J. Math. Computer Sci., 37 (2025), 59–81

Online: ISSN 2008-949X

Journal of Mathematics and Computer Science

Journal Homepage: www.isr-publications.com/jmcs

A fast method to estimate the Moore-Penrose inverse for well-determined numerical rank matrices based on the Tikhonov regularization

Pablo Soto-Quiros

Escuela de Matemática, Instituto Tecnológico de Costa Rica, Cartago 30101, Costa Rica.

Abstract

This paper introduces a novel approach for estimating the Moore-Penrose inverse. The method proposed relies on Tikhonov regularization, which requires the computation of all positive singular values of an $m \times n$ matrix. Additionally, we present a highly efficient and accurate procedure for estimating these singular values. This procedure assumes the well-determined numerical rank of matrices A^*A (if $m \geqslant n$) and AA^* (if $m \leqslant n$). Furthermore, we demonstrate the application of our proposed method in solving linear discrete well-posed problems. The paper concludes with numerical simulations to illustrate the advantages of our novel approach. Notably, we compare the execution time associated with our technique to that of some relevant methods in the existing literature, demonstrating that our method outperforms others in terms of computational efficiency. To further substantiate our findings, we conduct computational experiments to measure execution time and speedup. The results affirm the efficiency of our proposed method, showcasing reduced execution times compared to other methods. This contributes to establishing our approach's practical viability and effectiveness in diverse applications.

Keywords: Moore-Penrose inverse, Tikhonov regularization, singular values, well-determined numerical rank.

2020 MSC: 65F15, 65F20, 65F22, 65F45, 65H10.

1. Introduction

Let $\mathbb{F}^{m\times n}$ be the set of all $m \times n$ matrices in $\mathbb{F}$, where $\mathbb{F}$ represents either $\mathbb{R}$ (real numbers) or $\mathbb{C}$ (complex numbers). The Moore-Penrose inverse of $A \in \mathbb{F}^{m\times n}$, denoted by $A^\dagger \in \mathbb{F}^{n\times m}$, is the unique matrix satisfying the following four Penrose equations:

$$\text{(i) } AA^\dagger A = A; \quad \text{(ii) } A^\dagger AA^\dagger = A^\dagger; \quad \text{(iii) } (AA^\dagger)^* = AA^\dagger; \quad \text{(iv) } (A^\dagger A)^* = A^\dagger A,$$

where $A^* \in \mathbb{F}^{n\times m}$ stands for the conjugate transpose of A (note that if $\mathbb{F} = \mathbb{R}$, then $A^* = A^T$). The Moore-Penrose inverse, also commonly called pseudoinverse, has been used in several applications, such as in the solutions of linear least-squares problems [6, 62], image and signal processing [35, 45, 57, 61], multilinear regression [25], and data analysis [26, 63].

The singular value decomposition (SVD) is the most popular method to compute $A^\dagger$ [37]. However, the SVD technique becomes impractical in the case of large matrices due to its computational complexity

Email address: jusoto@tec.ac.cr (Pablo Soto-Quiros)

doi: 10.22436/jmcs.037.01.05

Received: 2024-01-30 *Revised:* 2024-05-24 *Accepted:* 2024-08-10

[1]. Therefore, to reduce the computational time, several fast algorithms have been developed to estimate $A^\dagger$. For example, Courrieu [18] presents an algorithm to approximate the Moore-Pensore inverse based on reverse order law and a full-rank Cholesky factorization of singular symmetric positive matrices. Furthermore, in [5, 34, 58], researchers develop a set of methods based on QR decomposition to increase the computation speed of $A^\dagger$. Other related works are developed in [4, 7, 8, 10, 14, 17, 60].

Another relevant technique to estimate $A^\dagger$ is presented in [6, 13, 33, 36]. This method is explained as follows: if $A \in \mathbb{F}^{m\times n}$ is full-rank, then the Moore-Penrose inverse is computed by

$$A^\dagger = \begin{cases} (A^*A)^{-1}A^*, & \text{if } m \geqslant n, \\ A^*(AA^*)^{-1}, & \text{if } m \leqslant n. \end{cases} \tag{1.1}$$

Otherwise, if A is a rank-deficient matrix, then $A^\dagger$ is estimated by

$$A^\dagger \approx A_p(\alpha) = \begin{cases} (A^*A + \alpha I_n)^{-1}A^*, & \text{if } m \geqslant n, \\ A^*(AA^* + \alpha I_m)^{-1}, & \text{if } m \leqslant n, \end{cases} \tag{1.2}$$

where $\alpha > 0$ is the regularization parameter, and $I_m \in \mathbb{F}^{m\times m}$ and $I_n \in \mathbb{F}^{n\times n}$ are identity matrices. Here, the letter p in $A_p(\alpha)$ represents the initial letter of word *pseudoinverse*. Formula (1.2) is an approximation of Tikhonov regularization in order to estimate $A^\dagger$ (see, e.g., [6, Theorem 4.3]). This version of Tikhonov regularization establishes that if $\alpha \to 0$, then $A_p(\alpha) \to A^\dagger$. A suitable value of α is an essential part of computing $A_p(\alpha)$ accurately.

In this paper, we propose a new approach to estimate $A^\dagger$ for rank-deficient matrices using formula (1.2) such that

$$\|A^\dagger - A_p(\alpha)\|_{fr}^2 \leqslant tol, \tag{1.3}$$

where $\|\cdot\|_{fr}$ is the Frobenius norm and $tol > 0$ is a given tolerance. This innovative method outlines the procedure for selecting the regularization parameter, denoted as α, to ensure the validity of the inequality provided in (1.3). In particular, we demonstrate that when all positive singular values of matrix A are known, it is possible to compute α in a manner that satisfies (1.3). Moreover, we introduce an algorithm designed for estimating the Moore-Penrose inverse, denoted as $A^\dagger$. The algorithm is based on (1.1) for full-rank matrices and (1.2) for rank-deficient matrices. This algorithmic approach accommodates the specific characteristics of both full-rank and rank-deficient matrices, providing a comprehensive and versatile solution for estimating the Moore-Penrose inverse.

The proposed algorithm involves the computation of all positive singular values of the matrix A. To enhance computational efficiency, we assume that the Gram matrix associated with A is a well-determined numerical rank matrix. A matrix is considered to have a well-determined numerical rank if its positive singular values are lower-bounded by a constant dependent on machine precision [28, 65]. Further insights into well-determined numerical rank matrices are provided in Section 3 below.

Additionally, we demonstrate an application of the proposed method in solving linear discrete well-posed problems. We derive a dynamic absolute error bound between the solutions of an error-free consistent linear system problem and a noisy linear system problem with a regularization parameter α.

Afterwards, numerical simulations are presented to highlight the advantages of the proposed method. We illustrate that the execution time associated with the proposed method for estimating $A^\dagger$ is lower than that of other methods in the literature, particularly for large matrices. Moreover, the computational effort required for this new method is substantially less than that of the algorithm used in MATLAB. Furthermore, we present the efficiency and accuracy of the proposed method in a real-life application involving image denoising.

The remainder of the paper is structured as follows. Notation, definitions, and auxiliary results are included in Section 2. Section 3 provides a concise explanation of the concept of well-determined numerical rank matrices, along with relevant results. The main results, which enable the determination of

the regularization parameter α ensuring the validity of inequality (1.3), are presented in Section 4. An application of the proposed method to calculate a solution of a noisy linear discrete well-posed problem is explained in Section 5. Section 6 details how to efficiently compute the proposed technique for estimating the Moore-Penrose inverse. Numerical experiments are reported in Section 7. Finally, Section 8 contains some concluding remarks.

2. Preliminaries

Throughout this paper, we use the following notation and definitions. We define the Gram matrix of $A \in \mathbb{F}^{m\times m}$ as the matrix T such that

$$T = \begin{cases} A^*A \in \mathbb{R}^{n\times n}, & \text{if } m \geqslant n, \\ AA^* \in \mathbb{R}^{m\times m}, & \text{if } m \leqslant n, \end{cases} \tag{2.1}$$

Let $P = U_P\Sigma_P V_P^*$ be the singular value decomposition of $P \in \mathbb{F}^{m\times n}$, where $U_P \in \mathbb{F}^{m\times m}$ and $V_P \in \mathbb{F}^{n\times n}$ are two unitary matrices, i.e., $U_P^*U_P = U_PU_P^* = I_m$ and $V_P^*V_P = V_PV_P^* = I_n$ and $\Sigma_P \in \mathbb{F}^{m\times n}$ is a generalized diagonal matrix defined by $\Sigma_P = \operatorname{diag}(\sigma_1(P), \sigma_2(P), \ldots, \sigma_{\min\{m,n\}}(P))$. Here, $\sigma_1(P) \geqslant \sigma_2(P) \geqslant \cdots \geqslant \sigma_{\min\{m,n\}}(P)$ are the singular values of P. In addition, we consider the following matrix norms:

- Frobenius norm: $\|P\|_{fr} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|P(i,j)|^2}$;
- 1-norm: $\|P\|_1 = \max\limits_{1\leqslant j\leqslant n}\sum_{i=1}^{m}|P(i,j)|$;
- 2-norm: $\|P\|_2 = \sigma_1(P)$.

