Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

BLOCK RECURSIVE COMPUTATION OF GENERALIZED INVERSES*

MARKO D. PETKOVIĆ† AND PREDRAG S. STANIMIROVIĆ*

Abstract. A fully block recursive method for computing outer generalized inverses of given square matrix is introduced. The method is applicable even in the case when some of main diagonal minors of A are singular or A is singular. Computational complexity of the method is not harder than the matrix multiplication, under the assumption that the Strassen matrix inversion algorithm is used. A partially recursive algorithm for computing various classes of generalized inverses is also developed. This method can be efficiently used for the acceleration of the known methods for computing generalized inverses.

Key words. Moore-Penrose inverse, Outer inverses, Banachiewicz-Schur form, Strassen method.

AMS subject classifications. 15A09.

1. Introduction. Following the usual notations, the set of all $m\times n$ real matrices of rank r is denoted by $\mathbb{R}_r^{m\times n}$, while the set of all real $m\times n$ matrices is denoted by $\mathbb{R}^{m\times n}$. The principal submatrix of $n\times n$ matrix A which is composed of rows and columns indexed by $1\leq k_1<k_2<\cdots<k_l\leq n$, $1\leq l\leq n$, is denoted by $A_{\{k_1,\ldots,k_l\}}$.

For any $m\times n$ real matrix the following matrix equations in X are used to define various generalized inverses of A:

$$(1)\quad AXA=A,\quad (2)\quad XAX=X,\quad (3)\quad (AX)^T=AX,\quad (4)\quad (XA)^T=XA.$$

Also, in the case $m=n$, the following two additional equations are exploited:

$$(5)\quad AX=XA\qquad (1^k)\quad A^{k+1}X=A^k,$$

where (1^k) is valid for any positive integer k satisfying $k\geq \mathrm{ind}(A)=\min\{p:\mathrm{rank}(A^{p+1})=\mathrm{rank}(A^p)\}$. The set of matrices obeying the equations represented in $\mathcal{S}$ is denoted by $A\{\mathcal{S}\}$, for arbitrary sequence $\mathcal{S}$ of the elements from $\{1,2,3,4,5,1^k\}$. Any matrix $X\in A\{\mathcal{S}\}$ is known as an $\mathcal{S}$-inverse of A and it is denoted by $A^{(\mathcal{S})}$ [4].

The matrix X satisfying equations (1) and (2) is said to be a *reflexive g-inverse of* A, whereas the matrix X satisfying only the equation (2) is called an *outer inverse*

*Received by the editors on May 2, 2012. Accepted for publication on May 26, 2013. Handling Editor: Oscar Maria Baksalary.

†University of Niš, Department of Computer Science, Faculty of Science and Mathematics, Višegradska 33, 18000 Niš, Serbia (dexterofnis@gmail.com, pecko@pmf.ni.ac.rs). Authors gratefully acknowledge support from the Research Project 174013 of the Serbian Ministry of Science.

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

of A. Subsequently, the *Moore-Penrose inverse* $X = A^{\dagger}$ *of* A satisfies the set of the equations (1), (2), (3) and (4). *The Drazin inverse* $X = A^{D}$ *of* A satisfies the equations (1^k), (2) and (5).

When the subproblems are of the same type as the original problem, the same recursive process can be carried out until the problem size is sufficiently small. This special type of Divide and Conquer (D&C) type of algorithms, is referred to as D&C recursion.

For given function $T(n)$, notations $\Theta(T(n))$ and $\mathcal{O}(T(n))$ mean the set of functions defined in the following way (see, for example [8]):

$$\Theta(T(n)) = \{f(n) \mid 0 \le c_1 T(n) \le f(n) \le c_2 T(n),\ n \ge n_0,\quad c_1, c_2, n_0 > 0\}$$
$$\mathcal{O}(T(n)) = \{f(n) \mid 0 \le f(n) \le cT(n),\ n \ge n_0,\quad c, n_0 > 0\}.$$

Denote by $\mathrm{add}(n)$, $\mathrm{mul}(n)$ and $\mathrm{inv}(n)$ complexities of matrix addition, multiplication and inversion on $n \times n$ matrices. Also denote the complexity of multiplying $m \times n$ matrix with $n \times k$ matrix by $\mathrm{mul}(m, n, k)$. Particularly, $\mathrm{mul}(n) = \mathrm{mul}(n, n, n)$.

Results concerning block recursive algorithms in linear algebra, based on the LU decomposition, can be found for example in [12, 13, 16, 20, 22]. Our paper [20] deals with the time complexity of block recursive generalized Cholesky factorization and its applications to the generalized inversion.

