Journal of Computational and Applied Mathematics 230 (2009) 270–282

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

Generalized matrix inversion is not harder than matrix multiplication

Marko D. Petković [*,1], Predrag S. Stanimirović [1]

University of Niš, Department of Mathematics, Faculty of Science, Višegradska 33, 18000 Niš, Serbia

ARTICLE INFO

Article history:
Received 6 February 2008
Received in revised form 12 November 2008

MSC:
15A09

Keywords:
Cholesky factorization
Complexity analysis
Generalized inverses
Moore–Penrose inverse
Strassen method

ABSTRACT

Starting from the Strassen method for rapid matrix multiplication and inversion as well as from the recursive Cholesky factorization algorithm, we introduced a completely block recursive algorithm for generalized Cholesky factorization of a given symmetric, positive semi-definite matrix $A \in \mathbb{R}^{n\times n}$. We used the Strassen method for matrix inversion together with the recursive generalized Cholesky factorization method, and established an algorithm for computing generalized $\{2, 3\}$ and $\{2, 4\}$ inverses. Introduced algorithms are not harder than the matrix–matrix multiplication.

1. Introduction

The set of all $m \times n$ real matrices of rank r is denoted by $\mathbb{R}_r^{m\times n}$. By $A_{(k_1,\ldots,k_l)}$ we denote the main diagonal minor of $n \times n$ matrix A corresponding to rows and columns indexed by the indices $1 \le k_1 < k_2 < \cdots < k_l \le n$.

For any matrix $A \in \mathbb{R}^{m\times n}$ consider the following equations in G:

$$(1)\ AGA = A, \qquad (2)\ GAG = G, \qquad (3)\ (AG)^{\mathrm{T}} = AG, \qquad (4)\ (GA)^{\mathrm{T}} = GA$$

where the superscript T denotes transpose matrix.

For a sequence $\mathcal{S}$ of elements from the set $\{1, 2, 3, 4\}$, the set of matrices obeying the equations represented in $\mathcal{S}$ is denoted by $A\{\mathcal{S}\}$. A matrix from $A\{\mathcal{S}\}$ is called an $\mathcal{S}$-inverse of A and denoted by $A^{(\mathcal{S})}$. Subsequently, the Moore–Penrose inverse $G = A^{\dagger}$ of A is unique and satisfies the set of the equations (1)–(4).

The sets of $\{2, 3\}$, $\{2, 4\}$ inverses of rank s, $0 < s < r = \operatorname{rank}(A)$ is denoted by $A\{2, 3\}_s$ and $A\{2, 4\}_s$, as in [1], and defined in the following way:

$$A\{2,3\}_s = \{X \mid XAX = X,\ (AX)^* = AX,\ \operatorname{rank}(X) = s\},$$
$$A\{2,4\}_s = \{X \mid XAX = X,\ (XA)^* = XA,\ \operatorname{rank}(X) = s\}.$$

Our starting motivation in the present paper is the following Theorem 28.8 from [2]: **matrix inversion is no harder than matrix multiplication**. The theorem is stated under the assumptions that we can multiply two $n \times n$ real matrices in time $\operatorname{mul}(n) = \Omega(n^2)$, where $\operatorname{mul}(n)$ satisfies the following two regularity conditions: $\operatorname{mul}(n + k) = \mathcal{O}(\operatorname{mul}(n))$ for any k in the range $0 \le k \le n$ and $\operatorname{mul}(n/2) \le c \cdot \operatorname{mul}(n)$ for some constant $c < 1$. Then the ordinary inverse of any real nonsingular

[*] Corresponding author.
E-mail addresses: dexterofnis@gmail.com (M.D. Petković), pecko@pmf.ni.ac.yu (P.S. Stanimirović).

[1] The authors gratefully acknowledge support from the research project 144011 of the Serbian Ministry of Science.

$n \times n$ matrix can be computed in time $\mathcal{O}(\text{mul}(n))$. Definitions of $\Theta(f(n))$, $\Omega(f(n))$ and $\mathcal{O}(f(n))$ can be found, for example, in [2].

Let A, B be $n \times n$ real or complex matrices. The number of scalar operations required for computing the matrix product $C = AB$ by the ordinary method is $2n^3 - n^2 = \mathcal{O}(n^3)$ (n^3 multiplications and $n^3 - n^2$ additions). In the paper [3], V. Strassen introduced an algorithm for matrix multiplication which complexity is $\mathcal{O}(n^{\log_2 7}) \approx n^{2.807}$ (less than $\Theta(n^3)$). There are other algorithms for computing the product $C = AB$ in time below $\Theta(n^3)$. Currently the best one is due to Coppersmith and Winograd [4] and it works in time $\mathcal{O}(n^{2.376})$.

Strassen in [3] introduced the algorithm for finding the inverse of given $n \times n$ matrix A with the same complexity as the matrix multiplication. This algorithm is based on the block decomposition of the matrix A and analoguous decomposition of its ordinary inverse.

Lemma 1.1 *([3]). If A is given $n \times n$ matrix partitioned in the following way*

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

and both A and A_{11} are regular, then the inverse matrix $X = A^{-1}$ can be represented in well known form of the block matrix inversion [5]*:*

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1} A_{12} S^{-1} A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} S^{-1} \\ -S^{-1} A_{21} A_{11}^{-1} & S^{-1} \end{bmatrix}, \tag{1.2}$$

where $S = A_{22} - A_{21} A_{11}^{-1} A_{12} = (A/A_{11})$ is the Schur complement of A_{11} in the matrix A.

The number of matrix multiplications required in (1.2) to compute blocks X_{11}, X_{12}, X_{21} and X_{22} in the block form (1.2) can be decreased below $\Theta(n^3)$ using the temporary matrices $R_1, \ldots, R_7$ and the following relations [3]*:*

$$\begin{array}{ll} 1.\ R_1 = A_{11}^{-1} & \\ 2.\ R_2 = A_{21} R_1 & 7.\ X_{12} = R_3 R_6 \\ 3.\ R_3 = R_1 A_{12} & 8.\ X_{21} = R_6 R_2 \\ 4.\ R_4 = A_{21} R_3 & 9.\ R_7, = R_3 X_{21} \\ 5.\ R_5 = R_4 - A_{22} & 10.\ X_{11} = R_1 - R_7 \\ 6.\ R_6 = R_5^{-1} & 11.\ X_{22} = -R_6. \end{array} \tag{1.3}$$

Let us notice that the matrix R_5 in the relations (1.3) is equal to the minus Schur complement of A_{11} in the matrix A, i.e. $R_5 = -(A/A_{11})$.

Formulas (1.2) and (1.3) are applicable if both A_{11} and the Schur complement $S = (A/A_{11})$ are invertible.

Our main intention in the present paper is **development of an algorithm for rapid computation of** $\{2, 3\}$ **and** $\{2, 4\}$ **generalized inverses, with complexity which is not greater than the matrix multiplication complexity**.

Representations of $\{2, 3\}$ and $\{2, 4\}$ inverses are established in [6] and they are based on the generalized Cholesky decomposition defined in [7] and the usual matrix inversion. Therefore, we are caused to use Strassen algorithm for matrix inversion and develop algorithm which computes the generalized Cholesky factorization in the matrix multiplication complexity.

In order to accomplish our idea, we organized the paper as in the following.

In the second section we state a recursive algorithm for rapid matrix inversion, not harder than the matrix multiplication.