We denote the eigenvalues of $Q \in \mathbb{F}^{m\times m}$ by $\lambda_i(Q)$, for $i = 1, \ldots, m$. Moreover, if all eigenvalues of Q are in $\mathbb{R}$, they are arranged from the largest to the smallest number, i.e., $\lambda_1(Q) \geqslant \lambda_2(Q) \geqslant \cdots \geqslant \lambda_m(Q)$.

The condition number of a square matrix Q, represented by $\operatorname{cond}_p(Q)$ is given by $\operatorname{cond}_p(Q) = \|Q\|_p\|Q^{-1}\|_p$, where $p = 1, 2$. If Q is rank-deficient, then $\operatorname{cond}_p(Q) = \infty$.

Before proceeding to the next section, we consider the following lemmas, which will facilitate the proof of the main results in this paper.

Lemma 2.1 ([37, Proposition 8.1]). *For $P \in \mathbb{F}^{m\times n}$ and $y \in \mathbb{F}^n$, $\|Py\|_2 \leqslant \|P\|_{fr}\|y\|_2$.*

Lemma 2.2 ([68, Theorem 2]). *For $P \in \mathbb{F}^{m\times n}$, $\operatorname{cond}_1(P) \leqslant \sqrt{mn}\,\operatorname{cond}_2(P)$.*

Lemma 2.3. *If $a, b > 0$, then $\dfrac{1}{b^2(b^2+a)^2} \leqslant \dfrac{1}{b^6}$.*

Proof. Note that

$$0 \leqslant 2ab^2 + a^2 \Rightarrow b^4 \leqslant (b^2+a)^2 \Rightarrow b^6 \leqslant b^2(b^2+a)^2 \Rightarrow \frac{1}{b^2(b^2+a)^2} \leqslant \frac{1}{b^6}.$$

□

3. Well-determined numerical rank matrices

We define $P \in \mathbb{F}^{m\times n}$ as a well-determined numerical rank matrix [28, 65] if

$$\sigma_i(P) > \max\{m, n\}\cdot\sigma_1(P)\cdot \texttt{eps},$$

for all $i = 1, \ldots, \operatorname{rank}(P)$, where $\texttt{eps} = 2^{-52}$ is the machine precision for IEEE floating-point arithmetic with double precision [53]. This procedure is used to estimate the rank of a matrix in various programming

languages for numerical computations, for example, MATLAB [42], GNU Octave [47], Julia [32], Scilab [56], Python with NumPy library [46], and C++ with Armadillo library [3].

Below, we present two new results related to Gram matrices and well-determined numerical rank matrices.

Lemma 3.1. *The singular values of $A \in \mathbb{F}^{m\times n}$ satisfy the condition*

$$\sigma_i(A) > \max\{m,n\}\cdot\sigma_1(A)\cdot\sqrt{\texttt{eps}},$$

if and only if the Gram matrix of A is a well-determined numerical rank matrix.

Proof. Suppose w.l.g. that $m \geqslant n$. Let $T = A^*A$ be the Gram matrix of A. Based on the facts that $m^2 \geqslant n^2 > n$, and $\sigma_i^2(A) = \sigma_i(T)$, we obtain

$$\begin{aligned}\sigma_i(A) > m\cdot\sigma_1(A)\cdot\sqrt{\texttt{eps}} &\Leftrightarrow \sigma_i^2(A) > m^2\cdot\sigma_1^2(A)\cdot\texttt{eps}\\ &\Leftrightarrow \sigma_i(T) > m^2\cdot\sigma_1(T)\cdot\texttt{eps} \Leftrightarrow \sigma_i(T) > n\cdot\sigma_1(T)\cdot\texttt{eps}.\end{aligned}$$

The proof is similar if we consider $m < n$. □

Lemma 3.2. *If the Gram matrix of $A \in \mathbb{F}^{m\times m}$ has a well-determined numerical rank, then A also has a well-determined numerical rank.*

Proof. Let $T = A^*A$ be the Gram matrix of A. Based on the facts that $\sigma_i^2(A) = \sigma_i(T)$ and $\sigma_i(A) \leqslant \sigma_1(A)$, for all $i = 1, 2, \ldots, m$, we obtain

$$\begin{aligned}\sigma_i(T) > m\cdot\sigma_1(T)\cdot\texttt{eps} &\Rightarrow \sigma_i^2(A) > m\cdot\sigma_1^2(A)\cdot\texttt{eps}\\ &\Rightarrow \left(\frac{\sigma_i(A)}{\sigma_1(A)}\right)^2 > m\cdot\texttt{eps} \Rightarrow \frac{\sigma_i(A)}{\sigma_1(A)} > m\cdot\texttt{eps} \Rightarrow \sigma_i(A) > m\cdot\sigma_1(A)\cdot\texttt{eps}.\end{aligned} \tag{3.1}$$

It follows from (3.1) that A has a well-determined numerical rank. □

4. Main results

This section presents the results that allow us to derive the theoretical method to calculate the regularization parameter α such that inequality (1.3) is valid.

Theorem 4.1. *If $A \in \mathbb{F}^{m\times n}$, $r = rank(A)$ and $\alpha > 0$, then*

$$\|A^\dagger - A_p(\alpha)\|_{fr}^2 = \sum_{i=1}^{r}\frac{\alpha^2}{\sigma_i^2(A)(\sigma_i^2(A)+\alpha)^2}.$$

Proof. Suppose w.l.g. that $m \geqslant n$. Let $A = U_A\Sigma_A V_A^*$ be the SVD of A, and thus, $A^\dagger = V_A\Sigma_A^\dagger U_A^*$. Note that $\Sigma_A^*\Sigma_A + \alpha I_n$ is positive definite and, accordingly, is invertible. Therefore, $A_p(\alpha)$ can be expressed as follows

$$\begin{aligned}A_p(\alpha) = (A^*A+\alpha I_n)^{-1}A^* &= (V_A\Sigma_A^*\Sigma_A V_A^* + \alpha V_A V_A^*)^{-1}V_A\Sigma_A^*U_A^*\\ &= (V_A(\Sigma_A^*\Sigma_A+\alpha I_n)V_A^*)^{-1}V_A\Sigma^*U_A^* = V_A(\Sigma_A^*\Sigma_A+\alpha I_n)^{-1}\Sigma_A^*U_A^*.\end{aligned}$$

Based on the fact that the Frobenius norm is unitarily invariant, then we obtain that

$$\|A^\dagger - A_p(\alpha)\|_{fr}^2 = \|V_A\Sigma_A^\dagger U_A^* - V_A(\Sigma_A^*\Sigma_A+\alpha I_n)^{-1}\Sigma_A^*U_A^*\|_{fr}^2 = \|\Sigma_A^\dagger - (\Sigma_A^*\Sigma_A+\alpha I_n)^{-1}\Sigma_A^*\|_{fr}^2. \tag{4.1}$$

Note that $\Sigma_A^\dagger$, $\Sigma_A^*\Sigma_A + \alpha I_n$, and Σ_A^* are diagonal matrices. Moreover,

- $\Sigma_A^{\dagger} = \text{diag}(1/\sigma_1(A), \ldots, 1/\sigma_r(A), 0, \ldots, 0) \in \mathbb{F}^{n\times m}$;
- $(\Sigma_A^* \Sigma_A + \alpha I_n)^{-1} = \text{diag}(1/(\sigma_1^2(A)+\alpha), \ldots, 1/(\sigma_r^2(A)+\alpha), 1/\alpha, \ldots, 1/\alpha) \in \mathbb{F}^{n\times n}$;
- $\Sigma_A^* = \text{diag}(\sigma_1(A), \ldots, \sigma_r(A), 0, \ldots, 0) \in \mathbb{F}^{n\times m}$.

Thus,

$$\Sigma_A^{\dagger} - (\Sigma_A^* \Sigma_A + \alpha I_n)^{-1} \Sigma_A^* = \text{diag}(\widehat{\sigma}_1, \ldots, \widehat{\sigma}_r, 0, \ldots, 0) \in \mathbb{F}^{n\times m}, \tag{4.2}$$

where

$$\widehat{\sigma}_i = \frac{1}{\sigma_i(A)} - \frac{\sigma_i(A)}{\sigma_i^2(A)+\alpha} = \frac{\alpha}{\sigma_i(A)(\sigma_i^2(A)+\alpha)}, \tag{4.3}$$

for all $i = 1, 2, \ldots, r$. Finally, from (4.1), (4.2), and (4.3), we obtain that

$$\|A^{\dagger} - A_p(\alpha)\|_{fr}^2 = \|\Sigma_A^{\dagger} - (\Sigma_A^* \Sigma_A + \alpha I_n)^{-1} \Sigma_A^*\|_{fr}^2 = \|\text{diag}(\widehat{\sigma}_1, \ldots, \widehat{\sigma}_r, 0, \ldots, 0)\|_{fr}^2 = \sum_{i=1}^{r} \frac{\alpha^2}{\sigma_i^2(A)(\sigma_i^2(A)+\alpha)^2}.$$

The proof of the case when $m \leqslant n$ is similar to the proof presented above. □

Theorem 4.2. *Let us define* $A \in \mathbb{F}^{m\times n}$ *and* $r = \text{rank}(A)$. *If* $\text{tol} > 0$ *is a known tolerance and*

$$0 < \alpha \leqslant \sqrt{\frac{\text{tol}}{\sum_{i=1}^{r} \frac{1}{\sigma_i^6(A)}}}, \tag{4.4}$$

then $\|A^{\dagger} - A_p(\alpha)\|_{fr}^2 \leqslant \text{tol}$.