The Schur complement $S = (A/A_{11}) = A_{22} - A_{21}A_{11}^{-1}A_{12}$ of the block matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \in \mathbb{R}^{n\times n}, \quad A_{11} \in \mathbb{R}^{k\times k} \tag{1.1}$$

is a basic tool in computation of the inverse matrix A^{-1} [3]. The generalized Schur complement $S = A_{22} - A_{21}A_{11}^{-}A_{12}$, where A_{11}^{-} is a generalized inverse of A_{11}, plays an important role in representations of various generalized inverses A^{-} [2, 5, 6, 9, 10, 17, 18, 23, 24].

Although there are many representations of different generalized inverses in Banachievich-Schur form, its computational aspect is not well investigated so far. This problem is investigated in the present paper. Furthermore, we construct appropriate Strassen-type algorithm for generalized matrix inversion. The advantage of that algorithm is its computational complexity $\mathcal{O}(\mathrm{mul}(n))$, but the drawback is that it produces only outer inverses. For this purpose, we introduce one-step and partially block recursive Strassen-type algorithms. Although their time complexity is $\mathcal{O}(n^3)$, they are efficient and practically applicable.

2. Strassen method for efficient matrix multiplication and inversion. Let A, B be $n \times n$ real or complex matrices. The usual algorithms for computing the

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

matrix product $C = AB$ require n^3 multiplications and $n^3 - n^2$ additions ($2n^3 - n^2 = \mathcal{O}(n^3)$ floating point operations in total). In the paper [21], V. Strassen introduced an algorithm for matrix multiplication which complexity is $\mathcal{O}(n^{\log_2 7}) \approx \mathcal{O}(n^{2.807})$ (less than $\mathcal{O}(n^3)$). Some other algorithms for computing the matrix product in a time less than $\mathcal{O}(n^3)$ are known. Currently the best one is due to Coppersmith and Winograd [7] and works in time $\mathcal{O}(n^{2.376})$. An overview of various efficient matrix multiplication algorithms is presented in [11]. The authors of the paper [11] also improved some of efficient matrix multiplication algorithms in the case of small matrices.

Strassen in [21] also introduced the algorithm for finding the inverse of a given matrix A of the form (1.1), with the same complexity as the matrix multiplication.

LEMMA 2.1. [21] *Assume that A is partitioned as in* (1.1) *and*

$$X = A^{-1} = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \in \mathbb{R}^{n\times n}, \quad X_{11} \in \mathbb{R}^{k\times k}. \tag{2.1}$$

Matrices X_{11}, X_{12}, X_{21} and X_{22} are defined by the following relations:

$$\begin{array}{llll} 1. & R_1 = A_{11}^{-1} & 5. & R_5 = R_4 - A_{22} \\ 2. & R_2 = A_{21}R_1 & 6. & R_6 = R_5^{-1} \\ 3. & R_3 = R_1A_{12} & 7. & X_{12} = R_3R_6 \\ 4. & R_4 = A_{21}R_3 & 8. & X_{21} = R_6R_2 \end{array} \qquad \begin{array}{ll} 9. & R_7 = R_3X_{21} \\ 10. & X_{11} = R_1 - R_7 \\ 11. & X_{22} = -R_6. \end{array} \tag{2.2}$$

In the rest of this section, we assume that the matrix $A \in \mathbb{R}^{n\times n}$ is decomposed as in (1.1). The matrix R_5 in the relations (2.2) is equal to $R_5 = -(A_{22} - A_{21}A_{11}^{-1}A_{12}) = -S = -(A/A_{11})$. Intermediate matrices $R_1, \ldots, R_7$ are introduced to ensure a minimal number of matrix multiplications. However, these matrices require additional memory space to be stored. If we eliminate $R_1, \ldots, R_7$ from the relations (2.2) we obtain well-known explicit form of the block matrix inversion [3]:

$$X = A^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}S^{-1} \\ -S^{-1}A_{21}A_{11}^{-1} & S^{-1} \end{bmatrix}.$$

Now we state complete Strassen-type algorithm for fast matrix inversion (Algorithm 2.2).

ALGORITHM 2.2. **Strassen-based matrix inversion.**

Require: Invertible $n \times n$ matrix A which all principal submatrices are invertible.

1: If $n = 1$ then return $X = [a_{11}^{-1}]$. Else decompose matrix A as in (1.1) and continue.

2: Compute X_{11}, X_{12}, X_{21} and X_{22} using formulas (2.2), where the inverses are computed recursively and for matrix-matrix multiplication is used one of the $\Theta(n^{2+\epsilon})$ algorithms.

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

3: Return the inverse matrix $X = A^{-1}$.

The parameter k in the decomposition (1.1) can be chosen arbitrarily, but the most frequent choice is $k = \lfloor n/2 \rfloor$. Algorithm 2.2 in Step 2 recursively computes the inverse of principal submatrices and its Schur complements. Also, Algorithm 2.2 assumes that the recursion is continued down to the level 1×1. The only situation in which Algorithm 2.2 may crash is Step 1, in the case $n = 1$ and $a_{11} = 0$.