A new Strassen-type full recursive algorithm for simultaneous fast computation of the Cholesky factorization matrix U satisfying $A = U^T U$, and its inverse Y is introduced in Section 3. The algorithm is applicable to symmetric positive-definite matrix. A generalization of this algorithm to positive semi-definite matrices gives analogous recursive algorithm for the generalized Cholesky decomposition from [7]. Then the matrix Y becomes $\{1, 2, 3\}$ inverse of U.

In the fourth section we combine representations from [6] with effective generalized Cholesky decomposition, and developed algorithms for computing the Moore–Penrose and various classes of $\{2, 3\}$ and $\{2, 4\}$ generalized inverses. These algorithms are not harder than the matrix multiplication.

Algorithms are implemented in the package MATHEMATICA and numerical examples are presented.

2. Strassen matrix inversion method

Formulas (1.3) can be used for recursive computation of the matrix inverse A^{-1}. Relations 1. and 6. in (1.3) use matrix inverses of matrices with smaller dimensions ($k \times k$ and $(n - k) \times (n - k)$ respectively). By applying the same formulas recursively on these submatrices, it is obtained the recursive method for matrix inversion. Recursion can be continued down to the case of 1×1 matrices.

The original Strassen matrix inversion algorithm is based on the following two principles:

P1. in steps 1. and 6. recursively compute the inverses of smaller dimension matrices, and recursion is continued down to the level 1×1;

P2. use the Strassen's matrix–matrix multiplication method to perform all the matrix multiplications (steps 2, 3, 4, 7, 8 and 9).

Now we will state a Strassen-type algorithm for matrix inversion, based on the principle P1. Any correct method for matrix multiplication can be used. The matrix multiplication method used determines complexity of the algorithm.

Algorithm 2.1 (*Strassen-Based Matrix Inversion*).

```
Input:   Regular $n \times n$ matrix $A$ which all main diagonal minors are regular.
Step 1.  If $n = 1$ then return $X = [a_{11}^{-1}]$. Else decompose matrix $A$ with $k = \lfloor \frac{n}{2} \rfloor$ as in (1.1) and continue.
Step 2.  Apply formulas (1.3), where the inverses are computed fully recursively according to the principle P1.
Step 3.  Return the inverse matrix $X = A^{-1}$ as in (1.2).
```

Denote by $\text{inv}(n)$ the complexity of Algorithm 2.1. Also denote by $\text{add}(n)$ the complexity of the matrix addition on $n \times n$ matrices and by $\text{mul}(m, n, k)$ the complexity of multiplying $m \times n$ matrix with $n \times k$ matrix, and let $\text{mul}(n) = \text{mul}(n, n, n)$. Moreover denote by $\text{invs}(n)$, $\text{adds}(n)$ and $\text{muls}(m, n, k)$ corresponding storage complexities of Algorithm 2.1, matrix addition on $n \times n$ matrices and matrix multiplication of $m \times n$ with $n \times k$ matrix, and let $\text{muls}(n) = \text{muls}(n, n, n)$.

Remark 2.1. If any algorithm for matrix–matrix multiplication with complexity $\mathcal{O}(n^{2+\epsilon})$ is used, then Algorithm 2.1 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 2.1 requires

$$\frac{6}{5} n^{\log_2 7} - \frac{1}{5} n \approx n^{2.807}$$

multiplications [8,9,3]. Otherwise if the usual matrix–matrix multiplication algorithm with ordinary time complexity $\mathcal{O}(n^3)$ is used, then complexity of Algorithm 2.1 is $\mathcal{O}(n^3)$.

Proposition 2.1. *Storage complexity of Algorithm 2.1 is $\Theta(n^2)$.*

Proof. Note that the storage complexity of the usual matrix–matrix multiplication algorithm, as well as known methods for matrix multiplication with complexity $\text{mul}(n) = \mathcal{O}(n^{2+\epsilon})$ is equal to $\Theta(n^2)$.

Therefore, the storage complexity of Algorithm 2.1 is determined by the following recurrence formula

$$\text{invs}(n) = \text{invs}(n/2) + \text{muls}(n/2) + \Theta(n^2) = \text{invs}(n/2) + \Theta(n^2).$$

Its solution is determined by the case 3 of the Master theorem (see for example [2]) and it is equal to $\text{invs}(n) = \Theta(n^2)$. □

3. Recursive Cholesky factorization

It is well known that for a symmetric positive definite matrix A there exists an upper triangular matrix U such that holds $A = U^{\mathrm{T}}U$. This is well-known Cholesky factorization of matrix A. P. Courrieu in the paper [7] introduced the generalization of the usual Cholesky factorization. This generalization is applicable to both singular and regular matrices. The following theorem, proved in [7], guarantees its existence:

Theorem 3.1 ([7]). *Let A be a symmetric, possibly singular, positive semi-definite matrix of the order $n \times n$. Then there is an upper triangular matrix $U = [u_{ij}]$ such that $U^{\mathrm{T}}U = A$ and $u_{ii} \geq 0$ for all $i = 1, \ldots, n$. If for an index i one has $u_{ii} = 0$, then $u_{ij} = 0$ for all $j = 1, \ldots, n$. Moreover, the matrix U with these properties is unique.*

In this section we consider a recursive algorithm for computing the Cholesky factorization of both singular and regular matrices in complete block form which complexity is $\Theta(\text{mul}(n))$.

Consider again block representation (1.1) of the matrix A and an appropriate block decomposition of the matrix U:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{12}^{\mathrm{T}} & A_{22} \end{bmatrix}, \qquad U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}, \qquad U_{11}, A_{11} \in \mathbb{R}^{k \times k}. \tag{3.1}$$

Equation $A = U^{\mathrm{T}}U$ is equivalent with the following system of matrix equations:

$$\begin{aligned} &1.\ A_{11} = U_{11}^{\mathrm{T}} U_{11} \\ &2.\ A_{12} = U_{11}^{\mathrm{T}} U_{12} \\ &3.\ A_{22} = U_{12}^{\mathrm{T}} U_{12} + U_{22}^{\mathrm{T}} U_{22}. \end{aligned} \tag{3.2}$$

3.1. Regular case

Suppose first that symmetric matrix A is invertible and positive definite. Gustavson and Jonsson [10] presented a Cholesky factorization routine by combining recursion and blocking. Other results concerning recursive algorithms in linear algebra can be found for example in [11,10,12,16,17]. In the recursive algorithm from [10], the Cholesky factorization of a positive definite symmetric $n \times n$ matrix A is initiated by a recursive Cholesky factorization of the upper left square matrix A_{11} of the order $n_1 = \lfloor n/2 \rfloor$. Then the upper right matrix A_{12} can be transformed into U_{12} by the multiple solving of $n_2 = \lceil n/2 \rceil$ triangular systems of equations (each of size $n_1 = \lfloor n/2 \rfloor$) according to the second equation in (3.2). Finally, the matrix $\tilde{A}_{22} = A_{22} - U_{12}^{\mathrm{T}} U_{12}$ is recursively factored.

The complexity of solving $n \times n$ linear triangular system by Gaussian elimination is $\mathcal{O}(n^2)$. Therefore total complexity required for solving $n/2$ triangular $n \times n$ systems is $\mathcal{O}(n^3)$. For this purpose we propose an alternative method to generate block U_{12}. From the second equation in (3.2) we have that $U_{12} = (U_{11}^{\mathrm{T}})^{-1} A_{12}$. The regularity of matrix U_{11} (also U_{11}^{T}) comes from the positive-definity of matrix A. It is clear that the multiple solving of $n_2 = \lceil n/2 \rceil$ triangular systems of equations, contained in the second equation in (3.2), is equivalent with computation of the matrix expression $(U_{11}^{\mathrm{T}})^{-1} A_{12}$. Due to this rationale, we propose an algorithm for solving (3.2) which is based on the complete recursion and computes simultaneously both the matrix U and its inverse matrix Y. Moreover, this approach will be useful in computation of the generalized Cholesky factorization.

