J. Appl. Math. & Informatics Vol. **28***(2010), No. 5-6, pp. 1131 - 1141*
Website: http://www.kcam.biz

A SPARSE APPROXIMATE INVERSE PRECONDITIONER FOR NONSYMMETRIC POSITIVE DEFINITE MATRICES

DAVOD KHOJASTEH SALKUYEH

Abstract. We develop an algorithm for computing a sparse approximate inverse for a nonsymmetric positive definite matrix based upon the FFAPINV algorithm. The sparse approximate inverse is computed in the factored form and used to work with some Krylov subspace methods. The preconditioner is breakdown free and, when used in conjunction with Krylov-subspace-based iterative solvers such as the GMRES algorithm, results in reliable solvers. Some numerical experiments are given to show the efficiency of the preconditioner.

AMS Mathematics Subject Classification : 65F10, 65F50.
Key words and phrases : linear system, nonsymmetric, positive definite, sparse approximate inverse, preconditioning, FFAPINV, GMRES.

1. Introduction

Consider the linear system of equations

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

where the coefficient matrix $A \in \mathbb{R}^{n\times n}$ is nonsingular, large, sparse and $x, b \in \mathbb{R}^n$. An approximate solution of (1) is usually obtained by using a preconditioned iterative Krylov-type method, such as the GMRES [17] and the BiCGSTAB [18] methods. More precisely, to obtain good convergence rates, or even to converge, these methods are applied to the left preconditioned linear system

$$MAx = Mb,$$

or right preconditioned linear system

$$AMy = b, \quad x = My,$$

Received July 31, 2009. Revised August 18, 2009. Accepted September 22, 2009.
*Corresponding author.

where the matrix M is the preconditioner. Two-side preconditioning is also possible [1, 3, 16]. There are two types of preconditioners:

(1) Finding a matrix M such that $M = G^{-1}$ where G approximates A;

(2) Finding a matrix M that directly approximates A^{-1} ($M \approx A^{-1}$).

In this paper, we consider a preconditioner of the second type.

If the matrix A admits the LDU factorization $A = LDU$, where L and U^T are lower unitriangular matrices and $D = \mathrm{diag}(d_1, d_2, \ldots, d_n)$ is a diagonal matrix, then

$$L^{-1}AU^{-1} = D.$$

Letting $W = L^{-1}$ and $Z = U^{-1}$, we have $WAZ = D$. Obviously W and Z^T are lower unitriangular matrices. The matrices W and Z are called inverse factors of A. If $\tilde{W}$ and $\tilde{Z}^T$ be two sparse lower unitriangular matrices that approximate W and Z^T, respectively, and $\tilde{D}$ is a nonsingular diagonal matrix approximating D then $M = \tilde{Z}\tilde{D}^{-1}\tilde{W}$ ($\approx A^{-1}$) may be used as a preconditioner for (1). Here, we say that M is a sparse approximate inverse for A in the factored form.

There are several ways to compute sparse approximate inverse factors of a matrix. In [8, 9], Kolotilina and Yeremin have proposed the FSAI algorithm. The AIB algorithm, which is based on a bordering technique, was presented by Saad in [16]. The FSAI and AIB preconditioners are always correctly defined (at least in exact arithmetic) for any positive definite (not necessarily symmetric) matrix A and arbitrary sparsity patterns [7]. Benzi et al. proposed the AINV method in [4, 5] which is well defined (in exact arithmetic) if the matrix A is an H-matrix [5]. For symmetric positive definite (SPD) matrices, there exists a variant of the AINV method, denoted by SAINV (for Stabilized AINV), that is breakdown-free [2]. This algorithm is also presented with the name AINV-A, independently, by Kharchenko et al. in [7]. Recently, Rafiei and Toutounian in [15] have proposed another variant of the AINV algorithm say SAINV-NS which is free from breakdown for nonsymmetric positive definite (NSPD) matrices. Numerical experiments presented in [15] show that the SAINV-NS preconditioner is sparser and cheaper than the AINV-A preconditioner. Moreover, the SAINV-NS preconditioner can be computed without using the matrix A^T, whereas this is not correct for the AINV-A preconditioner.