It is hard to verify in advance invertibility of both A_{11} and S in each recursive call of Algorithm 2.2. In the rest of this section we give an equivalent condition which can be directly checked on the input matrix A. Such conditions are not investigated in the papers concerning Algorithm 2.2.

Let $\beta, \gamma \subseteq \{1, 2, \ldots, n\}$ and $A \in \mathbb{R}^{n \times n}$. Denote by $A_{\beta,\gamma}$ a submatrix of A obtained by taking rows of A indexed by β and columns of A indexed by γ. Also denote by $\beta^c = \{1, 2, \ldots, n\} \setminus \beta$ and $A_\beta = A_{\beta,\beta}$. The following theorem is well-known and its statement can be found, for example in [25] (p. 112, Th. 4.8 and Th. 4.9), restated here as Proposition 2.3:

PROPOSITION 2.3. *Let $A \in \mathbb{R}^{n \times n}$ and $\beta \subseteq \{1, 2, \ldots, n\}$. If the matrix A is invertible, then*

$$\det A_{\beta^c}^{-1} = \frac{\det A_\beta}{\det A}, \quad A_{\beta^c}^{-1} = (A/A_\beta)^{-1} = (A_{\beta^c} - A_{\beta^c,\beta}\, A_\beta^{-1}\, A_{\beta,\beta^c})^{-1}.$$

The following lemmas are direct corollaries of Proposition 2.3.

LEMMA 2.4. *If both A and A_{11} are invertible, then the same holds for X_{22} and $S = (A/A_{11})$. In addition, we have $X_{22} = S^{-1}$.*

LEMMA 2.5. *If both A and A_{22} are invertible, then the same holds for X_{11} and (A/A_{22}).*

LEMMA 2.6. *If all principal submatrices $A_{\{1,\ldots,l\}}$ $(l = 1, 2, \ldots, n)$ of an invertible matrix A are invertible, then the same holds for the principal submatrices of the Schur complement $S = (A/A_{11})$.*

Proof. Consider an arbitrary principal submatrix $S_{\{1,\ldots,l\}}$. Since $A_{\{1,\ldots,k+l\}}$ is invertible, Lemma 2.4 implies that $X_{\{k+l+1,\ldots,n\}}$ is invertible. Since $X_{\{k+l+1,\ldots,n\}} = (X_{22})_{\{l+1,\ldots,n-k\}}$, according to Lemma 2.5, we directly obtain that $S_{\{1,\ldots,l\}}$ is invertible. □

LEMMA 2.7. *If both A and $S = (A/A_{11})$ are invertible and $A_{\{1,\ldots,l\}}$ is singular for $l > k$, then $S_{\{1,\ldots,l-k\}}$ is singular.*

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

Proof. If we assume that $X_{\{l+1,\ldots,n\}}$ is invertible, by applying Lemma 2.5 on the matrix X and its decomposition

$$X = \begin{bmatrix} X'_{11} & X'_{12} \\ X'_{21} & X'_{22} \end{bmatrix}, \; X'_{11} = X_{\{1,\ldots,l\}},$$

we obtain that $A'_{11} = A_{\{1,\ldots,l\}}$ is invertible, which is contradiction. Hence, $X_{\{l+1,\ldots,n\}}$ is singular. Since $X_{22} = S^{-1}$ and $(X_{22})_{\{l-k+1,\ldots,n-k\}} = X_{\{l+1,\ldots,n\}}$, from Lemma 2.4 (applied on matrices S, $S_{\{1,\ldots,l-k\}}$ and $(X_{22})_{\{l-k+1,\ldots,n-k\}}$) we conclude that $S_{\{1,\ldots,l-k\}}$ is singular. □

Now we are ready to prove the following main theorem.

THEOREM 2.8. Algorithm 2.2 *works correctly for an input matrix A (independently of the choice of the parameter k), if and only if all principal submatrices $A_{\{1,\ldots,l\}}$ $(l = 1, 2, \ldots, n)$ are invertible.*

Proof. We use the mathematical induction. For $n = 1, 2$ the statement of the theorem trivially holds. Assume that it is valid for each matrix of the size less than n and consider $A \in \mathbb{R}^{n\times n}$.

If all principal submatrices $A_{\{1,\ldots,l\}}$ $(l = 1, 2, \ldots, n)$ are invertible, according to Lemma 2.4 we have that $S = (A/A_{11})$ is invertible. Furthermore, according to Lemma 2.6, all $S_{\{1,\ldots,l\}}$ $(l = 1, 2, \ldots, n-k)$ are invertible. Now, since both A_{11} and S satisfy the conditions of the theorem, induction hypothesis yields that Algorithm 2.2 correctly computes A_{11}^{-1} and S^{-1}, and hence, it also correctly computes X.