Consider the same block decomposition of the matrix $Y = U^{-1}$ as for the matrix U. We have that the following block matrix equation is satisfied

$$\begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \begin{bmatrix} Y_{11} & Y_{12} \\ 0 & Y_{22} \end{bmatrix} = \begin{bmatrix} U_{11}Y_{11} & U_{11}Y_{12} + U_{12}Y_{22} \\ 0 & U_{22}Y_{22} \end{bmatrix} = \begin{bmatrix} I_k & 0 \\ 0 & I_{n-k} \end{bmatrix}, \tag{3.3}$$

which is equivalent to the following set of equations:

$$Y_{11} = U_{11}^{-1}, \qquad Y_{22} = U_{22}^{-1}, \qquad Y_{12} = -Y_{11} U_{12} Y_{22}. \tag{3.4}$$

Combining relations (3.2) and (3.4), we can recursively compute both the Cholesky factorization matrix U satisfying $A = U^{\mathrm{T}} U$ and its inverse $Y = U^{-1}$.

Algorithm 3.1 (*Full Recursive Cholesky Factorization*).

```
Input:   Regular, symmetric, positive definite $n \times n$ matrix $A$.
Step 1.  If $n = 1$ then return $U = [\sqrt{a_{11}}]$, $Y = \left[\sqrt{a_{11}^{-1}}\right]$. Else decompose matrix $A$ as in (3.1) with $k = \lfloor \frac{n}{2} \rfloor$ and continue.
Step 2.  Compute recursively the Cholesky factorization matrix $U_{11}$ and its inverse $Y_{11}$ using the same algorithm for the
         input matrix $A_{11}$.
Step 3.  Find $U_{12} = Y_{11}^{\mathrm{T}} A_{12}$, $T_1 = U_{12}^{\mathrm{T}} U_{12}$ and $T_2 = A_{22} - T_1$.
Step 4.  Compute recursively the Cholesky factorization matrix $U_{22}$ and its inverse $Y_{22}$ using the same algorithm for the
         input matrix $T_2$.
Step 5.  Find $T_3 = -Y_{11} U_{12}$, $Y_{12} = T_3 Y_{22}$.
Step 6.  Return $U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$ and $Y = \begin{bmatrix} Y_{11} & Y_{12} \\ 0 & Y_{22} \end{bmatrix}$.
```

Correctness of Algorithm 3.1 can be easily verified:

Proposition 3.1. *The output matrices U and Y from Algorithm 3.1 satisfy $A = U^{\mathrm{T}} U$ and $Y = U^{-1}$.*

Let us compute the complexity of Algorithm 3.1 and its storage complexity (denoted by $\mathrm{Chol}(n)$ and $\mathrm{Chols}(n)$, respectively). The next theorem states that by using Strassen matrix multiplication method (or any matrix multiplication method with complexity $\mathcal{O}(n^{2+\epsilon})$ where $0 < \epsilon < 1$), we obtain complexity $\mathrm{Chol}(n)$ which is less than the complexity of the pivoting method ($\mathcal{O}(n^3)$).

Theorem 3.2. *Under the assumption* $\mathrm{mul}(n) = \Theta(n^{2+\epsilon})$, *where* $0 < \epsilon < 1$, *the complexity of Algorithm 3.1 is equal to*

$$\mathrm{Chol}(n) = \Theta(\mathrm{mul}(n)) = \Theta(n^{2+\epsilon}). \tag{3.5}$$

Moreover, its storage complexity $\mathrm{Chols}(n)$ *is equal to* $\mathrm{Chols}(n) = \Theta(n^2)$.

Proof. If we choose $k = \lfloor \frac{n}{2} \rfloor$, from relations (1.3) we have the following expression for $\mathrm{Chol}(n)$, where $l = \lceil \frac{n}{2} \rceil$:

$$\mathrm{Chol}(n) = \begin{cases} 1, & n = 1 \\ \mathrm{Chol}(k) + \mathrm{Chol}(l) + \mathrm{mul}(k,k,l) + \mathrm{mul}(l,k,l) + \mathrm{mul}(k,k,l) + \mathrm{mul}(k,l,l) + \mathrm{add}(l), & n > 1. \end{cases}$$

Assume now that n is an exact power of 2. Then since $k = l = n/2$ and $\mathrm{add}(n) = \mathcal{O}(n^2) < \mathrm{mul}(n)$, for $n > 1$ we have

$$\begin{aligned}\mathrm{Chol}(n) &= \mathrm{Chol}(n/2) + \mathrm{Chol}(n/2) + 4 \cdot \mathrm{mul}(n/2)\\ &= 2\mathrm{Chol}(n/2) + \Theta(\mathrm{mul}(n)).\end{aligned} \tag{3.6}$$

Since $\mathrm{mul}(n) = \Theta(n^{2+\epsilon})$, $0 < \epsilon < 1$, it is not difficult to verify inequality $2 \cdot \mathrm{mul}(n/2) < c \cdot \mathrm{mul}(n)$, for some constant $1 > c > 1/2$ and sufficiently large n. Therefore, by applying case 3 of the Master theorem, we obtain the solution of the recurrence relation (3.6) in the form $\mathrm{Chol}(n) = \Theta(\mathrm{mul}(n))$ and prove (3.5).

Otherwise let n is not an exact power of 2. If $U^{\mathrm{T}}U = A$ holds, for some q such that $n + q$ is the least exact power of 2 we have

$$\begin{bmatrix} A & 0 \\ 0 & I_q \end{bmatrix} = \begin{bmatrix} U & 0 \\ 0 & I_q \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} U & 0 \\ 0 & I_q \end{bmatrix}. \tag{3.7}$$

Thus we enlarge the matrix A to a size that is the closest power of 2 with respect to n and obtain the desired complexity $\mathrm{Chol}(n) = \Theta(n^{2+\epsilon})$ from the complexity $\Theta(\mathrm{mul}(n + q))$ of enlarged problem. It is not difficult to verify that holds $\mathrm{mul}(n+q) = \mathcal{O}(\mathrm{mul}(n))$ since $\mathrm{mul}(n) = \Theta(n^{2+\epsilon})$. This relation guarantees that enlargement does not changes the running time $\mathrm{Chol}(n + q)$ for more than a constant factor with respect to $\mathrm{Chol}(n)$.

Now consider the storage complexity. When n is exact power of 2, it satisfies the recurrence relation

$$\mathrm{Chols}(n) = \mathrm{Chols}(n/2) + \mathrm{muls}(n/2) + \Theta(n^2) = \mathrm{Chols}(n/2) + \Theta(n^2).$$

Similarly to the previous case, by applying case 3 of the Master theorem we can conclude that $\mathrm{Chols}(n) = \Theta(n^2)$. Otherwise, if n is not exact power of 2, we have to enlarge matrices A and U as in (3.7) and obtain $\mathrm{Chols}(n) = \mathrm{Chols}(n + q) = \Theta((n + q)^2) = \Theta(n^2)$. This completes the proof. □

3.2. Singular case

Now we will extend Algorithm 3.1 in the case when input $n \times n$ matrix A is singular or positive semi-definite. The only one point where Algorithm 3.1 might crash is Step 1 when $n = 1$ and $a_{11} = 0$. We will modify this step by using $U = Y = [0]$ in the case $n = 1$ and $a_{11} = 0$. Generalization of Step 1 in Algorithm 3.1 we denote by Step 1′.