Proof. From Lemma 2.3, we obtain that

$$\frac{1}{\sigma_i^2(A)(\sigma_i^2(A)+\alpha)^2} \leqslant \frac{1}{\sigma_i^6(A)}, \tag{4.5}$$

for all $i = 1, \ldots, r$. Thus, it follows from Theorem 4.1 and (4.5) that

$$\|A^{\dagger} - A_p(\alpha)\|_{fr}^2 = \sum_{i=1}^{r} \frac{\alpha^2}{\sigma_i^2(A)(\sigma_i^2(A)+\alpha)^2} \leqslant \alpha^2 \sum_{i=1}^{r} \frac{1}{\sigma_i^6(A)} \leqslant \frac{\text{tol}}{\sum_{i=1}^{r} \frac{1}{\sigma_i^6(A)}} \sum_{i=1}^{r} \frac{1}{\sigma_i^6(A)} = \text{tol}.$$

□

5. An application of Theorem 4.2: linear discrete well-posed problems

Theorem 4.2 in Section 4 can be applied in the solution of linear discrete well-posed problems when $\mathbb{F} = \mathbb{R}$. A linear discrete well-posed problem is a system of equations

$$Ax = b, \tag{5.1}$$

where $A \in \mathbb{R}^{m\times n}$ has a well-determined numerical rank, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. Here, b is contaminated by an error $e \in \mathbb{R}^m$, i.e., $b = \hat{b} + e$, where $\hat{b} \in \mathbb{R}^m$ denotes the unavailable error-free right-hand side. The error-free linear system of equations

$$Ax = \hat{b}, \tag{5.2}$$

is assumed to be consistent. The least-square solution of the minimal Euclidean norm of the unavailable linear system of equations (5.2) is given by $\hat{x}_0 = A^{\dagger}\hat{b}$. The least-squares solution of the minimal Euclidean

norm of (5.1), given by $x_0 = A^\dagger b$, is not a significant approximation of $\hat{x}_0$ due to the error e. The Tikhonov regularization for linear systems is a method that modifies (5.1) so that the sensitivity of the solution to the error e is reduced [12]. This method replaces the solution of the linear system in (5.1) with the minimization problem

$$\min_{x\in\mathbb{R}^n} \|Ax-b\|_2^2 + \alpha\|x\|_2^2, \tag{5.3}$$

for a suitable $\alpha > 0$. The unique solution of (5.3) is given by

$$x_0(\alpha) = A_p(\alpha)b, \tag{5.4}$$

where $A_p(\alpha)$ is defined in (1.2). A suitable value of α is an essential part of Tikhonov regularization. There are several methods for choosing α, such that the discrepancy principle [11], when $\|e\|_2$ is known, or the generalized cross validation [24], when $\|e\|_2$ is unknown. Based on Theorem 4.2, we propose a new method to choose a regularization parameter α such that $\|\hat{x}_0 - x_0(\alpha)\|_2^2$ is less than or equal to a given tolerance tol > 0. In this theorem, we assume that $\|e\|_2$ is available and $\|e\|_2 \neq 0$.

Theorem 5.1. *Let $\hat{x}_0$ and $x_0(\alpha)$ be the solution of problems (5.2) and (5.3), respectively. If* tol > 0 *is a known tolerance and*

$$0 < \alpha \leqslant \frac{\|e\|_2}{\|b\|_2}\sqrt{\frac{\frac{\text{tol}}{\|e\|_2^2} - \sum_{i=1}^{r}\frac{1}{\sigma_i^2(A)}}{\sum_{i=1}^{r}\frac{1}{\sigma_i^6(A)}}}, \tag{5.5}$$

then $\|\hat{x}_0 - x_0(\alpha)\|_2^2 \leqslant$ tol.

Proof. From Lemma 2.1, we obtain that

$$\begin{aligned}\|\hat{x}_0 - x_0(\alpha)\|_2^2 &= \|A^\dagger\hat{b} - A_p(\alpha)b\|_2^2\\ &= \|A^\dagger b - A_p(\alpha)b - A^\dagger e\|_2^2\\ &\leqslant \|(A^\dagger - A_p(\alpha))b\|_2^2 + \|A^\dagger e\|_2^2\\ &\leqslant \|A^\dagger - A_p(\alpha)\|_{fr}^2\|b\|_2^2 + \|A^\dagger\|_{fr}^2\|e\|_2^2.\end{aligned} \tag{5.6}$$

Finally, it follows from (5.5), (5.6), Theorem 4.2, and the fact that $\|A^\dagger\|_{fr}^2 = \sum_{i=1}^{r}\frac{1}{\sigma_i^2(A)}$, that

$$\begin{aligned}\|\hat{x}_0 - x_0(\alpha)\|_2^2 &\leqslant \|A^\dagger - A_p(\alpha)\|_{fr}^2\|b\|_2^2 + \|A^\dagger\|_{fr}^2\|e\|_2^2\\ &\leqslant \alpha^2\|b\|_2^2\sum_{i=1}^{r}\frac{1}{\sigma_i^6(A)} + \|e\|_2^2\sum_{i=1}^{r}\frac{1}{\sigma_i^2(A)}\\ &\leqslant \frac{\|e\|_2^2}{\|b\|_2^2}\left(\frac{\frac{\text{tol}}{\|e\|_2^2} - \sum_{i=1}^{r}\frac{1}{\sigma_i^2(A)}}{\sum_{i=1}^{r}\frac{1}{\sigma_i^6(A)}}\right)\|b\|_2^2\sum_{i=1}^{r}\frac{1}{\sigma_i^6(A)} + \|e\|_2^2\sum_{i=1}^{r}\frac{1}{\sigma_i^2(A)} = \text{tol}.\end{aligned}$$

□

The following elementary example illustrates the proposed method performance based on Theorem 5.1.

Example 5.1. We consider the linear discrete well-posed problem $Ax = b$, where

$$A = \begin{pmatrix} 8 & 10 & 19 & 16 \\ 31 & 26 & 12 & 28 \\ 16 & 20 & 38 & 32 \\ 7 & 8 & 13 & 12 \\ 21 & 24 & 39 & 36 \end{pmatrix} \in \mathbb{R}^{5\times 4} \quad \text{and} \quad b = \begin{pmatrix} 0.55168 \\ 1.05861 \\ 1.17303 \\ 0.51678 \\ 1.30061 \end{pmatrix} \in \mathbb{R}^5.$$

Matrix A is a well-determined numerical rank matrix, where $\operatorname{rank}(A) = 2$, and its positive singular values are $\sigma_1(A) = \sqrt{5365 + 6\sqrt{643170}}$ and $\sigma_2(A) = \sqrt{5365 - 6\sqrt{643170}}$. Additionally, b is a noisy vector, i.e., $b = \hat{b} + e$, where

$$\hat{b} = \begin{pmatrix} 0.53 \\ 0.97 \\ 1.06 \\ 0.4 \\ 1.2 \end{pmatrix} \in \mathbb{R}^5 \quad \text{and} \quad e = \begin{pmatrix} 0.02168 \\ 0.08861 \\ 0.11303 \\ 0.11678 \\ 0.10061 \end{pmatrix} \in \mathbb{R}^5.$$

In this example, we assume that the error-free system $Ax = \hat{b}$ is unavailable. The only information available is the system $Ax = b$ and $\|e\|_2 \approx 0.2117958$. It follows from Theorem 5.1 that if $\text{tol} = 10^{-4}$, then

$$\frac{\|e\|_2}{\|b\|_2}\sqrt{\frac{\frac{\text{tol}}{\|e\|_2^2} - \sum_{i=1}^{r}\frac{1}{\sigma_i^2(A)}}{\sum_{i=1}^{r}\frac{1}{\sigma_i^6(A)}}} \approx 22.699338396628153,$$

where $r = \operatorname{rank}(A) = 2$. Therefore, if we choose $\alpha = 11.349650$, then $x_0(\alpha)$ in (5.4) is given by

$$x_0(\alpha) = \begin{pmatrix} 0.010251345936108 \\ 0.009918221395114 \\ 0.010349801643984 \\ 0.012780129138826 \end{pmatrix}.$$

Further, we obtain that $\|\hat{x}_0 - x_0(\alpha)\|_2^2 \approx 3.5964979 \times 10^{-6} < \text{tol}$, where $\hat{x}_0$ is the solution of consistent and error-free system $Ax = \widehat{b}$. Thus, Theorem 5.1 is illustrated numerically by this example.