If $A_{11} = A_{\{1,\ldots,k\}}$ is singular, then Algorithm 2.2 will fail obviously. Assume that A_{11} is invertible, but $A_{\{1,\ldots,l\}}$ is singular for some $l \neq k$. If $l < k$ then $(A_{11})_{\{1,\ldots,l\}} = A_{\{1,\ldots,l\}}$ is singular and according to the induction hypothesis, Algorithm 2.2 will fail on the computation of A_{11}^{-1}. Otherwise, if $l > k$, then according to Lemma 2.7 we have that $S_{\{1,\ldots,l-k\}}$ is singular. Again, according to induction hypothesis, Algorithm 2.2 will fail on the computation of S^{-1}. □

REMARK 2.9. If any algorithm for matrix-matrix multiplication with complexity $\mathcal{O}(n^{2+\epsilon})$ is used to perform all the matrix multiplications in relations 2, 3, 4, 7, 8 and 9 of Lemma 2.1, then Algorithm 2.2 also works with complexity $\mathcal{O}(n^{2+\epsilon})$, $0 < \epsilon < 1$. Especially, if the Strassen's matrix-matrix multiplication algorithm and full recursion is applied, Algorithm 2.2 requires $6n^{\log_2 7}/5 - n/5 = \Theta(n^{2.807})$ multiplications [8, 14, 21]. Otherwise if the usual matrix-matrix multiplication algorithm with ordinary time complexity $\mathcal{O}(n^3)$ is used, then complexity of Algorithm 2.2 is $\mathcal{O}(n^3)$.

3. Strassen-type algorithm for generalized inversion. Let us mention that some of principal submatrices $A_{\{1,\ldots,l\}}$ can be singular even for an invertible matrix A. Some authors, for example [1], agree that the stated problem cannot be encountered

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

in practice. One counterexample can be found in [23].

In this section, we assume that matrix A does not satisfy the conditions of Theorem 2.8. Moreover, we may even suppose that complete A is singular. A natural extension of Algorithm 2.2 would be to avoid the crushing in the case $a_{11} = 0$. Recall that a generalized inverse (Moore-Penrose, Drazin, or any other unique inverse) of the zero matrix $\mathbf{0}_{n\times n}$ is equal to that zero matrix itself. Based on that idea, we provide Algorithm 3.1.

ALGORITHM 3.1. **Strassen generalized matrix inversion for singular matrices.**

Require: Arbitrary $n \times n$ matrix A.

1: If $n = 1$ then return $X = [a_{11}^{(\alpha)}]$, where $a_{11}^{(\alpha)} = \begin{cases} a_{11}^{-1}, & a_{11} \neq 0 \\ 0, & a_{11} = 0. \end{cases}$
Else decompose matrix A as in (1.1) with $k = \lfloor \frac{n}{2} \rfloor$ and continue.
2: Apply formulas (2.2) where A_{11}^{-1} and R_5^{-1} are replaced by $A_{11}^{(\alpha)}$ and $R_5^{(\alpha)}$, which are computed recursively.
3: Return matrix $X = A^{(\alpha)}$ of the form (2.1).

Algorithm 3.1 returns the matrix

$$X = \begin{bmatrix} A_{11}^{(\alpha)} + A_{11}^{(\alpha)} A_{12} S^{(\alpha)} A_{21} A_{11}^{(\alpha)} & -A_{11}^{(\alpha)} A_{12} S^{(\alpha)} \\ -S^{(\alpha)} A_{21} A_{11}^{(\alpha)} & S^{(\alpha)} \end{bmatrix}, \tag{3.1}$$

which is well-known Banachiewicz-Schur form of the generalized inverse X. Here, by $S = (A/A_{11})_{(\alpha)} = A_{22} - A_{21} A_{11}^{(\alpha)} A_{12}$ we denoted the *generalized Schur complement*. Also, by $S^{(\alpha)}$ we denote the $\{\alpha\}$-generalized inverse of S.

Generalized inverses of the form (3.1) are investigated by many authors. Baksalary and Styan in [2] gave the necessary and sufficient conditions such that the outer inverses, least-squares generalized inverses and minimum norm generalized inverses can be represented in the Banachiewicz-Schur form. Similarly, Y. Wei [24] found the sufficient conditions for the Drazin inverse to be represented in the form (3.1). In the paper [9] these results are generalized to the weighted Moore-Penrose inverse and the weighted Drazin inverse of A. Burns et al. [5] investigated $\{1\}, \{2\}, \{1,3\}, \{1,4\}$ inverses as well as the Moore-Penrose inverse of A in the form (3.1). In [19] these results are extended to the set of polynomial matrices.