Algorithm supervened after the replacement of Step 1 by Step 1′ we denote by Algorithm 3.1′. The input matrix of Algorithm 3.1′ is symmetric positive definite or positive semi-definite $n \times n$ matrix A.

Algorithm 3.1′ (*Full Recursive Generalized Cholesky Factorization*).

Input: Symmetric positive semi-definite $n \times n$ matrix A.
Step 1′. If $n = 1$ then return

$$U = \begin{cases} [\sqrt{a_{11}}], & a_{11} \neq 0 \\ [0], & a_{11} = 0, \end{cases} \qquad Y = \begin{cases} \left[\sqrt{a_{11}^{-1}}\right], & a_{11} \neq 0 \\ [0], & a_{11} = 0. \end{cases}$$

In the case $n > 1$ decompose matrix A as in (3.1) with $k = \lfloor \frac{n}{2} \rfloor$ and continue.

Other steps are the same as in Algorithm 3.1.

Note that complexity and storage complexity of Algorithm 3.1′ are the same as for the Algorithm 3.1.

Algorithm 3.1′ will compute matrix U of the generalized Cholesky decomposition defined in [7]. Also, output matrix Y is $\{1, 2, 3\}$ inverse for U. This is proved by the following theorem:

Theorem 3.3. *Consider a positive semi-definite (possibly singular) matrix $A \in \mathbb{R}^{n\times n}$ and output matrices $U = [u_{ij}]_{1\le i,j\le n}$ and $Y = [y_{ij}]_{1\le i,j\le n}$ from Algorithm 3.1′. Then the matrix U is generalized Cholesky decomposition matrix of the matrix A from Theorem 3.1 ([7]). If $u_{ii} = 0$ for some $i = 1, \ldots, n$ then $u_{ij} = y_{ji} = 0$ for every $j = 1, \ldots, n$. Moreover the output matrix Y is $\{1, 2, 3\}$ inverse of the matrix U, matrix UY is diagonal and all its entries on main diagonal are equal to* 0 *or* 1.

Proof. First observe that every main diagonal minor $A_{(S)}$, $S \subset \{1, \ldots, n\}$ of the positive-semidefinite matrix A is also positive-semidefinite. Denote with $S^C = \{1, \ldots, n\} \setminus S$ the complement set of indices S. Let $x' \in \mathbb{R}^{|S|}$ be arbitrary vector of length $|S|$ which is equal to the cardinality of the set S. Then by setting $x_{(S)} = x'$ and $x_{(S^C)} = 0$ we obtain $0 \le x^{\mathrm{T}}Ax = x'^{T}A_{(S)}x'$. This proves that $A_{(S)}$ is positive semi-definite.

Now we will prove the theorem by using mathematical induction.

For matrices of type 1×1, it trivially holds from Step 1′.

Assuming that the statement holds for every positive semi-definite matrix with lower dimensions, we prove the inductive step.

To verify $A = U^{\mathrm{T}}U$ it is sufficient to prove three equations in (3.2).

Since A is positive semi-definite, according to the Theorem 3.1 there exists a matrix U' such that $U'^T U' = A$. Let us partition the matrix U' in the same way as matrix U in relation (3.1): $U' = \begin{bmatrix} U'_{11} & U'_{12} \\ 0 & U'_{22} \end{bmatrix}$. Then the matrix U' satisfies equations in (3.2):

$$\begin{aligned} &1.\ A_{11} = U'^T_{11} U'_{11} \\ &2.\ A_{12} = U'^T_{11} U'_{12} \\ &3.\ A_{22} = U'^T_{12} U'_{12} + U'^T_{22} U'_{22}. \end{aligned} \tag{3.8}$$

In Step 2 of Algorithm 3.1 it is recursively applied to the matrix A_{11}. Since it is already proven that A_{11} is positive semi-definite matrix, inductive hypothesis yields that theorem holds for A_{11}. Hence by uniqueness (Theorem 3.1) of the matrix U'_{11} (for A_{11}) we conclude that $U'_{11} = U_{11}$. This confirms the first equation in (3.2). From the second equation in (3.8) we have that $U'^T_{11} U'_{12} = U^T_{11} U'_{12} = A_{12}$. Therefore the matrix equation $U^T_{11} X = A_{12}$ has a solution. From Theorem 1.2.5 [13] yields that $U^T_{11} (U^{(1)}_{11})^T A_{12} = A_{12}$ is satisfied for arbitrary $\{1\}$ inverse of the matrix U_{11}. By the inductive hypothesis holds $Y_{11} \in U_{11}\{1\}$ and therefore it is satisfied

$$U^T_{11} Y^T_{11} A_{12} = A_{12}. \tag{3.9}$$

According to (3.9) and Step 3 of Algorithm 3.1′, we obtain

$$A_{12} = U^T_{11} Y^T_{11} A_{12} = U^T_{11} U_{12},$$

which is the second equation in (3.2).

To prove the last equation in (3.2) we only need to show that $A_{22} - U^T_{12} U_{12}$ is positive semi-definite matrix. Consider arbitrary vector $y \in \mathbb{R}^{n-k \times n-k}$. Let $x_1 = -Y_{11} Y^T_{11} A_{12} y$, $x_2 = y$ and $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. From the positive-semidefinitness of matrix A and $A_{11} = U^T_{11} U_{11}$ we have

$$\begin{aligned} 0 \le x^T A x &= x_1^T A_{11} x_1 + 2 x_1^T A_{12} x_2 + x_2^T A_{22} x_2 \\ &= y^T A^T_{12} Y_{11} Y^T_{11} U^T_{11} U_{11} Y_{11} Y^T_{11} A_{12} y - 2 y^T A^T_{12} Y_{11} Y^T_{11} A_{12} y + y^T A_{22} y. \end{aligned}$$

According to the inductive hypothesis, we have $Y_{11} \in U_{11}\{1, 2, 3\}$ and

$$U_{11} Y_{11} Y^T_{11} = (U_{11} Y_{11})^T Y^T_{11} = (Y_{11} U_{11} Y_{11})^T = Y^T_{11}. \tag{3.10}$$

Therefore, one can verify the following:

$$\begin{aligned} 0 \le x^T A x &= y^T A_{22} y - y^T A^T_{12} Y_{11} Y^T_{11} A_{12} y \\ &= y^T (A_{22} - U^T_{12} U_{12}) y. \end{aligned} \tag{3.11}$$

Since $y^T(A_{22} - U^T_{12} U_{12}) y \ge 0$ holds for arbitrary vector $y \in \mathbb{R}^{n-k \times n-k}$, the matrix $A_{22} - U^T_{12} U_{12}$ is positive semi-definite. From the inductive hypothesis we have that $U^T_{22} U_{22} = A_{22} - U^T_{12} U_{12}$. Therefore we proved that U satisfies (3.2), i.e. that holds $U^T U = A$.

Next we will prove that if $u_{ii} = 0$ for some $i = 1, \ldots, n$ then $u_{ij} = y_{ji} = 0$ for each $j = 1, \ldots, n$. In other words we have to prove that if i-th diagonal element of U is equal to 0, then i-th row of U as well as i-th column of Y are zero. Suppose that $u_{ii} = 0$. Denote $k = \lfloor n/2 \rfloor$.

If $i > k$ then element u_{ii} is the diagonal element of the matrix U_{22}. According to the inductive hypothesis, the $i - k$-th row of matrix U_{22} and $i - k$-th column of matrix Y_{22} is zero. Block decomposition (3.1) yields that i-th row of U is zero. In order to prove that i-th column of Y is also zero, we have to prove that $i - k$-th column of Y_{12} is zero. This holds from $Y_{12} = T_3 Y_{22} = -Y_{11} U_{12} Y_{22}$.