Another method is the procedure presented by Zhang in [20]. He proposed a sparse approximate inverse technique for parallel preconditioning of general sparse matrices. The proposed algorithm is essentially due to Luo [11, 12, 13]. Since in this procedure the factorization is performed in backward direction, we call it BFAPINV (backward factored approximate inverse) algorithm. In [19], Zhang proposed an alternative procedure to compute the factorization in the forward direction, which we call it FFAPINV algorithm. In [10], Lee and Zhang have shown that the BFAPINV algorithm is well-defined for M-matrices. It can be easily seen that this is correct for the FFAPINV algorithm as well. In this paper, we develop an algorithm for computing a sparse approximate inverse for NSPD matrices based upon the FFAPINV algorithm.

This paper is organized as follows. In section 2, we review the FFAPINV algorithm. Section 3 is devoted to the main results and the new variant of the FFAPINV algorithm for NSPD matrices. Numerical experiments are given in section 4. Finally, we give some concluding remarks in section 5.

2. A review of the FFAPINV algorithm

Let W and Z be the inverse factors of $A = (a_{ij})$, i.e.,

$$WAZ = D, \tag{2}$$

where

$$W = (w_1^T, w_2^T, \dots, w_n^T)^T, \quad Z = (z_1, z_2, \dots, z_n), \quad D = \mathrm{diag}(d_1, d_2, \dots, d_n),$$

in which w_i's and z_i's are the rows and columns of W and Z, respectively. Using (2) we obtain

$$w_i A z_j = \begin{cases} d_i, & i = j \\ 0, & i \neq j. \end{cases} \tag{3}$$

From the structure of the matrices W and Z, we have (for more details see [19])

$$z_1 = e_1, \quad z_j = e_j + \sum_{i=1}^{j-1} \alpha_i z_i, \quad j = 2, \dots, n, \tag{4}$$

$$w_1 = e_1^T, \quad w_j = e_j^T + \sum_{i=1}^{j-1} \beta_i w_i, \quad j = 2, \dots, n, \tag{5}$$

where e_j is the jth column of the identity matrix.

First of all, we see that

$$d_1 = w_1 A z_1 = e_1^T A e_1 = a_{11}.$$

Now let $2 \leq j \leq n$ be fixed. Then from (3) and (4) and for $k = 1, \dots, j-1$, we have

$$\begin{aligned} 0 &= w_k A z_j \\ &= w_k A e_j + \sum_{i=1}^{j-1} \alpha_i w_k A z_i \\ &= w_k A_{*j} + \alpha_k w_k A z_k \\ &= w_k A_{*j} + \alpha_k d_k, \end{aligned}$$

where A_{*j} is the jth column of A. Therefore

$$\alpha_i = -\frac{w_i A_{*j}}{d_i}, \quad i = 1, \dots, j-1.$$

In the same manner

$$\beta_i = -\frac{A_{j*}z_i}{d_i}, \quad i = 1, \dots, j-1,$$

where A_{j*} is the jth row of A. Putting these results together gives the following algorithm for computing the inverse factors of A.

```
Algorithm 1:
1. $z_1 := e_1$, $w_1 := e_1^T$ and $d_1 := a_{11}$
2. For $j = 2, \dots, n$, Do
3.     $z_j := e_j$;   $w_j := e_j^T$
4.     For $i = 1, \dots, j-1$, Do
5.         $\alpha_i := -\frac{w_i A_{*j}}{d_i}$;   $\beta_i := -\frac{A_{j*} z_i}{d_i}$
6.         $z_j := z_j - \alpha_i z_i$;   $w_j := w_j - \beta_i w_i$
7.     EndDo
8.     $d_j := w_j A z_j$
9. EndDo
```

Some observation can be posed here. Multiplying both sides of (4) by $w_j A$ from the left yields $d_j = w_j A_{*j}$. In the same manner, we have $d_j = A_{j*}z_j$. Hence one can use this relation in step 8 of the above algorithm, avoiding matrix-vector multiplication. For symmetric matrices, we have $W = Z^T$. Hence the computational cost is halved. If, moreover, A is SPD, then $d_j = z_j^T A z_j > 0$. Hence the algorithm is well-defined for SPD matrices.