Some of the equations $(1), (2), (3)$ and (4) as well as the equations (1^k) and (5) for square matrices are satisfied by the matrix $X = A^{(\alpha)}$, if A and input matrices in recursive calls of Algorithm 3.1 satisfy conditions imposed in [2, 5, 19, 24]. However, this assumption is prohibitive. Only the equation (2) holds in general [5]. In this way,

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

we proved the following theorem.

THEOREM 3.2. *Let A be arbitrary $n \times n$ matrix and let $X = A^{(\alpha)}$ be the result given by* Algorithm 3.1. *The matrix X is $\{2\}$-inverse of the matrix A, i.e., holds $XAX = X$.*

4. Non-recursive and partially recursive algorithm for generalized inversion. Vast majority of matrix multiplication algorithms of complexity below $\mathcal{O}(n^3)$ are impractical for reasonable sized matrices [11]. The Strassen method for fast matrix inversion has been regarded as being fundamentally unstable [15]. Numerical experiments show that Strassen's inversion method do not always produce a small left or right residual even in the case when A is symmetric positive definite and conventional matrix multiplication is used [15] .

Moreover, Algorithm 3.1 returns only $\{2\}$ inverse in the general case, we need a different approach for computing other classes of generalized inverses. Furthermore, divide and conquer algorithms that are time efficient, often have relatively small recursion depth. Also, efficiency of these algorithms is usually improved if the recursion is stopped at relatively larger number of basic cases which can be solved non-recursively. In our case, our idea is to use equations (2.2), where A_{11}^{-1} and S^{-1} are replaced by some of known generalized inverses A_{11}^{-} and S^{-}. In such a way, we obtain Algorithm 4.1.

ALGORITHM 4.1. **Non-recursive matrix generalized inversion.**
Require: Arbitrary $n \times n$ matrix A.
1: Apply formulas (2.2) where we take particular generalized inverses A_{11}^{-} and S^{-} (computed outright by some other method) instead of A_{11}^{-1} and S^{-1}.
2: Return the matrix X of the form (2.1).

The following relations are of interest in determining the type of generalized inverse X computed by means of Algorithm 4.1. Those relations are given in various forms in [2, 3, 5, 9, 23, 24].

$$\mathcal{N}(A_{11}) \subseteq \mathcal{N}(A_{21}) \Leftrightarrow A_{21} = A_{21}A_{11}^{(1)}A_{11}, \quad \text{for every } A_{11}^{(1)} \tag{4.1}$$

$$\mathcal{N}(A_{11}^*) \subseteq \mathcal{N}(A_{12}^*) \Leftrightarrow A_{12} = A_{11}A_{11}^{(1)}A_{12}, \text{ for every } A_{11}^{(1)} \tag{4.2}$$

$$\mathcal{N}(S^*) \subseteq \mathcal{N}(A_{21}^*) \Leftrightarrow A_{21} = SS^{(1)}A_{21}, \text{ for every } S^{(1)} \tag{4.3}$$

$$\mathcal{N}(S) \subseteq \mathcal{N}(A_{12}) \Leftrightarrow A_{12} = A_{12}S^{(1)}S, \text{ for every } S^{(1)} \tag{4.4}$$

$$\mathcal{R}(A_{21}) \subseteq \mathcal{R}(A^D),\ \mathcal{N}(A^D) \subseteq \mathcal{N}(A_{21}),\ A_{12} = A_{12}S^DS,\ A_{21} = S^DSA_{21}. \tag{4.5}$$

In these relations, $\mathcal{N}(A)$ is the nullspace of A and $\mathcal{R}(A)$ is the range of A.

Following notations from [5], we use conditions $(N_1), (N_3), (N_4), (N)$ with the following meaning. Condition (N_1) is equivalent with (4.1) and (4.2), condition (N_3)

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

with (4.2) and (4.3), (N_4) assumes (4.1) and (4.4), while (N) is equivalent with (N_3) and (N_4).

THEOREM 4.2. Algorithm 4.1 *computes the following generalized inverses of A in the case when it is terminated in Step 2:*

1. *If $A_{11}^- = A_{11}^{(1)}$, $S^- = S^{(1)}$ then $X = A^{(1)}$ if A satisfies (N_1) or (N_3) or (N_4).*
2. *If $A_{11}^- = A_{11}^{(2)}$, $S^- = S^{(2)}$ then $X = A^{(2)}$.*
3. *If $A_{11}^- = A_{11}^{(j)}$, $S^- = S^{(j)}$ then $X = A^{(j)}$ if A satisfies (N_j), $j = 3, 4$.*
4. *If $A_{11}^- = A_{11}^{(1,j)}$, $S^- = S^{(1,j)}$ then $X = A^{(1,j)}$ if A satisfies (N_j), $j = 3, 4$.*
5. *If $A_{11}^- = A_{11}^{\dagger}$, $S^- = S^{\dagger}$ then $X = A^{\dagger}$ if A satisfies (N).*
6. *If $A_{11}^- = A_{11}^{D}$, $S^- = S^{D}$ and A satisfies (4.5), then $X = A^{D}$.*

Note that some practical implementations of Algorithm 2.2 do not use a full recursion. Instead, one switches to some other algorithm for matrix inversion at an appropriate block size [14]. In other words, it performs the recursion to some depth, but after that use another method to compute appropriate matrix inverses. We call such method a *partially recursive method*.