Otherwise, if $i \le k$ then u_{ii} is diagonal element of U_{11}. From the inductive hypothesis, i-th row of U_{11} and i-th column of Y_{11} are zero. Using that $U_{12} = Y^T_{11} A_{12}$ we can proceed as in the previous case.

Therefore we have proven that matrix U is the generalized Cholesky decomposition from Theorem 3.1.

Rest we need to prove that Y is $\{1, 2, 3\}$ inverse of the matrix U. From Theorem 1 in [14] it directly holds that Y is $\{2\}$ inverse of U. We can prove that Y is also $\{1\}$ inverse of U by direct verification of the equation $UYU = U$. It is easy to verify

$$UYU = \begin{bmatrix} U_{11} Y_{11} U_{11} & U_{11} Y_{11} U_{12} + U_{11} Y_{12} U_{22} + U_{12} Y_{22} U_{22} \\ & U_{22} Y_{22} U_{22} \end{bmatrix}. \tag{3.12}$$

From the inductive hypothesis we have $U_{11} Y_{11} U_{11} = U_{11}$ and $U_{22} Y_{22} U_{22} = U_{22}$. From Step 3 of Algorithm 3.1′ and (3.10) we have that

$$\begin{aligned} U_{11} Y_{11} U_{12} &= U_{11} Y_{11} Y^T_{11} A_{12} \\ &= Y^T_{11} A_{12} \\ &= U_{12}. \end{aligned} \tag{3.13}$$

From (3.13) and definition of Y_{12} in Step 5 of Algorithm 3.1′ we have

$$\begin{aligned} U_{11}Y_{11}U_{12} + U_{11}Y_{12}U_{22} + U_{12}Y_{22}U_{22} &= U_{12} - U_{11}Y_{11}U_{12}Y_{22}U_{22} + U_{12}Y_{22}U_{22} \\ &= U_{12} - U_{12}Y_{22}U_{22} + U_{12}Y_{22}U_{22} \\ &= U_{12}. \end{aligned} \tag{3.14}$$

Therefore, we prove that $Y \in U\{1\}$.

To prove that $Y \in U\{3\}$, we verify that the matrix UY is symmetric. It is not difficult to verify

$$UY = \begin{bmatrix} U_{11}Y_{11} & U_{11}Y_{12} + U_{12}Y_{22} \\ 0 & U_{22}Y_{22} \end{bmatrix}. \tag{3.15}$$

Again by definition of blocks Y_{12}, U_{12} and Y_{22} in Algorithm 3.1′ we have

$$\begin{aligned} U_{11}Y_{12} + U_{12}Y_{22} &= -U_{11}Y_{11}U_{12}Y_{22} + Y_{11}^{\mathrm{T}}A_{12}Y_{22} \\ &= -U_{11}Y_{11}Y_{11}^{\mathrm{T}}A_{12}Y_{22} + Y_{11}^{\mathrm{T}}A_{12}Y_{22} = 0. \end{aligned} \tag{3.16}$$

To prove (3.16), once more we used the property $U_{11}Y_{11}Y_{11}^{\mathrm{T}} = Y_{11}^{\mathrm{T}}$. Therefore, the statement $Y \in U\{1, 2, 3\}$ is proved. From (3.15) and (3.16) and the inductive hypothesis we have that the matrix UY is diagonal with all main diagonal entries equal to 0 or 1. □

4. Rapid computation of generalized inverses

In this section we will show how results obtained in the previous sections can be used in computing the Moore–Penrose and various classes of {2, 3} and {2, 4} generalized inverses. The main result will be an algorithm which computes the Moore–Penrose and {2, 3} and {2, 4} inverses in time $\Theta(\mathrm{mul}(n))$, under the assumption that the matrix inversion is of cost no more than the matrix multiplication.

Basis for our method are the results presented in [15,6]. Courrieu in [15] used the Cholesky decomposition of the matrix $A^{\mathrm{T}}A$ for computing the Moore–Penrose inverse $A^{\dagger}$.

Lemma 4.1 ([15]). *Let A be given $m \times n$ real matrix, and $S^{\mathrm{T}}S$ generalized Cholesky decomposition of the matrix $A^{\mathrm{T}}A$. If the matrix L^{T} is obtained from S by dropping zero rows, the Moore–Penrose inverse of A satisfies the following relation*

$$A^{\dagger} = L(L^{\mathrm{T}}L)^{-1}(L^{\mathrm{T}}L)^{-1}L^{\mathrm{T}}A^{\mathrm{T}}. \tag{4.1}$$

By combining the generalized Cholesky decomposition method (Algorithm 3.1′) and full recursive inversion method (Algorithm 2.1) we can compute the Moore–Penrose inverse in time $\Theta(\mathrm{mul}(n))$ using the relation (4.1).

Algorithm 4.1 (*Computing the Moore–Penrose Inverse in Matrix Multiplication Complexity, Based on the Generalized Cholesky Factorization*).

```
Input:    Matrix $A \in \mathbb{R}_r^{m\times n}$.
Step 1.   Form the matrix $A' = A^{\mathrm{T}}A$.
Step 2.   Find the generalized Cholesky factorization $A' = U^{\mathrm{T}}U$ of matrix $A'$ using Algorithm 3.1′.
Step 3.   Obtain the matrix $L^{\mathrm{T}}$ by dropping zero rows from matrix $U$. Form the matrix $T = L^{\mathrm{T}}L$.
Step 4.   Find the inverse $M = T^{-1}$ using Algorithm 2.1.
Step 5.   Return the Moore–Penrose inverse of $A$, defined by the formula
                $A^{\dagger} = LM^2L^{\mathrm{T}}A^{\mathrm{T}}$.
```

In order to prove correctness of Algorithm 4.1, it is necessary that all main diagonal minors of matrix T, defined in *Step* 3, are regular. We will prove this fact.

Theorem 4.1. *Matrix $T = L^{\mathrm{T}}L$ defined in Step 3 of Algorithm 4.1 is regular, symmetric, positive definite and all main diagonal minors of T are regular.*

Proof. We have that $A' = A^{\mathrm{T}}A \in \mathbb{R}^{n\times n}$ is symmetric and positive semi-definite. Using the results from [15] one can conclude that T is regular, symmetric and positive definite. To show that application of Algorithm 2.1 is possible in Step 4. of Algorithm 4.1, it is sufficient to prove that all main diagonal minors of T are regular. Let $T_{(S)}$ be a principal minor of T defined by the corresponding index set $S \subset \{1, \ldots, n\}$. For arbitrary non-zero vector $x \in \mathbb{R}^{|S|}$ the following is valid:

$$x^{\mathrm{T}}T_{(S)}x = x^{\mathrm{T}}L_{(S)}^{\mathrm{T}}L_{(S)}x = (L_{(S)}x)^{\mathrm{T}}L_{(S)}x.$$

Since $L_{(S)}$ is of full column rank, we have $L_{(S)}x \neq 0$, which implies $x'^{\mathrm{T}}T_{(S)}x' > 0$. Therefore, $T_{(S)}$ is of full column and full row rank, i.e. invertible. □

All steps of Algorithm 4.1 work in time $\Theta(\text{mul}(n))$, so this is also the complexity of whole Algorithm 4.1.

In the paper [6] (Theorem 2.1), Courrieu's method is generalized for computing various classes of generalized inverses including $\{1, 2, 3\}$, $\{1, 2, 4\}$, $\{2, 3\}_s$ and $\{2, 4\}_s$ inverses. We use this result for constant real matrices.