A sparse approximate inverse for the matrix A can be computed by incorporating a procedure for sparsifying the newly computed vectors z_j and w_j in step 6 of the algorithm. To do so, a dropping strategy is applied in two parts. Let $\tau > 0$ be given tolerance. If $|\alpha_i| < \tau$ ($|\beta_i| < \tau$) then updating z_j (w_j) in step 6 of the algorithm is skipped. After updating vectors z_j and w_j in step 6, their entries whose absolute values are smaller than τ are dropped. The drop tolerance $\tau = 0.1$ is very often the right one based on the numerical results reported in several papers [2, 5, 15, 19, 20] . Based on this kind of dropping strategy we can summarized the FFAPINV algorithm as follows.

```
Algorithm 2:  FFAPINV algorithm
1. $z_1 := e_1$, $w_1 := e_1^T$ and $d_1 := a_{11}$
2. For $j = 2, \dots, n$, Do
3.     $z_j := e_j$;   $w_j := e_j^T$
4.     For $i = 1, \dots, j-1$, Do
5.         $\alpha_i := -\frac{w_i A_{*j}}{d_i}$;   $\beta_i := -\frac{A_{j*} z_i}{d_i}$
6.         If $|\alpha_i| > \tau$, then $z_j := z_j - \alpha_i z_i$
7.         If $|\beta_i| > \tau$, then $w_j := w_j - \beta_i w_i$
8.         Drop entries of $z_j$ and $w_j$ whose absolute
           values are smaller than $\tau$
```

```
 9.         EndDo
10.         $d_j := w_j A z_j$
11. EndDo
```

Obviously the FFAPINV algorithm is well-defined for SPD matrices (in exact arithmetic), since all pivots are always positive, i.e., $d_j = z_j^T A z_j > 0$. When A is a NSPD matrix, the FFAPINV algorithm can breakdown in consequence of the occurrence of zero or negative pivot. In the next section, we see that a modification of this algorithm would be free from breakdown for NSPD matrices.

3. The FFAPINV algorithm for NSPD matrices

We first state and prove the following Lemma.

Lemma 1. *Let W and Z^T be lower unitriangular matrices such that $WAZ = D$, where $D = diag(d_1, d_2, \dots, d_n)$. Then*

$$d_j = z_j^T A z_j = w_j A w_j^T. \tag{6}$$

Proof. From $WAZ = D$, we have $Z^T AZ = Z^T W^{-1} D$. Obviously $Z^T W^{-1} D$ is a lower triangular matrix and $\text{diag}(Z^T W^{-1} D) = D$. Hence we conclude that $\text{diag}(Z^T AZ) = D$. Therefore $d_j = z_j^T A z_j,\ j = 1, 2, \dots, n$. The second equality can be proved in the same manner. □

Using this lemma a theorem can be stated for NSPD matrices as following.

Theorem 1. *Let A be a NSPD matrix and all the assumptions of Lemma 1 hold. Then $d_j > 0,\ j = 1, 2, \dots, n$.*

Proof. From Lemma 1, we have $d_j = z_j^T A z_j$, $j = 1, 2, \dots, n$. Therefore $d_j > 0$, since A is a NSPD matrix. □

This theorem helps us to modify the FFAPINV algorithm for NSPD matrices (FFAPINV-NSPD) as follows.

```
Algorithm 3:  FFAPINV-NSPD
1. $z_1 := e_1$, $w_1 := e_1^T$ and $d_1 := a_{11}$
2. For $j = 2, \dots, n$, Do
3.         $z_j := e_j$;    $w_j := e_j^T$
4.         For $i = 1, \dots, j-1$, Do
5.                 $\alpha_i := -\frac{w_i A_{*j}}{d_i}$;    $\beta_i := -\frac{A_{j*} z_i}{d_i}$
```

```
 6.            If $|\alpha_i| > \tau$, then $z_j := z_j - \alpha_i z_i$
 7.            If $|\beta_i| > \tau$, then $w_j := w_j - \beta_i w_i$
 8.            Drop entries of $z_j$ and $w_j$ whose absolute $\tau$
               values are smaller than $\tau$
 9.        EndDo
10.        $d_j := A_{j*} z_j$
11.        If $d_j = 0$, then  $d_j := z_j^T A z_j$
12.    EndDo
```

To avoid the matrix-vector multiplication in computing $d_j := z_j^T A z_j$, we first compute d_j via $d_j := A_{j*} z_j$. If $d_j = 0$, then we use $d_j := z_j^T A z_j$. Obviously, this algorithms is free from breakdown for NSPD matrices in exact arithmetic. However, very small pivots may occur in computations. In practice, if the absolute value of the computed d_j is less than 10^{-15} then, as it was suggested in [3, 7], we replace it by $\operatorname{sgn}(d_j) \times 10^{-1}$.

Two propositions have been presented in [20] for exploiting the sparsity pattern of the original matrix for the BFAPINV algorithm. This propositions can also be used for the FFAPINV algorithm by some simple modifications and we use them in our computation.