Partially recursive method (Algorithm 4.3) represent a balance between Algorithm 3.1, which is fully recursive, and Algorithm 4.1, which takes only one matrix decomposition. However, Algorithm 4.3 requires that appropriate conditions of the Theorem 4.2 are satisfied by an input matrix on each recursive call. Recursion depth d means that algorithm uses d recursive calls and after that produces selected generalized inverses A_{11}^- and S^- in order to obtain the inverse A^-.

ALGORITHM 4.3. **PartRecInv**(A, d): Partially recursive method for generalized inversion.

Require: Arbitrary $n \times n$ matrix A and recursion depth $d < n$.

1: If $d = 1$ then apply Algorithm 4.1. Otherwise continue.
2: Compute $A_{11}^- = \mathbf{PartRecInv}(A_{11}, d-1)$ and $S^- = \mathbf{PartRecInv}(S, d-1)$.
3: Apply formulas (2.2) where A_{11}^- and S^- are taken instead of A_{11}^{-1} and S^{-1}, respectively.
4: Return the matrix X of the form (2.1).

At the end of this section, note that the time complexity of both Algorithm 4.1 and Algorithm 4.3 is $\Theta(n^3)$ (under the assumption that $d \ll n$). That is true even in the case when $\Theta(n^{2+\epsilon})$ algorithm is used for matrix-matrix multiplications. However, as it will be seen in the next section, both algorithms are practically efficient.

5. Numerical examples. We implement Algorithm 4.1 in the package `Mathematica 8.0` and test it on numerous randomly generated test matrices. For this test, we compute the Moore-Penrose (MP) inverse of a test matrix A which satisfies

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

condition (N) (i.e., conditions (4.1)–(4.4)).

For the sake of simplicity, we used randomly generated test matrices satisfying (4.1)–(4.4), rather than some classes of test-matrices. To ensure that required conditions are satisfied, we used the procedure described in Algorithm 5.1.

ALGORITHM 5.1. **Creating a random matrix satisfying eq.** (4.1)–(4.4)**.**
Require: Matrix of dimension n, with dimension of the first block k.
1: Randomly generate a $k \times k$ matrix A_{11} and $(n-k) \times (n-k)$ matrix S.
2: Randomly generate $k \times (n-k)$ and $(n-k) \times k$ matrices C_1 and C_2 respectively.
3: Return the matrix $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{11}C_1S \\ SC_2A_{11} & S + SC_2A_{11}C_1S \end{bmatrix}$.

LEMMA 5.2. *Random matrix A generated by* Algorithm 5.1 *satisfies conditions* (4.1)–(4.4).

Proof. By putting expressions for A_{21} and A_{12} into (4.1)–(4.4) we obtain

$$\begin{aligned} A_{21}A_{11}^{(1)}A_{11} &= SC_2(A_{11}A_{11}^{(1)}A_{11}) = SC_2A_{11} = A_{21} \\ A_{11}A_{11}^{(1)}A_{12} &= (A_{11}A_{11}^{(1)}A_{11})C_1S = A_{11}C_1S = A_{12} \\ SS^{(1)}A_{21} &= (SS^{(1)}S)C_2A_{11} = SC_2A_{11} = A_{21} \\ A_{12}S^{(1)}S &= A_{11}C_1(SS^{(1)}S) = A_{11}C_1S = A_{12} \end{aligned}$$

Since the previous equations are satisfied for every $A_{11}^{(1)}$ and $S^{(1)}$, we conclude that A satisfies (4.1)–(4.4). □

Procedure described by Algorithm 5.1 can be generalized such that all generated input matrices in each recursion call (up to the depth d) of Algorithm 4.3 satisfy required conditions. Idea is to generate matrices A_{11} and S recursively, using the same method for $d-1$. That extension is used for generating a second set of test matrices.

We compared running times of Algorithm 4.1 versus the built-in `Mathematica` function `Pseudoinverse` which computes the Moore-Penrose inverse of a given matrix. The testing was done on Intel Core-i5 processor on 2.66 GHz (without multi-core optimizations) with 4 GB of RAM. Elements of all test matrices $A \in \mathbb{R}^{n\times n}$ are real numbers from the interval $(0, 1)$ and the value of the parameter k was $k = \lfloor n/2 \rfloor$. All running times (in seconds) are averaged from 20 generated test matrices and showed in Table 5.1.

