrices from Harwell-Boeing collection are chosen. These matrices with their generic properties are shown in Table 1. Column 1 of this table gives the name of the test matrices; Column 2 lists the size (n) of the matrix; Column 3 gives the number of the nonzero entries (nnz) of the matrix; Column 4 presents the spectral condition number of the matrix ($\kappa(.)$); Columns 5 and 6 report the number of iterations (Iters) and the CPU time of CG algorithm for convergence, respectively. Timings are in seconds. It is necessary to mention that for all of the numerical experiments in this section the right-hand side of the system of equations were taken such that the exact solution is $x = (1, \ldots, 1)^T$ and the iteration is stopped as soon as the residual 2-norm is reduced by a factor of 10^7. The initial guess was taken to be zero vector.

Table 1: First set of test problems information and the number of iterations of CG algorithm for the convergence.

matrix	n	nnz	$\kappa(.)$	iters	CPU time	
NOS1	237	1017	2.53×10^7	1933	0.10	
NOS2	957	4137	4.97×10^9	6602	0.28	
1138_bus	1138	4054	1.23×10^7	1959	0.08	
bcsstk12	1473	32241	5.25×10^8	2756	0.19	
s3rmt3m3	5357	207123	4.44×10^{10}	9409	2.99	
s3rmt3m1	5489	217669	4.57×10^{10}	8837	3.26	
s2rmt3m1	5489	217681	4.84×10^8	7275	3.28	
s2rmq4m1	5489	263351	3.22×10^8	9955	4.11	
bcsstk17	10974	428650	1.95×10^{10}	9927	12.67	

We compare the numerical results of the split-preconditioned CG algorithm [9] for solving linear systems with preconditioner $D^{-1/2}$ where $D = \text{diag(A)}$ (Precon. 1), and W, where its entries are defined by Eqs. (11) and (12) (Precon. 2). Numerical results are given in Table 2. This table, for each of the preconditioners, reports the number of split-preconditioned CG iterations for convergence (Iters) and the setup time for computing the preconditioner and iterations (CPU Time). As we observe, the proposed preconditioner outperform the proposed preconditioner. We also see that diagonal scaling does not improve the convergence of the CG algorithm for the matrices s3rmt3m3 and s3rmt3m1.

Table 2: Numerical results for the first set of test matrices.

	Precon. 1		Precon. 2	
matrix	Iters	CPU time	Iters	CPU time
NOS1	362	0.03	242	0.03
NOS2	3115	0.17	2049	0.15
1138_bus	848	0.06	255	0.05
bcsstk12	1774	0.19	735	0.13
s3rmt3m3	9700	4.43	2488	1.41
s3rmt3m1	9151	4.21	2468	1.53
s2rmt3m1	2176	1.06	961	0.72
s2rmq4m1	1531	0.93	779	0.64
bcsstk17	2435	2.49	1442	2.01

For the third set of the numerical experiments we consider the equation

$$-\Delta u + g(x, y)u = f(x, y), \qquad (x, y) \in \Omega = (0, 1) \times (0, 1).$$

Discretizing this equation on an $nx \times ny$ grid, by using the second order centered differences for the Laplacian

gives a linear system of equations of order $n = nx \times ny$ with n unknowns $u_{ij} = u(ih, jh)(1 \leq i, j \leq n)$:

$$-u_{i-1,j} - u_{i,j-1} + (4 + h^2 g(ih, jh))u_{ij} - u_{i+1,j} - u_{i,j+1} = h^2 f(ih, jh).$$

Let $nx = ny$. It can be seen that the coefficient matrix of this system is of the form (13). Hence, we apply the method presented in Section 4. We set $g(x, y) = -10e^{xy}$. It can easily be verified that, for small enough h, the coefficient matrix of this system is SPD. We give the numerical results for $nx = 100, 200, 300, 400, 500$ in Table 3. We compare the number of iterations of the CG algorithm for the original system and left-preconditioned CG in conjunction with the proposed method in Section 4. Since, our codes have not been optimized for highest efficiency and we do not report timings. All of the assumptions are as before. As we see the proposed preconditioner reduces the iteration number of CG by a factor of about 6.

Table 3: Numerical results for the third set of the experiments.

nx	n	nnz	Unpre.	Precon.	
$nx = 100$	10000	49600	276	53	
$nx = 200$	40000	199200	545	92	
$nx = 300$	90000	448800	809	129	
$nx = 400$	160000	798400	1067	163	
$nx = 500$	250000	1248000	1307	201	

6 Conclusion and Future Work

We have proposed an approach for computing a sparse approximate inverse of a SPD matrix. Our numerical results show that the computed sparse approximate inverse can be used as a preconditioner for SPD systems and it outperforms the diagonal scaling from the number of iterations and the CPU time point of view. This method can be implemented for the normal equations by a little revision. Future work may focus on extending the proposed algorithm to general matrices.

References

[1] M. Benzi, Preconditioning techniques for large linear systems: A survey, J. Comput. Phys., 182(2002), 418–477.

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

[3] M. Benzi and M. Tuma, A comparative study of sparse approximate inverse preconditioners, Appl. Numer. Math., 30(1999), 305–340.

[4] Y. Chen, X. Tian, H. Liu, Z. Chen, B. Yang, W. Liao, P. Zhang, R. He and M. Yang, Parallel ILU preconditioners in GPU computation, Soft Comput., 22(2018), 8187–8205.

[5] Y.-H. Fan, L.-H. Wang, Y. Jia, X.-G. Li, X.-X. Yang and C.-C. Chen, Investigation of high-efficiency iterative ILU preconditioner algorithm for partial differential equation systems, Symmetry, 11(2019), 1461.

[6] L. Y. Kolotilina and A. Y. Yeremin, Factorized sparse approximate inverse preconditioning I. Theory, SIAM J. Matrix Anal. Appl., 14(1993), 45–58.

[7] L. Y. Kolotilina and A. Y. Yeremin, Factorized sparse approximate inverse preconditioning II: Solution of 3D FE systems on massively parallel computers, Int. J. High Speed Comput., 7(1995), 191–215.

[8] M. H. Koulaei and F. Toutounian, On computing of block ILU preconditioner for block tridiagonal systems, J. Comput. Appl. Math., 202(2007), 248–257.

[9] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS press, New York, 1995.

[10] D. K. Salkuyeh and F. Toutounian, BILUS: a block version of ILUS factorization, J. Appl. Math. Comput., 15(2004), 299–312.