4. Numerical examples

In order to evaluate the effectiveness of the proposed algorithm we apply the FFAPINV-NSPD algorithm on some NSPD linear systems of equations. All the numerical experiments presented in this section were computed in double precision using Fortran PowerStation version 4.0 on a Pentium 4 PC, with a 3.06 GHz CPU and 1.00GB of RAM. We divide this section into three parts. In the first two parts, we present the numerical results of the FFAPINV-NSPD algorithm in conjunction with the restarted GMRES with restart every m steps, namely GMRES(m) algorithm [16, 17] with and without left preconditioning [16]. CPU timings are all given in seconds and were measured with the function `etime()`. The initial guess was always $x_0 = 0$, b was selected such that the exact solution was $x = (1, 1, , \ldots , 1)^T$, and the stopping criterion was

$$\frac{\|b - Ax_i\|_2}{\|b\|_2} < 10^{-10}.$$

The maximum number of iterations was 10000. In all the tables a dagger (†) indicates no convergence of the iterative method. We do not use any scaling or reordering for the original coefficient matrix.

The first set of the test problems were derived from the five point discretization of the following partial differential equation

$$-(bu_x)_x - (cu_x)_x + du_x + (du)_x + eu_y + (eu)_y + fu = g,$$

TABLE 1. Numerical results for the first set of test problems.

		No prec.		Preconditioned system					
Matrix	nnz	Its	Time	τ	ρ	P-time	It-time	T-time	P-Its
PDE4900	24220	173	3.33	0.1	2.29	0.34	0.83	1.17	35
				0.2	0.88	0.31	0.93	1.24	44
PDE6400	31680	192	4.86	0.1	2.19	0.56	1.33	1.89	43
				0.2	0.88	0.53	1.34	1.87	48
PDE8100	40140	226	7.19	0.1	2.09	0.89	2.22	3.11	57
				0.2	0.87	0.86	2.01	2.87	53
PDE10000	49600	258	10.09	0.1	2.00	1.32	2.47	3.79	51
				0.2	0.86	1.31	3.70	5.01	79
PDE12100	60060	319	15.64	0.1	1.92	1.93	3.45	5.38	59
				0.2	0.84	1.90	5.18	7.08	97

in the unit square $(0,1)\times(0,1)$, where

$$\begin{aligned} b(x,y) &= e^{-xy}, \quad c(x,y)=e^{xy}, \quad d(x,y)=\beta(x+y), \\ e(x,y) &= \gamma(x+y), \quad \text{and} \quad f(x,y)=1/(1+x+y), \end{aligned}$$

with parameters $\beta = 20$ and $\gamma = 0$. Note that the matrix A resulting from the discretization remains NSPD, independent of these parameters [17]. A Fortran code, say MATPDE, is available at Matrix-Market website [14] for generating these matrices. Let n be the number of interior nodes on each side of the square. Using MATPDE and taking $n = 70, 80, 90, 100$ and 110, we generate matrices PDE4900, PDE6400, PDE8100, PDE10000 and PDE12100 with orders 4900, 6400, 8100, 10000, and 12100, respectively. The drop tolerances $\tau = 0.1$ and $\tau = 0.2$ were used for preserving the sparsity of the preconditioner. The numerical results are given in Table 1. For each matrix, the number of nonzero entries, nnz, and the number of iterations (Its) and time required to solve the linear system using the GMRES(5) without preconditioning are presented. In the last six columns of Table 1, the numerical results of solving preconditioned systems with left preconditioned GMRES(5) algorithm were given. Here, P-time, It-time, T-time and P-Its stand for the CPU time for constructing the preconditioner, the required time for the convergence of the GMRES(5) with left preconditioning, T-time=P-time+It-time and the number of the iterations for the convergence, respectively. Meanwhile, $\rho = (nnz(W) + nnz(Z))/nnz(A)$, where $nnz(X)$ means the number of nonzero entries of the matrix X. Numerical results presented in Table 1 show that the proposed preconditioner is robust and effective for NSPD linear system of equation. As we see, the proposed preconditioner reduces the number of required iterations for the convergence by about a factor 5 for $\tau = 0.1$ and 4 for $\tau = 0.2$.

It is well-known that the matrix A is positive definite if and only if $(A+A^T)/2$ is SPD. Hence, if A is an SPD matrix, then the matrix $S = A + 0.5L - 0.5L^T$ is NSPD, where L is the strictly lower triangular part of A. Note that the number

Table 2. Numerical results for the second set of test problems.