We see that Algorithm 4.1 clearly outperforms `Mathematica` build-in function `Pseudoinverse`, although the time complexity of both implementations is $\Theta(n^3)$. Note that matrix-matrix multiplications in `Mathematica` are also $\Theta(n^3)$ operations. We can give one simplified explanation of this fact. Assume that the running time of

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

n	`Pseudoinverse`	Alg. 4.1
500	0.078	0.049
600	0.109	0.094
700	0.180	0.125
800	0.250	0.156
900	0.359	0.203
1000	0.484	0.250

$\operatorname{rank} A_{11} = \operatorname{rank} S = \lfloor k/2 \rfloor$

n	`Pseudoinverse`	Alg. 4.1
500	0.110	0.078
600	0.156	0.093
700	0.234	0.125
800	0.344	0.187
900	0.438	0.218
1000	0.578	0.266

$\operatorname{rank} A_{11} = \operatorname{rank} S = \lfloor 7k/8 \rfloor$

TABLE 5.1

Average running times of the implementation of Algorithm 4.1 *and* `Mathematica` *build-in function* `Pseudoinverse`.

the function `Pseudoinverse` on $n \times n$ matrix is equal to $\operatorname{pinv}(n) \approx C \cdot n^3$, for some constant $C > 0$. Also, assume that the matrix-matrix multiplication is performed in time $\operatorname{mul}(n) \approx c \cdot n^3$, where $c > 0$ is another constant. We may also assume that $c < C$, since the Moore-Penrose inverse computation is (computationally) more difficult job than the matrix multiplication. Under these assumptions, we estimate running time of Algorithm 4.1:

$$2 \cdot \operatorname{pinv}(n/2) + 6 \cdot \operatorname{mul}(n/2) \approx \frac{(C + 3c)}{4} \cdot n^3,$$

which is definitely less than $\operatorname{pinv}(n) \approx C \cdot n^3$. Of course, the ratio between individual running times may vary significantly, depending on the matrix dimension n, rank r and the entire structure of the input matrix as well.

We also compared running times of Algorithm 4.1, Algorithm 4.3 (for various values of the recursion depth d) and `Mathematica` build-in function `Pseudoinverse`. Note that, for this purpose, we needed test matrices A such that Algorithm 4.3 works correctly for a given value of d. Results (in seconds) obtained on different randomly generated test matrices are shown in Table 5.2.

n	`Pseudoinverse`	Alg. 4.1	Alg. 4.3 $d = 2$	Alg. 4.3 $d = 4$	Alg. 4.3 $d = 8$
600	0.166	0.097	0.070	0.054	0.098
700	0.239	0.131	0.091	0.075	0.121
800	0.333	0.177	0.122	0.091	0.141
900	0.434	0.233	0.180	0.137	0.176
1000	0.575	0.293	0.219	0.175	0.204
1100	0.760	0.373	0.284	0.216	0.247

TABLE 5.2

Average running times of Algorithms 4.1 *and* 4.3, *versus* `Mathematica` *function* `Pseudoinverse`.

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

We see that best running time comes from Algorithm 4.3 in the case $d = 4$. Performances are degraded by further increase of the depth d. In that case, the overhead coming from handling of the recursive calls becomes significant and comparable to the time spent on generalized inverses computation. Note that an optimal value of d may vary for different values of the matrix dimension n. It is also further limited by a maximal depth such that all required conditions from Theorem 4.2 are satisfied for an input matrix in each recursive call.

Finally, let us note that the application of Algorithms 4.1 and 4.3 did not destroy the accuracy of the obtained result. Residual norms $\|AXA - A\|$, $\|XAX - X\|$, $\|AX - (AX)^T\|$ and $\|XA - (XA)^T\|$ were the same order of magnitude as in the case of `PseudoInverse`.

6. Conclusion. In the present paper, we follow the main idea of the paper [20], expressed in its title: construction of algorithms for generalized inversion having the same complexity as the matrix multiplication. Guided by the same result for inverting nonsingular matrices (see for example [8], we tend to use Strassen algorithm for fast matrix multiplication and completely block recursive algorithms.

We show that if all principal submatrices of A are invertible, then the same holds for its inverse matrix X and for the Schur complements of A, and describe corresponding purely recursive algorithm for computing the usual inverse of A. We also extended this method in the case when some principal submatrices of A are singular or complete A is singular, and define a recursive algorithm for computing $\{2\}$-inverses in purely block recursive form which complexity is also $\Theta(\mathrm{mul}(n))$. Therefore, by using any method for matrix multiplication with cost $\mathcal{O}(n^{2+\epsilon})$, $0 < \epsilon < 1$, we generate algorithm for computing outer inverses of computational cost below $\mathcal{O}(n^3)$.