6. Efficient and accurate implementation of the proposed method

This section presents the procedure to compute the Moore-Penrose inverse of $A \in \mathbb{R}^{m\times n}$, considering formulas (1.1) for full-rank matrices and (1.2) for rank-deficient matrices. Further, parameter regularization α in (1.2) is calculated using Theorem 4.2. As mentioned in the Section 1, we assume that the Gram matrix T of A, given by (2.1), is a well-determined numerical rank matrix.

Firstly, note that $\operatorname{rank}(A) = \operatorname{rank}(T)$. If T is rank-deficient, then $\operatorname{cond}_1(T) = \infty$. Otherwise, if T is full-rank and (w.l.g.) $m \geqslant n$, then it follows from Lemma 2.2, and the fact that T is a well-determined numerical rank matrix that

$$\operatorname{cond}_1(T) \leqslant n \cdot \operatorname{cond}_2(T) = n \cdot \frac{\sigma_1(T)}{\sigma_n(T)} \leqslant \frac{n \cdot \sigma_1(T)}{n \cdot \sigma_1(T) \cdot \text{eps}} \leqslant \frac{1}{\text{eps}}.$$

Therefore, numerically speaking, T is an ill-conditioned matrix if $\text{rcond}_1(T) < \texttt{eps}$, where $\text{rcond}_1(T)$ is the reciprocal of $\text{cond}_1(T)$. This condition is used in MATLAB to determine whether matrix is ill-conditioned [44]. Thus, since T is a well-determined numerical rank matrix in this paper, we state that if $\text{rcond}_1(T) < \texttt{eps}$, then A is rank-deficient; otherwise, if $\text{rcond}_1(T) \geqslant \texttt{eps}$, then A is full-rank. Moreover, we show in Lemma 3.2 that if $m = n$ and T is a well-determined numerical rank matrix, then A is also a well-determined numerical rank matrix. The complexity in flops to compute $\text{rcond}_1(T)$ is $2n^2 - n$, when $m \geqslant n$, and $2m^2 - m$, when $m \leqslant n$, approximately (see, e.g., [31]). Besides, it follows from Theorem 4.2 that α can be chosen by formula

$$\alpha = \frac{1}{2}\sqrt{\frac{\text{tol}}{\sum_{i=1}^{r}\frac{1}{\sigma_i^6(A)}}}, \tag{6.1}$$

where $r = \text{rank}(A)$. Positive singular values $\sigma_1(A), \ldots, \sigma_r(A)$ are calculated efficiently through eigenvalues of T. These eigenvalues can be computed using the QR algorithm with transformation to Hessenberg/tridiagonal form [2]. Note that T is a symmetric and positive semidefinite matrix, thus $\lambda_i(T) = \sigma_i(T)$. Additionally, as mentioned before, $\text{rank}(A) = \text{rank}(T) = r$. Therefore, to compute $\text{rank}(T)$ when (w.l.g.) $m \geqslant n$, we choose all eigenvalues $\lambda_i(T)$ such that

$$\lambda_i(T) > n \cdot \lambda_1(T) \cdot \texttt{eps}. \tag{6.2}$$

Thus, the rank of T is given by the number of eigenvalues such that (6.2) is rightful. Finally, positive singular values of A are the square root of positive eigenvalues of T, i.e., $\sigma_i(A) = \sqrt{\lambda_i(T)}$, for all $i = 1, \ldots, r$. Therefore, the regularization parameter α in (6.1) can already be calculated. The complexity in flops to compute all singular values of A using this procedure is $\frac{4}{3}n^3$, when $m \geqslant n$, and $\frac{4}{3}m^3$, when $m \leqslant n$, approximately (see, e.g., [2, Chapter 4]).

The last step is computing $A^\dagger$ using formulas (1.1) and (1.2). Both formulas are obtained by solving the following linear system:

$$\begin{aligned} (A^*A)X &= A^*, \quad \text{if } A \text{ is full-rank and } m \geqslant n, \\ X(AA^*) &= A^*, \quad \text{if } A \text{ is full-rank and } m \leqslant n, \\ (A^*A + \alpha I_n)X &= A^*, \quad \text{if } A \text{ is rank-deficient and } m \geqslant n. \\ X(AA^* + \alpha I_m) &= A^*, \quad \text{if } A \text{ is rank-deficient and } m \leqslant n. \end{aligned} \tag{6.3}$$

Matrices A^*A, AA^*, $A^*A + \alpha I_n$, and $AA^* + \alpha I_m$ are symmetric and positive definite. Therefore, we can solve the linear systems using the Cholesky decomposition. The complexity in flops to obtain X in (6.3) using this procedure is $\frac{1}{3}n^3 + 2n^2$ when $m \geqslant n$, and $\frac{1}{3}m^3 + 2m^2$ when $m \leqslant n$, approximately (see, e.g., [29]).

The implementation of the proposed method to estimate the Moore-Penrose inverse is presented in Algorithm 1. Further, Table 1 shows an approximation of the number of flops required to compute $A^\dagger$ using Algorithm 1.

6.1. Some relevant remarks on Algorithm 1

Remark 6.1. Algorithm 1 is designed for matrices with a well-determined numerical rank. Hilbert, Vandermonde, and Wilkinson matrices are examples of ill-determined numerical rank matrices [20], and therefore, Algorithm 1 approximates their Moore-Penrose inverses with poor accuracy. To avoid this issue, we can choose an algorithm that works for ill-determined numerical rank matrices, for example, the `qrginv` method [34]. This method approximates $A^\dagger$ based on QR decomposition. However, the execution time of the `qrginv` method is greater than that of the proposed method in this paper. Other methods to estimate $A^\dagger$ for ill-determined numerical rank matrices are developed in [4, 17].

Algorithm 1: Proposed method to estimate $A^{\dagger}$.

Input : $A \in \mathbb{R}^{m\times n}$, $tol > 0$
Output: $X \in \mathbb{R}^{n\times m}$

$A_t = A^*$
if $m \geqslant n$ **then**
  $T = A_tA$
  if $rcond_1(T) < eps$ **then**
    $\nu = [\lambda_1(T)\ \lambda_2(T)\ \cdots\ \lambda_n(T)]$
    $\beta = n \cdot \nu(1) \cdot eps$
    Find largest k such that $\nu(k) > \beta$
    $s = [\sqrt{\nu(1)}\ \sqrt{\nu(2)}\ \ldots\ \sqrt{\nu(k)}]$
    $\alpha = \frac{1}{2}\sqrt{tol/\sum_{j=1}^{k}\frac{1}{(s(i))^6}}$
    Find X such that $(T + \alpha I_n)X = A_t$
  else
    Find X such that $TX = A_t$
else
  $T = AA_t$
  if $rcond_1(T) < eps$ **then**
    $\nu = [\lambda_1(T)\ \lambda_2(T)\ \ldots\ \lambda_m(T)]$
    $\beta = m \cdot \nu(1) \cdot eps$
    Find largest k such that $\nu(k) > \beta$
    $s = [\sqrt{\nu(1)}\ \sqrt{\nu(2)}\ \ldots\ \sqrt{\nu(k)}]$
    $\alpha = \frac{1}{2}\sqrt{tol/\sum_{j=1}^{k}\frac{1}{(s(i))^6}}$
    Find X such that $X(T + \alpha I_m) = A_t$
  else
    Find X such that $XT = A_t$

Table 1: Approximation of the number of flops to estimate $A^{\dagger}$ using Algorithm 1.