			No prec.		Preconditioned system					
Matrix	n	nnz	Its	Time	τ	ρ	P-time	It-time	T-time	P-Its
BC23	3134	45178	†	-	0.05	1.29	0.34	0.45	0.79	6
BC38	8032	355460	†	-	0.05	1.15	1.54	1.23	2.77	4
BC18	11948	149090	†	-	0.05	1.41	2.81	0.88	3.69	3
BC25	15439	252241	†	-	0.05	2.28	3.92	2.26	6.18	5
S1Q1	5489	262411	797	113.9	0.2	1.45	0.91	4.96	5.87	22
S2Q1	5489	263351	†	-	0.2	2.94	1.38	6.39	7.77	21
S1T1	5489	217651	†	-	0.2	1.64	0.88	1.73	2.61	8
S2T1	5489	217607	†	-	0.2	2.78	1.11	4.01	5.12	15

of nonzero entries of A and S are the same. For the second set of numerical experiments, we consider eight matrices BC23, BC38, BC18, BC25, S1Q1, S2Q1, S1T1 and S2T1 obtained from applying S on, respectively, matrices BCSSTK23, BCSSTK38, BCSSTK18, BCSSTK25, S1RMQ4M1, S2RMQ4M1, S1RMT3M1 and S2RMT3M1 taken from Matrix-Market website [14], except for BCSSTK38 which was taken from Tim Davis's collection [6]. These matrices together with their generic properties and the numerical results are given in Table 2. All of the assumptions are as before except that the GMRES(20) was used to solve the system of linear equations. Numerical results presented in Table 2, show that the proposed preconditioner greatly reduces the time and iterations to converge. In fact, GMRES(20) algorithm does not converge for seven of the eight problems. Whereas GMRES(20) with left preconditioning converges for all of the problems in very small number of iterations.

For the third set of the numerical experiments, we consider nine out of fourteen matrices tested in [15] that are the most difficult ones based on the numerical results in this reference. All of these matrices can be downloaded from Matrix-Market website or Tim Davis's collection [6, 14]. We also use the same stopping criterion and preconditioned iterative solver used in [15], i.e. $\|r_i\|_2 < 10^{-6}$ and GMRES(10) with right preconditioning, respectively. We first use the sparse approximate inverse preconditioner computed by the FFAPINV-NSPD algorithm with $\tau = 0.1$. Numerical results are given in Table 3. All of the other notations are as before. As the numerical results in this table show the proposed preconditioner is very effective in improving the convergence rate of the GMRES(m) iterative solver for nine out of the ten test problems. For RJAT04, GMRES(10) does not converge for preconditioned and unpreconditioned system of linear equations. Note that the preconditioner density for this matrix is very small ($\rho = 0.17$).

Next, for the selected matrices, we compare the numerical results of the proposed preconditioner with that of the AINV-A and SAINV-NS algorithms. Results of the AINV-A and SAINV-NS preconditioners were extracted from Table 2 and Table 5 in [15]. The parameter τ was chosen in such a way that the FFAPINV-NSPD preconditioner density, ρ, is always less than that of the AINV-A and SAINV-NS preconditioners. Numerical results in this table show that the proposed preconditioner usually gives better results than the AINV-A and SAINV-NS algorithms.

Table 3. Numerical results for the FFAPINV-NSPD algorithm with $\tau = 0.1$ for the third set of test problems.

			No prec.		Preconditioned system				
Matrix	n	nnz	Its	Time	ρ	P-time	It-time	T-time	P-Its
HOR131	434	4710	†	-	2.01	0.02	0.02	0.04	6
PDE900	900	4380	22	0.16	2.53	0.02	0.03	0.05	5
SHERMAN4	1104	3786	67	0.58	0.84	0.03	0.13	0.16	14
ADD20	2395	17319	168	3.16	0.35	0.25	0.05	0.30	2
SHERMAN1	1000	3750	520	3.98	1.85	0.02	0.05	0.07	5
PDE2961	2961	14585	44	1.03	2.35	0.13	0.28	0.41	11
RAJAT04	1041	9642	†	-	0.17	0.05	-	-	†
RAJAT12	1879	12926	†	-	0.28	0.19	1.28	1.47	79
RAEFSKY1	3242	293409	1071	60.89	0.10	0.31	3.47	3.78	56
RAEFSKY2	3242	293551	524	29.97	0.16	0.38	9.48	9.86	147

Table 4. Numerical results of FFAPINV-NSPD, AINV-A and SAINV-NS algorithms for the third set of test problems .