The lack of such an algorithm is that it generates only outer inverses. We also develop partially recursive algorithm for computing various classes of generalized inverses. This algorithm uses in advance prescribed recursive depth d. The efficiency of the algorithm depending on the value d is investigated.

Acknowledgment. The authors thank to anonymous reviewers for extensive review and valuable comments improving the quality of the paper.

REFERENCES

[1] S.K. Abdali and D.S. Wise. Experiments with quadtree representation of matrices. In: P. Gianni (editor), *Symbolic and Algebraic Computation, Lecture Notes in Computer Science*, Proceedings ISSAC 88, Springer-Verlag, Berlin, 358:96–108, 1989.

[2] J.K. Baksalary and G.P.H. Styan. Generalized inverses of partitioned matrices in Banachiewicz-Schur form. *Linear Algebra Appl.*, 354:41–47, 2002.

Electronic Journal of Linear Algebra ISSN 1081-3810
A publication of the International Linear Algebra Society
Volume 26, pp. 394-405, July 2013

ELA

[3] T. Banachiewicz. Zur berechnung der determinanten wie auch der inversen, und zur darauf basierten auflosung der systeme linearer gleichungen. *Acta Astronom. Ser. C*, 3:41–67, 1937.

[4] A. Ben-Israel and T.N.E. Greville. *Generalized Inverses: Theory and Applications*, second edition. Springer-Verlag, New York, 2003.

[5] F. Burns, D. Carlson, E. Haynsworth, and T. Markham. Generalized inverse formulas using Schur complement. *SIAM J. Appl. Math*, 26:254–259, 1974.

[6] N. Castro-González and M.F. Martinez-Serrano. Drazin inverse of partitioned matrices in terms of Banachiewicz-Schur form. *Linear Algebra Appl.*, 432:1691–1702, 2010.

[7] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progression. *J. Symbolic Comput.*, 9:251–280, 1990.

[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*, second edition. The MIT Press, Cambridge, Massachusetts-London, 2001.

[9] D.S. Cvetković and B. Zheng. Weighted generalized inverses of partitioned matrices in Banachiewicz-Schur forms. *J. Appl. Math. Comput.*, 22:175–184, 2006.

[10] C.Y. Deng. A note on the Drazin inverses with Banachiewicz-Schur forms. *Applied Math. Comput.*, 213:230–234, 2009.

[11] C.É. Drevet, Md.N. Islam, and É. Schost. Optimization techniques for small matrix multiplication. *Theoret. Comput. Sci.*, 412:2219–2236, 2011.

[12] F.G. Gustavson. Recursion leads to automatic variable blocking for dense linear algebra algorithms. *IBM J. Res. Develop.*, 41:737–755, 1997.

[13] F.G. Gustavson and I. Jonsson. Minimal-storage high-performance Cholesky factorization via blocking and recursion. *IBM J. Res. Develop.*, 44:823–850, 2000.

[14] N.J. Higham. Exploiting fast matrix multiplication within the level 3 BLAS. *ACM Trans. Math. Software*, 16:352–368, 1990.

[15] N.J. Higham. *Accuracy and Stability of Numerical Algorithms*, second edition. SIAM, Philadelphia, 2002.

[16] L. Karlsson. *Computing explicit matrix inverses by recursion.* Master's Thesis, Umeå University, Department of Computing Science, SE-901 87 Umeå, Sweden, February 15, 2006.

[17] X. Li. A representation for the Drazin inverse of block matrices with a singular generalized Schur complement. *Appl. Math. Comput.*, 217:7531–7536, 2011.

[18] T.-Tzer Lu and S.-Hua Shiou. Inverses of 2×2 block matrices. *Comput. Math. Appl.*, 43:119–129, 2002.

[19] A.R. Meenakshi and N. Anandam. Polynomial generalized inverses of a partitioned polynomial matrix. *J. Indian Math. Soc. (N.S.)*, 58:11–18, 1992.

[20] M.D. Petković and P.S. Stanimirović. Generalized matrix inversion is not harder than matrix multiplication. *J. Comput. Appl. Math.*, 230:270–282, 2009.

[21] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.* 13:354–356, 1969.

[22] S. Toledo. Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 18:1065–1081, 1997.

[23] S.M. Watt. Pivot-free block matrix inversion. *Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, IEEE Computer Society, Washington, DC, 151–155, 2006.

[24] Y. Wei. Expressions for the Drazin inverse of a 2×2 block matrix. *Linear Multilinear Algebra*, 45:131–146, 1998.

[25] F. Zhang (editor). *The Schur Complement and its Applications.* Springer-Verlag, New York, 2005.