Dimension	Rank condition	Steps	Flops	Total flops
$m \geqslant n$	Rank-deficient	1) $T = A^*A$	$(2m-1)n^2$	$\frac{5}{3}n^3 + 3n^2 + 2mn^2 - n + 6r$
		2) $rcond_1(T)$	$2n^2 - n$	
		3) $s = [\sigma_1(A)\ \ldots\ \sigma_r(A)]$	$\frac{4}{3}n^3$	
		4) $\alpha = \frac{1}{2}\sqrt{tol/\sum_{i=1}^{r}\frac{1}{(s(i))^6}}$	$6r$	
		5) Solve $(T + \alpha I_n)X = A^*$	$\frac{1}{3}n^3 + 2n^2$	
$m \geqslant n$	Full-rank	1) $T = A^*A$	$(2m-1)n^2$	$\frac{1}{3}n^3 + 3n^2 + 2mn^2 - n$
		2) $rcond_1(T)$	$2n^2 - n$	
		3) Solve $TX = A^*$	$\frac{1}{3}n^3 + 2n^2$	
$m \leqslant n$	Rank-deficient	1) $T = AA^*$	$(2n-1)m^2$	$\frac{5}{3}m^3 + 3m^2 + 2nm^2 - m + 6r$
		2) $rcond_1(T)$	$2m^2 - m$	
		3) $s = [\sigma_1(A)\ \ldots\ \sigma_r(A)]$	$\frac{4}{3}m^3$	
		4) $\alpha = \frac{1}{2}\sqrt{tol/\sum_{i=1}^{r}\frac{1}{(s(i))^6}}$	$6r$	
		5) Solve $X(T + \alpha I_n) = A^*$	$\frac{1}{3}m^3 + 2m^2$	
$m \leqslant n$	Full-rank	1) $T = AA^*$	$(2n-1)m^2$	$\frac{1}{3}m^3 + 3m^2 + 2nm^2 - m$
		2) $rcond_1(T)$	$2m^2 - m$	
		3) Solve $XT = A^*$	$\frac{1}{3}m^3 + 2m^2$	

Remark 6.2. The proposed method in Algorithm 1 is based on the fact that it needs to compute eigenvalues vector ν of the matrix T. To avoid this, there are iterative methods to approximate the pseudoinverse which involve only basic matrix operations such as multiplications and sums. For example, the classical Schulz's method [7] approximates $Z^{\dagger}$ with the iterative expression $X^{(k+1)} = X^{(k)}(2I_m - ZX^{(k)})$, for $k = 0, 1, 2, \ldots$, where $X^{(0)} = \gamma Z^T$ and $\gamma \in]0, 2/\|Z\|_2^2[$. However, the execution time of Schulz's method is greater than that of Algorithm 1. For example, if $D \in \mathbb{R}^{2500\times 2000}$ is a full-rank random matrix generated from the standard uniform distribution, then the Schulz's method approximates $D^{\dagger}$ in 354.80s, using 22 iterations such that $\|X^{(22)} - D^{\dagger}\|_{fr} < 10^{-3}$. Algorithm 1 computes $D^{\dagger}$ in 0.5485s. Other iterative methods to approximate the pseudoinverse are developed in [50–52, 69].

Remark 6.3. For the MATLAB code developed for Algorithm 1, we use the commands `T=A'*A` and `T=A*A'` to perform the computations in lines 3 and 14, respectively. These simple commands for computing the Gram matrix T take advantage of MATLAB's internal optimizations for matrix multiplication. MATLAB is engineered to handle matrix operations very efficiently, utilizing libraries such as BLAS and LAPACK [43, 64]. Alternative methods for efficiently computing the Gram matrix are discussed in [21, 30, 39], and are provided for the reader's further investigation.

Remark 6.4. If the Gram matrix T is ill-conditioned in Algorithm 1, then the proposed method computes the singular values of A using the eigenvalues of T to estimate constant α. However, if an eigenvalues solver is not available, an iterative method can be used to approximate the eigenvalues of T. For example, the QR algorithm can be used to approximate the eigenvalues of T, and consequently, the singular values of A. In this case, the approximation of the singular values of A may affect the accuracy of the calculation of the constant α in Algorithm 1, because the inequality (4.4) in Theorem 4.2 may not hold. In the following Corollary 6.1, we present a sufficient condition to ensure that the approximation of the singular values of A does not affect the accuracy of the calculation of its Moore-Penrose inverse.

Corollary 6.1. *Let us define* $A \in \mathbb{F}^{m\times n}$ *and* $r = \operatorname{rank}(A)$. *Consider* $\widehat{\sigma}_i(A) \in \mathbb{R}$, *which is an approximation of the* i*-th singular value* $\sigma_i(A)$. *If* $\operatorname{tol} > 0$ *is a known tolerance,*

$$0 \leqslant \sum_{i=1}^{r} \frac{\sigma_i^6(A) - \widehat{\sigma}_i^{\,6}(A)}{\sigma_i^6(A)\widehat{\sigma}_i^{\,6}(A)}, \tag{6.4}$$

and

$$0 < \widehat{\alpha} \leqslant \sqrt{\frac{\operatorname{tol}}{\sum_{i=1}^{r} \frac{1}{\widehat{\sigma}_i^{\,6}(A)}}}, \tag{6.5}$$

then $\|A^{\dagger} - A_p(\widehat{\alpha})\|_{fr}^2 \leqslant \operatorname{tol}$.

Proof. It follows from (6.4) that

$$\begin{aligned} 0 \leqslant \sum_{i=1}^{r} \frac{\sigma_i^6(A) - \widehat{\sigma}_i^{\,6}(A)}{\sigma_i^6(A)\widehat{\sigma}_i^{\,6}(A)} &\Rightarrow 0 \leqslant \sum_{i=1}^{r} \frac{1}{\widehat{\sigma}_i^{\,6}(A)} - \sum_{i=1}^{r} \frac{1}{\sigma_i^6(A)} \\ &\Rightarrow \frac{\sum_{i=1}^{r} \frac{1}{\sigma_i^6(A)}}{\operatorname{tol}} \leqslant \frac{\sum_{i=1}^{r} \frac{1}{\widehat{\sigma}_i^{\,6}(A)}}{\operatorname{tol}} \Rightarrow \sqrt{\frac{\operatorname{tol}}{\sum_{i=1}^{r} \frac{1}{\widehat{\sigma}_i^{\,6}(A)}}} \leqslant \sqrt{\frac{\operatorname{tol}}{\sum_{i=1}^{r} \frac{1}{\sigma_i^6(A)}}}. \end{aligned} \tag{6.6}$$

Based on (6.5) and (6.6), we obtain that

$$0 < \widehat{\alpha} \leqslant \sqrt{\frac{\operatorname{tol}}{\sum_{i=1}^{r} \frac{1}{\sigma_i^6(A)}}}. \tag{6.7}$$

Finally, inequality $\|A^{\dagger} - A_p(\widehat{\alpha})\|_{fr}^2 \leqslant \operatorname{tol}$ follows from (6.7) and Theorem 4.2. □

Corollary 6.1 indicates that if we have $\widehat{\sigma_1}(A), \ldots, \widehat{\sigma_k}(A)$, which are approximations of $\sigma_1(A), \ldots, \sigma_k(A)$, respectively, then these approximations can be used to select a constant $\widehat{\alpha}$ such that $\|A^\dagger - A_p(\widehat{\alpha})\|_{fr}^2 < tol$. To achieve this, the approximations of the singular values must satisfy inequality (6.4), and the constant $\widehat{\alpha}$ must satisfy inequality (6.5). Note that this result is very similar to Theorem 4.2, with the only difference being the addition of condition (6.4), which is sufficient to ensure that the final accuracy of the Moore-Penrose inverse approximation is not compromised.

7. Numerical experiments

In this section, we show numerically the advantages of the proposed Algorithm 1. The numerical examples were run on a desktop computer with a 2.80 GHz processor (Intel(R) Core(TM) i9-10900F CPU) and 32.00 RAM, using MATLAB R2019a. MATLAB code for numerical experiments is made available on GitHub (`https://github.com/jusotoTEC/fast_moore_penrose_inverse`).

The computational performance analysis of Algorithm 1 is evaluated using the metrics speedup and percent difference [19, p. 27]. Consider two methods that solve the same numerical problem, Method 1 and Method 2, with execution times t_1 and t_2, respectively. The speedup S (or acceleration) is the ratio between the execution times of both methods, i.e., $S = t_2/t_1$. If Method 1 is an improvement of Method 2, then S will be greater than 1. However, if Method 1 hurts performance, speedup will be less than 1. The percent difference P between Method 1 and Method 2, where $t_1 > t_2$, is represented by $P = 100\,(t_1 - t_2)\,/t_1 = 100(1 - S)$. Thus, we say that Method 2 is P% faster than Method 1. In addition, we use the following error formula to estimate the accuracy of the methods used in this paper:

$$E_0 = \|A^\dagger - X\|_{fr}^2, \tag{7.1}$$

where X is the approximation of $A^\dagger$ given by the methods used in this section. We assume that $A^\dagger$ is obtained in (7.1) by MATLAB's `pinv` command. We use the error formula given by (7.1) because the proposed method follows from the squared error given by (1.3). Recall that Theorem 4.2 estimates α such that $\|A^\dagger - X\|_{fr}^2 < tol$, where $X = A_p(\alpha)$.

7.1. Numerical experiment 1: Algorithm 1 vs. other methods

We first evaluate the efficiency of Algorithm 1 to approximate the Moore-Penrose inverse using random matrices. We compare our proposed method with MATLAB's `pinv` command, Courrieu method [18] (`geniv`), Katsikis et al. method [34] (`qrginv`), Alireza method [5] (`imqrginv`), and Stanimirović et al. method [58] (`ats2`). All algorithms were carefully implemented and tested in MATLAB, using the codes presented in the appendix section associated with each scientific article.

