2013

Block recursive computation of generalized inverses

Marko D. Petkovic
dexterofnis@gmail.com

Predrag S. Stanimirovic

Follow this and additional works at: http://repository.uwyo.edu/ela

Recommended Citation
DOI: https://doi.org/10.13001/1081-3810.1660
BLOCK RECURSIVE COMPUTATION
OF GENERALIZED INVERSES∗

MARKO D. PETKOVIC† AND PREDRAG S. STANIMIROVIC∗

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 $\mathbb{R}^{m \times n}_r$, 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 $A$:

\begin{align*}
(1) & \quad AXA = A, & (2) & \quad XAX = X, & (3) & \quad (AX)^T = AX, & (4) & \quad (XA)^T = XA.
\end{align*}

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

\begin{align*}
(5) & \quad AX = XA & (1^k) & \quad A^{k+1}X = A^k,
\end{align*}

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

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.
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 $O(T(n))$ mean the set of functions defined in the following way (see, for example [8]):

$$\Theta(T(n)) = \{f(n) \mid 0 \leq c_1 T(n) \leq f(n) \leq c_2 T(n), \ n \geq n_0, \ c_1, c_2, n_0 > 0\}$$

$$O(T(n)) = \{f(n) \mid 0 \leq f(n) \leq c T(n), \ n \geq n_0, \ c, n_0 > 0\}.$$}

Denote by $\text{add}(n)$, $\text{mul}(n)$ and $\text{inv}(n)$ 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 $\text{mul}(m, n, k)$. Particularly, $\text{mul}(n) = \text{mul}(n, n, n)$.

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

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

is a basic tool in computation of the inverse matrix $A^{-1}$ [3]. The generalized Schur complement $S = A_{22} - A_{21} A_{11}^{-1} A_{12}$, where $A_{11}^{-1}$ is a generalized inverse of $A_{11}$, plays an important role in representations of various generalized inverses $A^{-1}$ [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 $O(\text{mul}(n))$, 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 $O(n^3)$, 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
matrix product \( C = AB \) require \( n^3 \) multiplications and \( n^3 - n^2 \) additions \((2n^3 - n^2 = O(n^3)\) floating point operations in total). In the paper [21], V. Strassen introduced an algorithm for matrix multiplication which complexity is \( O(n^{\log_2 7}) \approx O(n^2.807) \) (less than \( O(n^3) \)). Some other algorithms for computing the matrix product in a time less than \( O(n^3) \) are known. Currently the best one is due to Coppersmith and Winograd [7] and works in time \( O(n^{2.376}) \). 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

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

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

\[
\begin{align*}
1. & \quad R_1 = A_{11}^{-1} & 5. & \quad R_5 = R_4 - A_{22} \\
2. & \quad R_2 = A_{21} R_1 & 6. & \quad R_6 = R_5^{-1} \\
3. & \quad R_3 = R_1 A_{12} & 7. & \quad X_{12} = R_3 R_6 \\
4. & \quad R_4 = A_{21} R_3 & 8. & \quad X_{21} = R_6 R_2 \\
9. & \quad R_7 = R_3 X_{21} & 10. & \quad X_{11} = R_1 - R_7 \\
11. & \quad X_{22} = -R_6.
\end{align*}
\]

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]:

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

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 \( X = [a_{i1}^{-1}] \). 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.
3: Return the inverse matrix \( X = A^{-1} \).

The parameter \( k \) in the decomposition 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 \( 1 \times 1 \). 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 \( \beta \) and columns of \( A \) indexed by \( \gamma \). 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 2.3 implies that \( X_{\{k+l+1,\ldots,n\}} \) is invertible. Since \( X_{\{k+l+1,\ldots,n\}} = (X_{22})_{\{1,\ldots,n-k\}} \), 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.
Proof. If we assume that $X_{(l+1,...,n)}$ is invertible, by applying Lemma 2.5 on the matrix $X$ and its decomposition

$$X = \begin{bmatrix} X'_{11} & X'_{12} \\ X'_{21} & X'_{22} \end{bmatrix}, \quad X'_{11} = X_{[1,...,l]},$$

we obtain that $A'_{11} = A_{[1,...,l]}$ is invertible, which is contradiction. Hence, $X_{(l+1,...,n)}$ is singular. Since $X'_{22} = S^{-1}$ and $(X_{22})_{(l-k+1,...,n-k)} = X_{(l+1,...,n)}$, from Lemma 2.4 (applied on matrices $S$, $S_{[1,...,l-k]}$ and $(X_{22})_{(l-k+1,...,n-k)}$) we conclude that $S_{[1,...,l-k]}$ 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 $A_{[1,...,l]}$ ($l = 1, 2, \ldots, n$) 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 $A_{[1,...,l]}$ ($l = 1, 2, \ldots, n$) are invertible, according to Lemma 2.4 we have that $S = (A/A_{11})$ is invertible. Furthermore, according to Lemma 2.3 all $S_{[1,...,l]}$ ($l = 1, 2, \ldots, n-k$) 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 $A_{11} = A_{[1,...,k]}$ is singular, then Algorithm 2.2 will fail obviously. Assume that $A_{11}$ is invertible, but $A_{[1,...,l]}$ is singular for some $l \neq k$. If $l < k$ then $(A_{11})_{[1,...,l]} = A_{[1,...,l]}$ 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 $S_{[1,...,l-k]}$ 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 $O(n^{2+\varepsilon})$ 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 $O(n^{2+\varepsilon})$, $0 < \varepsilon < 1$. 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, 13, 21]. Otherwise if the usual matrix-matrix multiplication algorithm with ordinary time complexity $O(n^3)$ is used, then complexity of Algorithm 2.2 is $O(n^3)$.