Using representations of generalized inverses from [6] together with the results from the previous section, we obtain the method for computing all mentioned classes of generalized inverses in time $\Theta(\text{mul}(n))$. Also the corresponding storage complexity is again $\Theta(n^2)$ if holds $\text{muls}(n) = \Theta(n^2)$. These are actually two analogous methods (other one is obtained by exchanging expressions in brackets)

Algorithm 4.2 (*Computing Generalized Inverses in the Matrix Multiplication Time, Based on the Generalized Cholesky Factorization*).

```
Input:   Matrix $A \in \mathbb{R}_r^{m\times n}$ and the matrix $R \in \mathbb{R}_s^{m\times n_1}$ (respectively $T \in \mathbb{R}_s^{m_1\times n}$), where $0 < s \le r$ and $m_1, n_1 \ge s$.
Step 1.  Form the matrix $P = (AT^{\mathrm{T}})(AT^{\mathrm{T}})^{\mathrm{T}}$ or $Q = (R^{\mathrm{T}}A)^{\mathrm{T}}(R^{\mathrm{T}}A)$.
Step 2.  Find the generalized Cholesky factorization $P = U^{\mathrm{T}}U$ or $Q = U^{\mathrm{T}}U$, by applying Algorithm 3.1′.
Step 3.  Obtain the matrix $L^{\mathrm{T}}$ by dropping zero rows from matrix $U$. Form the matrix $T = L^{\mathrm{T}}L$.
Step 4.  Find the inverse $M = T^{-1}$ using Algorithm 2.1.
Step 5.  Return the matrix $X_M = LM^2L^{\mathrm{T}}A^{\mathrm{T}}RR^{\mathrm{T}}$ or $X_N = T^{\mathrm{T}}TA^{\mathrm{T}}LM^2L^{\mathrm{T}}$.
```

Next theorem verifies the correctness of Algorithm 4.2.

Theorem 4.2. *Consider the arbitrary matrix $A \in \mathbb{R}_r^{m\times n}$. Let $0 < s \le r$ be any chosen integer and assume that m_1, n_1 are positive integers satisfying $m_1, n_1 \ge s$. Then Algorithm 4.2 satisfies the following statements:*

(a) *In the case $s < r$, $X_M \in A\{2, 4\}_s$.*
(b) *In the case $s < r$, $X_N \in A\{2, 3\}_s$.*
(c) *In the case $s = r$, $X_M \in A\{1, 2, 4\}$.*
(d) *In the case $s = r$, $X_N \in A\{1, 2, 3\}$.*
(e) *When $R = A (T = A)$ inverse X_M obtained in case (c) (X_N obtained in case (d)) reduces to $A^\dagger$.*

Moreover Algorithm 4.2 works in time $\Theta(\text{mul}(n))$ if we use the Strassen method for the matrix–matrix multiplication and the matrix inversion. The storage complexity of Algorithm 4.2 is $\Theta(n^2)$.

Proof. Matrices P and Q are symmetric and positive semi-definite. Therefore, *Step* 2 of Algorithm 4.2 is correct. Applying the same principle as in the proof of Theorem 4.1, one can verify that $L^{\mathrm{T}}L$ and all its main diagonal minors are regular. This means that *Step* 4 of Algorithm 4.2 is correct. Then the statements (a)–(e) follows from Theorem 2.1 in [6].

Applying $\text{inv}(n) = \Theta(n^{2+\epsilon})$ and Theorem 3.2, it is clear that all steps in Algorithm 4.2 are of the complexity $\Theta(\text{mul}(n))$. Therefore, we can compute any $\{2, 3\}$ and $\{2, 4\}$ generalized inverse in time $\Theta(\text{mul}(n))$.

Since $\text{invs}(n) = \Theta(n^2)$ and $\text{Chols}(n) = \Theta(n^2)$ we can easily conclude that storage complexity of Algorithm 4.2 is $\Theta(n^2)$. □

5. Numerical experience

Algorithms introduced in previous sections are implemented in the symbolic programming package MATHEMATICA (version 6.0). Source code is included in the Appendix. Matrix multiplication in MATHEMATICA works with complexity $\mathcal{O}(n^3)$, so that this is also the working time of our algorithms implementation.

Example 5.1. We will demonstrate how Algorithms 4.1 and 4.2 work on the matrix

$$A = \begin{bmatrix} 8 & 8 & 1 & 1 \\ 10 & 10 & 2 & 2 \\ 11 & 11 & 3 & 3 \\ 12 & 12 & 4 & 4 \end{bmatrix}.$$

By applying Algorithm 3.1′ on the matrix $A^{\mathrm{T}}A$ we obtain the following generalized Cholesky decomposition for $A^{\mathrm{T}}A$:

$$A^{\mathrm{T}}A = U^{\mathrm{T}}U \text{ where } U = \begin{bmatrix} \sqrt{429} & \sqrt{429} & \frac{109}{\sqrt{429}} & \frac{109}{\sqrt{429}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{\frac{989}{429}} & \sqrt{\frac{989}{429}} \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The generalized inverse Y of matrix U returned by Algorithm 3.1′ is equal to

$$Y = \begin{bmatrix} \frac{1}{\sqrt{429}} & 0 & -\frac{109}{\sqrt{424281}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{\frac{429}{989}} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

and satisfies the equations (1), (2) and (3) with respect to matrix U. This agrees with the results contained in Theorem 3.3. By direct verification we have that matrix UY is equal to diag(1, 0, 1, 0) which also agrees with Theorem 3.3. Subsequently, main diagonal of UY correspond to the zero rows in U and zero columns in Y respectively. Eq. (4) is not satisfied, but the matrix YU is upper triangular with only two non-zero entries above the main diagonal

$$YU = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Also, observe that $u_{22} = u_{44} = 0$. Since the second and fourth row in U are zero rows (together with the second and fourth column in Y), this confirms Theorem 3.3.

The next step is obtaining the matrix L^T by dropping the zero rows from U and inverting the matrix L^TL using Algorithm 2.1. It holds

$$L = \begin{bmatrix} \sqrt{429} & 0 \\ \sqrt{429} & 0 \\ \frac{109}{\sqrt{429}} & \sqrt{\frac{989}{429}} \\ \frac{109}{\sqrt{429}} & \sqrt{\frac{989}{429}} \end{bmatrix}.$$

Further we obtain matrix M as follows

$$L^TL = \begin{bmatrix} \frac{391844}{429} & \frac{218\sqrt{989}}{429} \\ \frac{218\sqrt{989}}{429} & \frac{1978}{429} \end{bmatrix} \Longrightarrow M = (L^TL)^{-1} = \begin{bmatrix} \frac{1}{858} & -\frac{109}{858\sqrt{989}} \\ -\frac{109}{858\sqrt{989}} & \frac{97961}{424281} \end{bmatrix}.$$

Now from the last step of Algorithm 4.1 we obtain the following Moore–Penrose inverse of the matrix A:

$$A^\dagger = \begin{bmatrix} \frac{131}{1978} & \frac{41}{989} & \frac{3}{1978} & -\frac{38}{989} \\ \frac{131}{1978} & \frac{41}{989} & \frac{3}{1978} & -\frac{38}{989} \\ -\frac{443}{1978} & -\frac{116}{989} & \frac{44}{989} & \frac{204}{989} \\ -\frac{443}{1978} & -\frac{116}{989} & \frac{44}{989} & \frac{204}{989} \end{bmatrix}.$$