On the other hand, in this document, we do not consider iterative methods for approximating the Moore-Penrose inverse, such as those developed in [7, 50–52, 69], as these methods have a longer execution time compared to that obtained by Algorithm 1. However, the author of this scientific paper acknowledges the importance of these iterative methods for approximating the Moore-Penrose inverse, as they are highly efficient and useful when computational commands for approximating the eigenvalues of the Gram matrix T are unavailable because these iterative methods involve only basic matrix operations, such as multiplications and additions. In this experiment, we consider the following cases.

Case 1. Rectangular and rank-deficient: $A = BC \in \mathbb{F}^{m\times m/2}$, where $B \in \mathbb{F}^{m\times r}$, $C \in \mathbb{F}^{r\times m/2}$, and $r = m/4$. Matrices B and C are generated from a normal distribution with zero-mean and standard deviation 1.

Case 2. Square and rank-deficient: $A = DE \in \mathbb{F}^{m\times m}$, where $D \in \mathbb{F}^{m\times r}$, $E \in \mathbb{F}^{r\times m}$, and $r = m/2$. Matrices D and E are generated from a normal distribution with zero-mean and standard deviation 1.

Case 3. Rectangular and full-rank: $A \in \mathbb{F}^{m\times m/4}$, where A is generated from a normal distribution with zero-mean and standard deviation 1.

Table 2: Computational time, speedup, percent difference, and error for a matrix $A \in \mathbb{R}^{m \times m/2}$, where $\text{rank}(A) \leqslant m/4$.

Dimension (m)	Method	Time (s)	Speed	Percent difference	$E_0 = \|X - A^\dagger\|_{fr}^2$
5000	`pinv`	3.6022	—	—	—
	`geniv`	5.4915	0.65596	-52.449	1.8738×10^{-26}
	`qrginv`	4.7042	0.76574	-30.592	2.8070×10^{-32}
	`imqrginv`	4.4353	0.81216	-23.129	2.8157×10^{-32}
	`ats2`	4.1070	0.87710	-14.013	2.5780×10^{-32}
	Alg. 1	**2.6753**	1.34650	25.731	5.5519×10^{-17}
7500	`pinv`	14.854	—	—	—
	`geniv`	18.727	0.79317	-26.076	5.6634×10^{-27}
	`qrginv`	17.134	0.86692	-15.351	3.0134×10^{-32}
	`imqrginv`	16.055	0.92515	-8.0900	3.0206×10^{-32}
	`ats2`	15.438	0.96213	-3.9358	2.7367×10^{-32}
	Alg. 1	**10.051**	1.47790	32.334	5.5515×10^{-17}
10000	`pinv`	36.110	—	—	—
	`geniv`	44.017	0.82037	-21.896	3.9523×10^{-26}
	`qrginv`	41.017	0.86116	-16.122	3.1203×10^{-32}
	`imqrginv`	39.196	0.92127	-8.5455	3.1263×10^{-32}
	`ats2`	36.724	0.98328	-1.7009	2.6700×10^{-32}
	Alg. 1	**6.2085**	5.81620	82.807	5.5514×10^{-17}
12500	`pinv`	69.523	—	—	—
	`geniv`	84.746	0.82106	-21.794	8.3187×10^{-24}
	`qrginv`	79.954	0.87027	-14.907	3.6164×10^{-32}
	`imqrginv`	74.657	0.93202	-7.2933	3.6214×10^{-32}
	`ats2`	71.117	0.97841	-2.2072	2.8043×10^{-32}
	Alg. 1	**11.523**	6.03860	83.440	5.5513×10^{-17}
15000	`pinv`	119.68	—	—	—
	`geniv`	145.04	0.82514	-21.192	5.7433×10^{-24}
	`qrginv`	138.01	0.86721	-15.312	3.7257×10^{-32}
	`imqrginv`	128.25	0.93316	-7.1633	3.7299×10^{-32}
	`ats2`	121.85	0.98219	-1.8128	3.0424×10^{-32}
	Alg. 1	**18.897**	6.33330	84.210	5.5512×10^{-17}
17500	`pinv`	192.24	—	—	—
	`geniv`	226.27	0.84959	-17.704	3.1443×10^{-26}
	`qrginv`	219.03	0.87771	-13.933	3.6855×10^{-32}
	`imqrginv`	206.31	0.93182	-7.3170	3.6893×10^{-32}
	`ats2`	197.63	0.97274	-2.8021	3.2780×10^{-32}
	Alg. 1	**28.185**	6.82060	85.338	5.5512×10^{-17}
20000	`pinv`	282.73	—	—	—
	`geniv`	339.03	0.83392	-19.915	2.0298×10^{-24}
	`qrginv`	335.72	0.84215	-18.744	4.0717×10^{-32}
	`imqrginv`	312.91	0.90354	-10.676	4.0747×10^{-32}
	`ats2`	288.99	0.97832	-2.2161	3.3683×10^{-32}
	Alg. 1	**42.327**	6.67950	85.029	5.5512×10^{-17}

Table 3: Computational time, speedup, percent difference and error for a matrix $A \in \mathbb{R}^{m\times m}$, where $\text{rank}(A) \leqslant m/2$.

Dimension (m)	Method	Time (s)	Speed	Percent difference	$E_0 = \|X - A^\dagger\|_{fr}^2$
5000	`pinv`	30.822	—	—	—
	`geniv`	42.953	0.71758	-39.358	1.6697×10^{-17}
	`qrginv`	14.983	2.05710	51.388	1.1436×10^{-31}
	`imqrginv`	14.611	2.10960	52.597	1.1465×10^{-31}
	`ats2`	15.551	1.98200	49.546	1.1741×10^{-31}
	Alg. 1	**4.4183**	6.97600	85.665	5.5585×10^{-17}
7500	`pinv`	102.29	—	—	—
	`geniv`	142.16	0.71950	-38.986	8.5463×10^{-26}
	`qrginv`	51.826	1.97360	49.332	1.3125×10^{-31}
	`imqrginv`	49.563	2.06370	51.544	1.3148×10^{-31}
	`ats2`	50.746	2.01560	50.388	1.2433×10^{-31}
	Alg. 1	**13.668**	7.48330	86.637	5.5546×10^{-17}
10000	`pinv`	241.52	—	—	—
	`geniv`	337.36	0.71591	-39.682	4.9910×10^{-23}
	`qrginv`	112.95	1.96440	49.094	1.3721×10^{-31}
	`imqrginv`	117.34	2.05830	51.416	1.3737×10^{-31}
	`ats2`	119.90	2.01440	50.357	1.3950×10^{-31}
	Alg. 1	**30.662**	7.91880	87.305	5.5533×10^{-17}
12500	`pinv`	458.73	—	—	—
	`geniv`	652.97	0.70254	-42.341	4.4723×10^{-24}
	`qrginv`	236.09	1.94300	48.533	1.5433×10^{-31}
	`imqrginv`	222.04	2.06590	51.596	1.5446×10^{-31}
	`ats2`	231.02	1.98570	49.640	1.5722×10^{-31}
	Alg. 1	**57.930**	7.91880	87.372	5.5526×10^{-17}
15000	`pinv`	793.15	—	—	—
	`geniv`	1102.1	0.71967	-38.952	2.2361×10^{-24}
	`qrginv`	401.74	1.97430	49.349	1.7134×10^{-31}
	`imqrginv`	383.38	2.06880	51.663	1.7144×10^{-31}
	`ats2`	392.96	2.01840	50.455	1.8727×10^{-31}
	Alg. 1	**88.240**	8.98850	88.874	5.5522×10^{-17}
17500	`pinv`	1255.8	—	—	—
	`geniv`	1767.8	0.71036	-40.773	1.0311×10^{-23}
	`qrginv`	642.72	1.95390	48.819	1.7752×10^{-31}
	`imqrginv`	616.67	2.03640	50.893	1.7761×10^{-31}
	`ats2`	628.25	1.99890	49.971	1.8659×10^{-31}
	Alg. 1	**135.43**	9.27250	89.215	5.5520×10^{-17}
20000	`pinv`	1945.1	—	—	—
	`geniv`	2511.2	0.77459	-29.100	3.6809×10^{-22}
	`qrginv`	1009.8	1.92630	48.086	2.0294×10^{-31}
	`imqrginv`	967.50	2.01050	50.261	2.0301×10^{-31}
	`ats2`	962.08	2.02180	50.539	1.9991×10^{-31}
	Alg. 1	**210.45**	9.24250	89.181	5.5519×10^{-17}

Table 4: Computational time, speedup, percent difference and error for a matrix full-rank $A \in \mathbb{R}^{m \times m/4}$.