**3. Strassen-type algorithm for generalized inversion.** Let us mention that some of principal submatrices $A_{[1,...,l]}$ can be singular even for an invertible matrix $A$. Some authors, for example [1], agree that the stated problem cannot be encountered
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 $0_{n \times n}$ 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 $R_s^{-1}$ are replaced by $A_{11}^{(\alpha)}$ and $R_s^{(\alpha)}$, which are computed recursively.

3. Return matrix $X = A^{(\alpha)}$ of the form (2.1).

Algorithm 3.1 returns the matrix

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

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. Bak-
salary 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 [3] these results are generalized to the weighted Moore-Penrose inverse and the weighted Drazin inverse of $A$. Burns et al. [4] 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 (1k) 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,
we proved the following theorem.

**Theorem 3.2.** Let $A$ be arbitrary $n \times n$ matrix and let $X = A^{(s)}$ 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 $O(n^3)$ 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 $A^{-1}_{11}$ and $S^{-1}$ are replaced by some of known generalized inverses $A^{-1}_{11}$ and $S^{-1}$. 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 $A^{-1}_{11}$ and $S^{-1}$ (computed outright by some other method) instead of $A^{-1}_{11}$ 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].

(4.1) $\mathcal{N}(A_{11}) \subseteq \mathcal{N}(A_{21}) \iff A_{21} = A_{21}A_{11}^{-1}A_{11}$, for every $A_{11}$
(4.2) $\mathcal{N}(A_{11}) \subseteq \mathcal{N}(A_{12}) \iff A_{12} = A_{12}A_{11}^{-1}A_{12}$, for every $A_{11}$
(4.3) $\mathcal{N}(S^*) \subseteq \mathcal{N}(A_{21}) \iff A_{21} = SS^{-1}A_{21}$, for every $S^{-1}$
(4.4) $\mathcal{N}(S) \subseteq \mathcal{N}(A_{12}) \iff A_{12} = A_{12}S^{-1}S$, for every $S^{-1}$
(4.5) $\mathcal{R}(A_{21}) \subseteq \mathcal{R}(A^{D})$, $\mathcal{N}(A^{D}) \subseteq \mathcal{N}(A_{21})$, $A_{12} = A_{12}S^{D}S$, $A_{21} = S^{D}SA_{21}$.

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

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 5.1 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 $A_{-11} = \text{PartRecInv}(A_{11}, d - 1)$ and $S^- = \text{PartRecInv}(S, d - 1)$.
3. Apply formulas (4.2), 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
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

\[
A_{21}A_{11} = SC_2(A_{11}A_{11}^{(1)}A_{11}) = SC_2A_{11} = A_{21}
\]
\[
A_{11}A_{11}^{(1)}A_{12} = (A_{11}A_{11}^{(1)}A_{11})C_1S = A_{11}C_1S = A_{12}
\]
\[
SS^{(1)}A_{21} = (SS^{(1)}S)C_2A_{11} = SC_2A_{11} = A_{21}
\]
\[
A_{12}S^{(1)}S = A_{11}C_1(SS^{(1)}S) = A_{11}C_1S = A_{12}
\]

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 \texttt{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 \texttt{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
the function \texttt{Pseudoinverse} on $n \times n$ matrix is equal to \texttt{pinv}(n) \approx C \cdot n^3$, for some constant $C > 0$. Also, assume that the matrix-matrix multiplication is performed in time \texttt{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:

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

which is definitely less than \texttt{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 \texttt{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.

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

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

**Table 5.1**

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

<table>
<thead>
<tr>
<th>$n$</th>
<th>\texttt{Pseudoinverse}</th>
<th>Alg. 4.1</th>
<th>Alg. 4.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.078</td>
<td>0.049</td>
<td>0.110</td>
</tr>
<tr>
<td>600</td>
<td>0.109</td>
<td>0.094</td>
<td>0.156</td>
</tr>
<tr>
<td>700</td>
<td>0.180</td>
<td>0.125</td>
<td>0.234</td>
</tr>
<tr>
<td>800</td>
<td>0.250</td>
<td>0.156</td>
<td>0.344</td>
</tr>
<tr>
<td>900</td>
<td>0.359</td>
<td>0.203</td>
<td>0.438</td>
</tr>
<tr>
<td>1000</td>
<td>0.484</td>
<td>0.250</td>
<td>0.578</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$d$</th>
<th>Alg. 4.1</th>
<th>Alg. 4.3</th>
<th>Alg. 4.3</th>
<th>Alg. 4.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>0.166</td>
<td>0.097</td>
<td>0.070</td>
<td>0.054</td>
</tr>
<tr>
<td>700</td>
<td>0.239</td>
<td>0.131</td>
<td>0.091</td>
<td>0.075</td>
</tr>
<tr>
<td>800</td>
<td>0.333</td>
<td>0.177</td>
<td>0.122</td>
<td>0.091</td>
</tr>
<tr>
<td>900</td>
<td>0.434</td>
<td>0.233</td>
<td>0.180</td>
<td>0.137</td>
</tr>
<tr>
<td>1000</td>
<td>0.575</td>
<td>0.293</td>
<td>0.219</td>
<td>0.175</td>
</tr>
<tr>
<td>1100</td>
<td>0.760</td>
<td>0.373</td>
<td>0.284</td>
<td>0.216</td>
</tr>
</tbody>
</table>

**Table 5.2**

Average running times of Algorithms 4.1 and 4.3 versus Mathematica function Pseudoinverse.
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 \( \Theta(\text{mul}(n)) \). Therefore, by using any method for matrix multiplication with cost \( O(n^{2+\epsilon}) \), \( 0 < \epsilon < 1 \), we generate algorithm for computing outer inverses of computational cost below \( O(n^3) \).

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

Block Recursive Computation of Generalized Inverses