	FFAPINV-NSPD			AINV-A		SAINV-NS	
Matrix	τ	ρ	Its	ρ	Its	ρ	Its
HOR131	0.02	6.52	2	14.44	1	8.8	2
PDE900	0.01	24.83	1	62.9	1	33.61	2
SHERMAN4	0.003	18.84	2	34.77	3	23.18	3
ADD20	0.01	0.86	1	2.66	1	1.8	1
SHERMAN1	0.01	16.00	1	60.8	1	35.08	2
PDE2961	0.0046	64.00	1	151.92	1	71.73	3
RAJAT04	0.0015	6.33	8	9.33	7	6.45	10
RAJAT12	0.002	8.50	2	49.36	3	26.77	4
RAEFSKY1	0.0035	3.71	4	6.17	4	3.94	16
RAEFSKY2	0.005	5.96	5	10.35	†	7.58	9

5. Conclusion

We have presented a variant of the FFAPINV algorithm that is free from breakdown for NSPD linear system of equations. The proposed preconditioner is very effective in improving the number of iterations and total CPU time to converge. We have also compared the performance of the proposed preconditioner with that of the SAINV-NS and A-AINV algorithms. The numerical results show that our preconditioner usually give better results than the SAINV-NS and A-AINV algorithms.

6. Acknowledgement

The author is grateful to the anonymous referee for his/her comments which improved the quality of this paper.

References

1. M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. of Computational Physics, **182** (2002) 418-477.
2. M. Benzi, J. K. Cullum, and M. Tuma, and C. D. Meyer, *Robust approximate inverse preconditioning for the conjugate gradient Method,* SIAM J. Sci. Comput., **22** (2000) 1318-1332.
3. M. Benzi, M. Tuma, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., **30** (1999)305-340.
4. M. Benzi, C. D. Meyer, and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., **17** (1996) 1135-1149.
5. M. Benzi, M. Tuma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., **19** (1998) 968-994.
6. T. Davis, *University of Florida sparse matrix collection*, NA Digest, 92(1994), `http://www.cise.ufl.edu/research/sparse/matrices`.
7. S. A. Kharchenko, L. Yu. Kolotilina, A. A. Nikishin, A. Yu. Yeremin, *A robust AINV-type method for constructing sparse approximate inverse preconditioners in factored form*, Numer. Linear Algebra With Appl., **8** (2001) 165179.
8. L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditioning I. Theory*, SIAM J. Matrix Anal. Appl., **14** (1993) 45-58.
9. 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.
10. E.-J. Lee and J. Zhang, *Fatored approximate inverse preonditioners with dynamic sparsity patterns*, Tehnial Report No.488-07,Department of Computer Science,University of Kentuky, Lexington, KY, 2007.
11. J.-G. Luo, *An incomplete inverse as a preconditioner for the conjugate gradient method*, Comput. Math. Appl., **25** (1993) 7379.
12. J.-G. Luo, *A new class of decomposition for inverting asymmetric and indefinite matrices*, Comput. Math. Appl., **25** (1993) 95104.
13. J.-G. Luo, *A new class of decomposition for symmetric systems*, Mechanics Research Communications, **19** (1992) 159166.
14. Matrix Market page, `http://math.nist.gov/MatrixMarket`.
15. A. Rafiei and F. Toutounian, *New breakdown-free variant of AINV method for nonsymmetric positive definite matrices*, Jornal of Computational and Applied Mathematics, **219** (2008) 72-80.
16. Y. Saad, *Iterative Methods for Sparse linear Systems*, PWS press, New York, 1995.
17. Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., **7** (1986) 856-869.
18. H. A. van der Vorst, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., **12** (1992) 631-644.
19. J. Zhang, *A procedure for computing factored approximate inverse*, M.Sc. dissertation, Department of Computer Science, University of Kentucky, 1999.
20. J. Zhang, *A sparse approximate inverse technique for parallel preconditioning of general sparse matrices*, Appl. Math. Comput., **130** (2002) 63-85.

Davod Khojasteh Salkuyeh received his B.Sc from Sharif University of Technology, Tehran, Iran and his M.Sc from Ferdowsi University of Mashhad, Mashhad, Iran. He received his Ph.D degree under supervision of professor Faezeh Toutounian at Ferdowsi University of Mashhad in 2003. He is currently an assistant professor of Mathematics at

University of Mohaghegh Ardabili, Ardabil, Iran. His research interests are mainly iterative methods for large sparse linear systems of equations and error analysis.

Department of Mathematics, University of Mohaghegh Ardabili,
P. O. Box. 179, Ardabil, Iran.
e-mail: khojaste@uma.ac.ir