Dimension (m)	Method	Time (s)	Speed	Percent difference	$E_0 = \|X - A^{\dagger}\|_{fr}^2$
5000	`pinv`	0.4344	—	—	—
	`geniv`	0.7558	0.57476	-73.986	8.0716×10^{-30}
	`qrginv`	1.4996	0.28969	-245.19	6.7380×10^{-30}
	`imqrginv`	1.2618	0.34429	-190.46	6.6831×10^{-30}
	`ats2`	1.2630	0.34397	-190.73	6.8062×10^{-30}
	Alg. 1	**0.1544**	2.81280	64.448	6.2209×10^{-30}
7500	`pinv`	1.4421	—	—	—
	`geniv`	2.6721	0.53970	-85.287	9.6818×10^{-30}
	`qrginv`	5.2195	0.27629	-261.94	8.6065×10^{-30}
	`imqrginv`	4.9707	0.29012	-244.68	8.5398×10^{-30}
	`ats2`	4.6307	0.31142	-221.11	8.7234×10^{-30}
	Alg. 1	**0.5009**	2.87850	65.260	7.9578×10^{-30}
10000	`pinv`	4.5987	—	—	—
	`geniv`	7.1303	0.64496	-55.049	1.2172×10^{-29}
	`qrginv`	13.919	0.33038	-202.68	1.0764×10^{-29}
	`imqrginv`	13.328	0.34504	-189.82	1.0697×10^{-29}
	`ats2`	11.467	0.40106	-149.34	1.0822×10^{-29}
	Alg. 1	**1.1509**	3.99580	74.974	9.9773×10^{-30}
12500	`pinv`	10.353	—	—	—
	`geniv`	14.495	0.71428	-40.002	1.3407×10^{-29}
	`qrginv`	27.944	0.37050	-169.90	1.2230×10^{-29}
	`imqrginv`	25.645	0.40370	-147.71	1.2171×10^{-29}
	`ats2`	22.408	0.46204	-116.43	1.2286×10^{-29}
	Alg. 1	**2.1619**	4.78890	79.118	1.1288×10^{-29}
15000	`pinv`	17.567	—	—	—
	`geniv`	22.491	0.78107	-28.029	1.4464×10^{-29}
	`qrginv`	44.419	0.39548	-152.86	1.3669×10^{-29}
	`imqrginv`	43.595	0.40296	-148.17	1.3613×10^{-29}
	`ats2`	38.813	0.45261	-120.94	1.3786×10^{-29}
	Alg. 1	**3.3005**	5.32250	81.212	1.2654×10^{-29}
17500	`pinv`	29.314	—	—	—
	`geniv`	36.245	0.80879	-23.642	1.7233×10^{-29}
	`qrginv`	77.979	0.37593	-166.01	1.6361×10^{-29}
	`imqrginv`	73.323	0.39980	-150.13	1.6303×10^{-29}
	`ats2`	63.710	0.46012	-117.33	1.6459×10^{-29}
	Alg. 1	**4.9352**	5.93990	83.165	1.5241×10^{-29}
20000	`pinv`	45.552	—	—	—
	`geniv`	54.262	0.83948	-19.122	1.8559×10^{-29}
	`qrginv`	115.37	0.39483	-153.27	1.7713×10^{-29}
	`imqrginv`	108.47	0.41994	-138.13	1.7652×10^{-29}
	`ats2`	93.681	0.48624	-105.66	1.7762×10^{-29}
	Alg. 1	**7.6159**	5.98110	83.281	1.6428×10^{-29}

Tables 2, 3, and 4 present execution time, speedup, percent difference, and error associated with Cases 1, 2, and 3, respectively. Here, we consider $m \in \{5000, 7500, 10000, 12500, 15000, 17500, 20000\}$. Algorithm 1 uses machine precision as tolerance, i.e., $\text{tol} = \texttt{eps} = 2^{-52} \approx 2.2204 \times 10^{-16}$.

It follows from Tables 2, 3, and 4 that Algorithm 1 estimates $A^{\dagger}$ in less time than the other methods. Moreover, numerical results show that the speedup and percent difference of the proposed method increase as dimension m increases. Thus, Algorithm 1 is between 25% to 89% faster than MATLAB's `pinv` command. Besides, although the associated error of the new algorithm is slightly larger than that obtained by the other methods for rank-deficient matrices, the numerical results in Tables 2 and 3 show that the associated error with Algorithm 1 is less than the given tolerance $\text{tol} = \texttt{eps}$.

Therefore, this experiment illustrates that Algorithm 1 is faster than the other methods and preserves a good accuracy, which is less than the machine precision.

7.2. *Numerical experiment 2: matrices with known singular values*

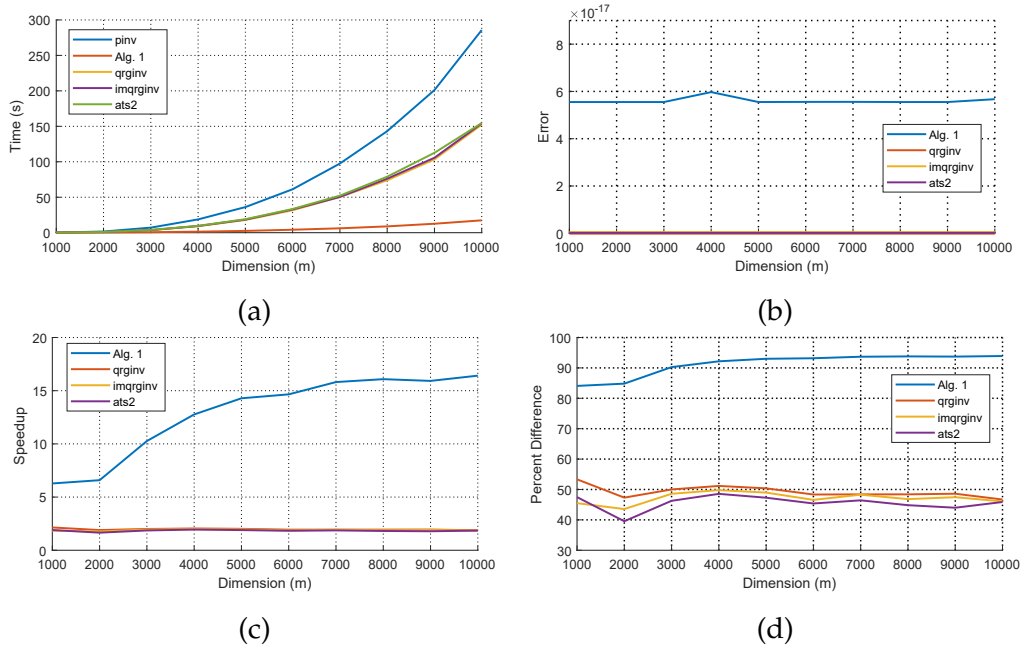


Figure 1: Diagrams of MATLAB's `pinv` command, Algorithm 1, `qrginv`, `imqrginv`, and `ats2` methods. The matrix dimension m versus: (a) execution time (in seconds); (b) error ($E_0 = \|X - A^{\dagger}\|_{fr}^2$); (c) speedup; (d) percent difference between the methods.

Note that Steps 5-8 and 16-19 in Algorithm 1 compute all positive singular values of A. Therefore, if the singular values of A are known previously, then the execution time of the proposed Algorithm 1 is significantly reduced. This numerical experiment estimates the Moore-Penrose inverse of a structured matrix, whose positive singular values are formerly known. Precisely, we consider the tridiagonal matrix $A \in \mathbb{R}^{m \times m}$ defined by

$$A = \begin{pmatrix} 1 & \sqrt{d} & & & \\ \sqrt{d} & d+1 & \sqrt{d} & & \\ & \sqrt{d} & \ddots & \ddots & \\ & & \ddots & d+1 & \sqrt{d} \\ & & & \sqrt{d} & d \end{pmatrix}, \tag{7.2}$$

where d is a positive real number. This matrix is used in statistical mechanics and mathematics in Bethe trees. Bethe trees, also called Bethe Lattice, are simple connected undirected graphs with no cycles [48, 54]. It follows from Lemma 8 in [54] that A is a rank-deficient matrix, where $\text{rank}(A) = m - 1$. Moreover, the nonzero eigenvalues of A are given by $\lambda_k(A) = d + 1 + 2\sqrt{d}\cos(\pi k/m)$, for $k = 1, 2, \ldots, m-1$. Note that

A is symmetric and $\lambda_k(A) > 0$, for all $k = 1, 2, \ldots, m-1$. Therefore, singular values of A are given by

$$\sigma_k(A) = d + 1 + 2\sqrt{d}\cos\left(\frac{\pi k}{m}\right), \tag{7.3}$$

for $k = 1, 2, \ldots, m-1$, and $\sigma_m(A) = 0$. In this experiment, we estimate the Moore-Penrose inverse of the rank-deficient matrix given by (7.2). In addition, Algorithm 1 is modified, so singular values are not computed. Instead, this algorithm receives all positive singular values of A given by (7.3). Numerical simulations consider $m = 1000k$, where $k \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, and $d = 10\alpha$, where α is generated from a uniform distribution with zero-mean and standard deviation 1. Moreover, Algorithm 1 uses machine precision as tolerance, i.e., $tol = 2^{-52}$.