Example 5.2. For matrices

$$A = \begin{bmatrix} 30 & 78 & 54 & 66 & 66 & 42 & 60 \\ 42 & 89 & 55 & 70 & 82 & 51 & 74 \\ 78 & 113 & 34 & 55 & 127 & 66 & 98 \\ 96 & 115 & 80 & 113 & 137 & 108 & 166 \end{bmatrix}, \qquad \operatorname{rank}(A) = 3$$

$$R = \begin{bmatrix} 52 & 9 & 23 & 40 & 35 & 5 & 37 \\ 92 & 54 & 72 & 64 & 56 & 30 & 68 \\ 4 & 18 & 16 & 0 & 0 & 10 & 4 \\ 22 & 54 & 51 & 8 & 7 & 30 & 19 \end{bmatrix}, \qquad \operatorname{rank}(R) = 2$$

by applying Algorithm 4.2 we obtain the following {2, 4} inverse of A with rank 2:

$$A_2^{(2,4)} = \begin{bmatrix} -\frac{2064786876}{231354215041} & -\frac{4755131732}{694062645123} & \frac{1424383465}{694062645123} & \frac{5928345641}{694062645123} \\ \frac{4016182158}{231354215041} & \frac{3180731937}{231354215041} & -\frac{1034331045}{462708430082} & -\frac{6217922757}{462708430082} \\ \frac{2682758156}{231354215041} & \frac{6419265925}{694062645123} & -\frac{848359916}{694062645123} & -\frac{5890267708}{694062645123} \\ -\frac{2365817440}{231354215041} & -\frac{1833498584}{231354215041} & \frac{471776128}{231354215041} & \frac{2133596416}{231354215041} \\ -\frac{2070090260}{231354215041} & -\frac{1604311261}{231354215041} & \frac{412804112}{231354215041} & \frac{1866896864}{231354215041} \\ \frac{2231212310}{231354215041} & \frac{5301219895}{694062645123} & -\frac{1723885075}{1388125290246} & -\frac{10363204595}{1388125290246} \\ -\frac{1177605336}{231354215041} & -\frac{2692445825}{694062645123} & \frac{893635321}{694062645123} & \frac{3528049673}{694062645123} \end{bmatrix}.$$

Example 5.3. Our implementations are tested on several random generated test examples.

In the first three tables we compared the performance of our implementation of Algorithm 3.1′ (function `Ch`) with the implementation of the generalized Cholesky decomposition from [7], given in [6] (function `Cholesky`). Due to the implementation of matrix–matrix multiplication in MATHEMATICA time complexities of both implementations are $\mathcal{O}(n^3)$. All presented times are in seconds. Each table item is average time on 20 different random generated test matrices with the same rank and order. Matrix dimensions are 2^k for $k = 4, 5, 6, 7$ together with the values $\lfloor 2^k\sqrt{2}\rfloor$ also for $k = 4, 5, 6, 7$.

n	`Ch`	`Cholesky` [6]	n	`Ch`	`Cholesky` [6]	n	`Ch`	`Cholesky` [6]
16	0.010	0	16	0.0047	0.0031	16	0.0	0.0
23	0.011	0.005	23	0.016	0.023	23	0.003	0.0062
32	0.018	0.016	32	0.0256	0.0171	32	0.0094	0.0188
45	0.021	0.047	45	0.031	0.047	45	0.0094	0.0436
64	0.052	0.120	64	0.042	0.1232	64	0.0378	0.1308
90	0.078	0.328	90	0.078	0.328	90	0.0346	0.3306
128	0.151	0.905	128	0.1498	0.9207	128	0.078	0.9422
180	1.295	3.562	180	0.265	2.481	180	0.195	2.5895
			256	3.24	10.5428	256	0.5616	7.9872
			362	8.83	348.6	362	1.17	197.622
			512	25.9895	1959.15	512	12.32	983.4
rank $A = n$			rank $A = n/2$			rank $A = n/10$		

In the following three tables we compare implementation of Algorithm 4.1 with corresponding MATHEMATICA implementation of Algorithm 2.1 from [6]. These algorithms compute the Moore–Penrose inverse.

n	Algorithm 4.1	Algorithm 2.1 in [6]	n	Algorithm 4.1	Algorithm 2.1 in [6]	n	Algorithm 4.1	Algorithm 2.1 in [6]
16	0.00775	0	16	0	0.00375	16	0.00775	0
23	0.016	0.0155	23	0.01575	0	23	0	0.004
32	0.031	0.01575	32	0.01575	0.0115	32	0	0.0155
45	0.039	0.0505	45	0.02325	0.03525	45	0.00775	0.0115
64	0.05025	0.133	64	0.043	0.08175	64	0.0235	0.01575
90	0.09725	0.3315	90	0.08975	0.2225	90	0.043	0.03125
128	0.17125	0.94	128	0.15225	0.7645	128	0.1015	0.06625
180	0.31175	2.62075	180	0.2925	2.57	180	0.17175	0.164
rank $A = n$			rank $A = n/2$			rank $A = n/10$		

Finally in the last three tables we compare our implementation of Algorithm 4.2 with corresponding implementation of Algorithm 2.1 in [6]. These algorithms compute {2, 3} and {2, 4} inverses.

n	Algorithm 4.2	Algorithm 2.1 in [6]	n	Algorithm 4.2	Algorithm 2.1 in [6]	n	Algorithm 4.2	Algorithm 2.1 in [6]
16	0.0115	0.008	16	0.0115	0.004	16	0	0
23	0.0195	0.00775	23	0.0195	0.008	23	0.02725	0.01175
32	0.02375	0.02325	32	0.02725	0.01575	32	0.0195	0.02725
45	0.03125	0.04675	45	0.03125	0.05475	45	0.0315	0.0505
64	0.05075	0.1365	64	0.0545	0.1405	64	0.05475	0.14025
90	0.09375	0.35125	90	0.0935	0.35475	90	0.09775	0.38575
128	0.17175	0.98675	128	0.16425	0.97875	128	0.152	1.09975
180	0.328	2.699	180	0.30025	2.777	180	0.28075	2.96
rank A = rank $R = n$			rank A = rank $R = n/2$			rank A = rank $R = n/10$		

From the arranged results we can conclude that our implementation has better working times on almost all examples. Performances of function Ch as well as implementations of Algorithms 4.1 and 4.2 are additionally degraded due to the slow recursion calls. Therefore, we can conclude that the implementations of algorithms introduced in this paper possess better performances with respect to corresponding MATHEMATICA implementations of basic method in [7] and Algorithm 2.1 in [6], even in the case when matrix multiplication runs in $\mathcal{O}(n^3)$.

For suitable implementation of matrix multiplication which runs in $\mathcal{O}(n^{2+\epsilon})$ Algorithm 3.1 will also run in the same time and possess better performance.

6. Conclusion

Our main result in the present paper can be formulated as follows: **generalized inversion is no harder than matrix multiplication**. Guided by the same result for inverting regular matrices (see for example [2]), we tend to use the Strassen algorithm for fast matrix multiplication and true block recursive algorithms.

Since algorithms for computing various classes of $\{2, 3\}$ and $\{2, 4\}$ inverses are based on the generalized Cholesky factorization, introduced in [7], it was necessary to develop rapid algorithms for the usual and generalized Cholesky factorization, which work in the matrix multiplication time. The starting point in the achievement of this goal was the factorization routine based on the recursion, introduced in Jonsson and Gustavson [10]. We introduce a new Strassen-type recursive algorithm for the Cholesky factorization of a given symmetric, positive definite matrix $A \in \mathbb{R}^{n\times n}$. Our algorithm is based on different approach than in [10], and computes both the Cholesky factorization matrix U and its inverse Y. We also presented the extension of our algorithm to the set of positive semi-definite matrices (possibly singular). In the singular case we generate the matrix U from the generalized Cholesky decomposition $A = U^{\mathrm{T}}U$ and generalized inverse $Y \in U\{1, 2, 3\}$. By combining these results with known representations from [15,6], we state algorithms which compute the Moore–Penrose inverse and various classes of $\{2, 3\}$ and $\{2, 4\}$ inverses. Using a method for matrix multiplication from [3] or from [4] and the method for matrix inversion from [3], we prove that introduced algorithms work in time $\Theta(\mathrm{mul}(n))$, which is the main result of the paper.

Proposed algorithms are implemented in MATHEMATICA and tested on randomly generated test set matrices. Testing results show that our algorithms have better running times even in the case when the matrix multiplication runs with complexity $\mathcal{O}(n^3)$.

Appendix. Implementation details and code in MATHEMATICA

Generalized Cholesky factorization, determined in Algorithm 3.1′ is implemented in the following MATHEMATICA function.

```
Ch[AA_] :=
    Module[{A, U11, U11inv, U22inv, U12, U22, A11, A12, A21, A22,
          Uinv, U, m, n, m1p, n1p, n1},
      A = AA;
      {m, n} = Dimensions[A];
      A = Chop[A, 10^(-6)];
      If [Simplify[A] == 0*A, Return[{0*A, 0*A}]];
      If [n == 1,
        If[A[[1, 1]] < 0, Return[{0*A, 0*A}]];
        Return[{{{Sqrt[A[[1, 1]]]}}, {{1/Sqrt[A[[1, 1]]]}}}]
        ];
      n1 = n/2 // Floor;
      m1p = n1p = n1;
      A11 = A // Take[#, m1p] & // Transpose[Take[Transpose[#], n1p]] &;
      A12 = A // Take[#, m1p] & // Transpose[Drop[Transpose[#], n1p]] &;
      A21 = A // Drop[#, m1p] & // Transpose[Take[Transpose[#], n1p]] &;
      A22 = A // Drop[#, m1p] & // Transpose[Drop[Transpose[#], n1p]] &;

      {U11, U11inv} = Ch[A11];
      If [Simplify[U11] == 0*U11,
        U12 = 0*A12;,
        U12 = Transpose[U11inv].A12;
      ];
      {U22, U22inv} = Ch[A22 - Transpose[U12].U12];
      U =
        Join[Transpose[Join[Transpose[U11], Transpose[U12]]],
          Transpose[Join[Transpose[0*A21], Transpose[U22]]]];
      Uinv =
        Join[Transpose[
            Join[Transpose[U11inv], Transpose[-U11inv.U12.U22inv]]],
          Transpose[Join[Transpose[0*A21], Transpose[U22inv]]]];
      Return[{U, Uinv} // Simplify]
    ];
```

Function Adop is auxiliary, and implement Step 3 in Algorithms 4.1 and 4.2.

```
Adop[a_List] := Module[{m, n, a1, i},
    {m, n} = Dimensions[a];
    a1 = Transpose[a];
    Do[
      If [Chop[Norm[Abs[a1[[i]]]], 10^(-6)] == 0,
        a1 = Drop[a1, {i}];
        ];
      , {i, Length[a1], 1, -1}
      ];
    Return[Transpose[a1]];
  ];
```

The following functions implement Algorithm 4.1 (function MoorePenroseCh) and Algorithm 4.2 (functions Inverse1Ch and Inverse2Ch).

```
MoorePenroseCh[A_] := Module[{U, Uinv, M},
    {U, Uinv} = Ch[Transpose[A].A];
    U = Adop[Transpose[U]];
    M = Inverse[Transpose[U].U // Simplify];
    Return[U.M.M.Transpose[U].Transpose[A]];
  ];

Inverse1Ch[A_, R_] := Module[{A1, U, Uinv, M},
    A1 = Transpose[R].A;
    {U, Uinv} = Ch[Transpose[A1].A1];
    U = Adop[Transpose[U]];
    M = Inverse[Transpose[U].U];
    Return[U.M.M.Transpose[U].Transpose[A1].Transpose[R] // Simplify];
  ];

Inverse2Ch[A_, T_] := Module[{A1, U, Uinv, M},
    A1 = A.Transpose[T];
    {U, Uinv} = Ch[A1.Transpose[A1]];
    U = Adop[Transpose[U]];
    M = Inverse[Transpose[U].U];
    Return[Transpose[T].Transpose[A1].U.M.M.Transpose[U] // Simplify];
  ];
```

References

[1] A. Ben-Israel, T.N.E. Greville, Generalized Inverses: Theory and Applications, second ed., Springer, 2003.
[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, Second ed., The MIT Press, Cambridge, Massachusetts, London, 2001, McGraw-Hill Book Company, Boston Burr Ridge, IL Dubuque, IA Madison, WI New York San Francisco St. Louis Montreal Toronto.
[3] V. Strassen, Gaussian elimination is not optimal, Numer. Math. 13 (1969) 354–356.
[4] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progression, J. Symb. Comput. 9 (1990) 251–280.
[5] T. Banachiewicz, Zur Berechnung der Determinanten, wie auch der Inversen und zur darauf basierten Auflosung der Systeme linearer Gleichungen, Acta Astron. Ser. C 3 (1937) 41–67.
[6] P. Stanimirović, M. Tasić, Computing generalized inverses using LU factorization of matrix product, Int. J. Comput. Math., doi:10.1080/00207160701582077.
[7] P. Courrieu, Straight monotonic embedding of data sets in Euclidean spaces, Neural Netw. 15 (2002) 1185–1196.
[8] S.M. Balle, P.C. Hansen, N.J. Higham, A Strassen-type matrix inversion algorithm, Danish computing center for research and education, in: PaA2 Deliverable APPARC ESPRIT Contract, Building 305, DK-2800 Lyngby, Technical University of Denmark, Denmark, 1993.
[9] N.J. Highman, Exploiting fast matrix multiplication within the level 3 BLAS, ACM Trans. Math. Softw. 16 (1990) 352–368.
[10] F.G. Gustavson, I. Jonsson, Minimal-storage high-performance Cholesky factorization via recursion and blocking, IBM J. Res. Dev. 44 (6) (2000) 823–850.
[11] F.G. Gustavson, Recursion leads to automatic variable blocking for dense linear algebra algorithms, IBM J. Res. Dev. 41 (6) (1997) 737–755.
[12] S. Toledo, Locality of reference in LU decomposition with partial pivoting, SIAM J. Matrix Anal. Appl. 18 (1997) 1065–1081.
[13] G. Wang, Y. Wei, S. Qiao, Generalized Inverses: Theory and Computations, Science Press, 2003.
[14] F. Burns, D. Carlson, E. Haynsworth, T. Markham, Generalized inverse formulas using Schur complement, SIAM J. Appl. Math. 26 (1974).
[15] P. Courrieu, Fast computation of Moore–Penrose inverse matrices, Neural Inform. Process. Lett. Rev. 8 (2) (2005) 25–29.
[16] S.K. Abdali, D.S. Wise, Experiments with quadtree representation of matrices, in: P. Gianni (Ed.), Proc. ISSAC 88, in: LNCS, vol. 358, Springer Verlag, 1989, pp. 96–108.
[17] A.R. Meenakshi, N. Anandam, Polynomial generalized inverses of a partitioned polynomial matrix, J. Indian Math. Soc. 58 (1) (1992) 11–18.