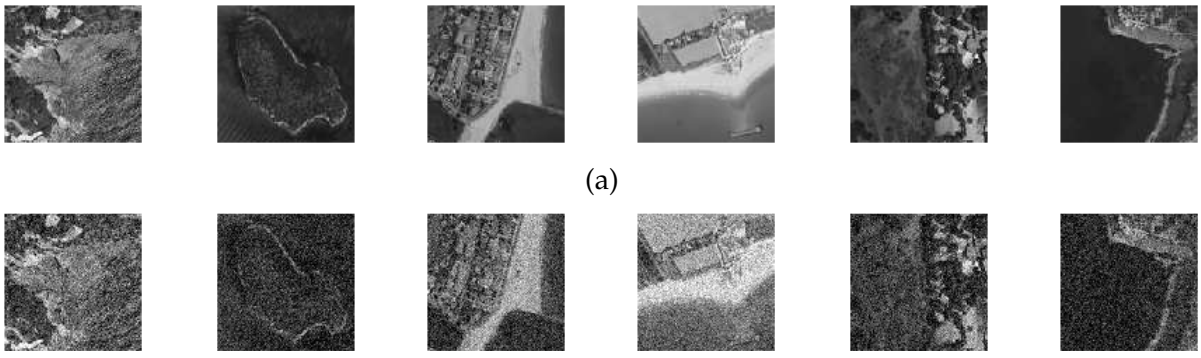
Figure 1 presents the diagrams of execution time, error, speedup, and percent difference associated with MATLAB's `pinv` command, Algorithm 1, `qrginv`, `imqrginv`, and `ats2` methods. This experiment does not consider the `geniv` method because its associated errors in numerical simulations are greater than 10^{-1}.

Figure 1 (a) shows that the proposed method estimates the Moore-Penrose inverse of the rank-deficient matrix (7.2) in less time than the other methods. Figures 1 (c) and (d) show that speedup and percent difference of Algorithm 1 increase as dimension m increases. Moreover, the proposed method in this paper is between 85% to 95% faster than MATLAB's `pinv` command. Similar to numerical experiment 1, the associated error of the new algorithm is slightly larger than that obtained by the other methods. However, Figure 1 (b) shows that the error of the proposed method is less than the given tolerance $tol = eps$.

Therefore, if all positive singular values of A are previously known, Algorithm 1 is much faster than the other methods. Moreover, the proposed method preserves a good accuracy, which is less than the machine's precision.

7.3. Numerical experiment 3: Real-life application in image denoising

Numerical simulations presented in Sections 7.1 and 7.2 showed that the execution time of Algorithm 1 is much less than MATLAB's `pinv` command and other methods in the literature. Although the error associated with this new method is slightly higher than that of different techniques for rank-deficient matrices, we can ensure that the estimation of the Moore-Penrose inverse is accurate because the associated error is less than machine precision when $tol = eps$. This numerical experiment presents a real-life application in denoising images [13, 15, 16, 57], where the accuracy of Algorithm 1 allows reconstructing a noisy image with the same precision as MATLAB's `pinv` command.



(a)

(b)

Figure 2: (a) Some randomly selected images of $\mathbb{X}$. (b) Noisy versions of images in (a).

Specifically, we use Algorithm 1 for the problem of filtering a noisy grayscale image $\overline{Y} \in \mathbb{R}^{128\times128}$ based on a set of training images $\mathbb{X} = \{X^{(1)}, \ldots, X^{(p)}\}$, where $X^{(j)} \in \mathbb{R}^{128\times128}$, for $j = 1, \ldots, p$. Training set $\mathbb{X}$ consists of $p = 1110$ grayscale images of coast scenes of optical aerial pictures from visible spectrum

(see Figure 2 (a) for six sample images). Images in $\mathbb{X}$ are chosen from the MASATI database [23]. The MASATI database comprises of satellite images labeled according to the following seven classes: land, coast, sea, ship, multi, coastship, and detail. In this experiment, we consider the coast class.

It is assumed that instead of images in $\mathbb{X}$, we observed their noisy version $\mathbb{Y} = \{Y^{(1)}, \dots, Y^{(p)}\}$, where $Y^{(j)} \in \mathbb{R}^{128\times128}$, for $j = 1, \dots, p$. Each $Y^{(j)}$ is simulated as $Y^{(j)} = X^{(j)} + 0.1N^{(j)}$, where $N^{(j)} \in \mathbb{R}^{128\times128}$ is a random matrix generated from a normal distribution with zero-mean and standard deviation 1. For each image $X^{(j)}$, matrix $N^{(j)}$ simulates noise (see Figure 2 (b) for six sample images). Here, we vectorize matrices $X^{(j)}$ and $Y^{(j)}$, i.e., convert each matrix into a column vector by stacking the columns of the matrix. Let $\text{vec} : \mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ be the vectorization transform. We write $x_j = \text{vec}(X^{(j)}) \in \mathbb{R}^{16384}$ and $y_j = \text{vec}(Y^{(j)}) \in \mathbb{R}^{16384}$, for $j = 1, 2, \dots, p$. If $X = [x_1\ x_2\ \cdots\ x_p] \in \mathbb{R}^{16384\times1110}$ and $Y = [y_1\ y_2\ \cdots\ y_p] \in \mathbb{R}^{16384\times1110}$, the goal of the image denoising problem is find a filter matrix $\widehat{F} \in \mathbb{R}^{16384\times16384}$ that gives a small reconstruction error for the training set, i.e., find $\widehat{F}$ such that

$$\|\widehat{F}Y - X\|_{fr}^2 = \min_{F\in\mathbb{R}^{16384\times16384}} \|FY - X\|_{fr}^2. \tag{7.4}$$

A solution of problem (7.4), given by Penrose [49], is $\widehat{F} = XY^\dagger$. We consider vectors $\bar{x} \in \mathbb{R}^{16384}$ and $\bar{y} \in \mathbb{R}^{16384}$, which are the vector representation of images $\overline{X} \in \mathbb{R}^{128\times128}$ and $\overline{Y} \in \mathbb{R}^{128\times128}$, respectively, i.e., $\bar{x} = \text{vec}(\overline{X})$ and $\bar{y} = \text{vec}(\overline{Y})$. Here, $\overline{Y}$ represents a noisy observed image of a desired source image $\overline{X}$. Figures 3 (a) and 3 (b) show images $\overline{X}$ and $\overline{Y}$ used in this numerical experiment. Finally, estimation of $\overline{X}$ is given by $\widetilde{X} = \text{vec}^{-1}(\widehat{F}\bar{y})$. Finally, we assume that noisy image for filtering $\overline{Y}$ does not necessarily belong to $\mathbb{Y}$, but it is "similar" to one of them, i.e., $\overline{Y} \in \mathbb{J}(\delta)$, where

$$\mathbb{J}(\delta) = \left\{ Y \in \mathbb{R}^{128\times128} : \|Y - Y^{(j)}\|_{fr} \leqslant \delta, \text{ for some } Y^{(j)} \in \mathbb{Y} \right\},$$

for a given $\delta \geqslant 0$. In this experiment, $T = Y^TY \in \mathbb{R}^{1110\times1110}$ is a rank-deficient and well-determined numerical matrix, where $\text{rank}(T) = 1100$. Thus, we use Algorithm 1 to estimate $Y^\dagger$, with $\text{tol} = \texttt{eps}$.

Table 5: Computational time, speedup, percent difference, and error for computing $Y^\dagger$.

Method	Time (s)	Speed	Percent difference	$E_0 = \|Z - Y^\dagger\|_{fr}^2$
`pinv`	1.2479	—	—	—
Alg. 1	**0.64186**	1.9443	48.567	2.3639×10^{-17}

Table 6: Relative errors of estimation $\widetilde{X}$ in simulations presented in Figures 3 (c) and 3 (d).

Method	Simulation 1	Simulation 2	Simulation 3	Simulation 4
`pinv`	1.11895×10^{-1}	1.1045×10^{-1}	9.7809×10^{-2}	9.1159×10^{-2}
Alg. 1	1.11895×10^{-1}	1.1045×10^{-1}	9.7809×10^{-2}	9.1159×10^{-2}

Table 5 presents the execution time, speedup, percent difference, and error associated with MATLAB's `pinv` command and Algorithm 1 for computing $Y^\dagger$. Error formula is given by $E_0 = \|Z - Y^\dagger\|_{fr}^2$, where Z is the approximation of $Y^\dagger$ given by Algorithm 1. Figures 3 (c) and 3 (d) show the estimates of four noisy images in Figure 3 (b) using both methods. Further, Table 6 presents the relative error of estimation $\widetilde{X}$, where the relative error formula is given by $E_r = \|\overline{X} - \widetilde{X}\|_{fr}/\|\overline{X}\|_{fr}$.

Results obtained in Figure 3 and Tables 5 and 6 demonstrate the advantages of the proposed method. Table 5 shows that Algorithm 1 estimates $Y^\dagger$ faster than that MATLAB's `pinv` command. Moreover, it follows from Table 6 and Figures 3 (c) and 3 (c) that both methods have the same accuracy in obtaining $\widetilde{X}$.

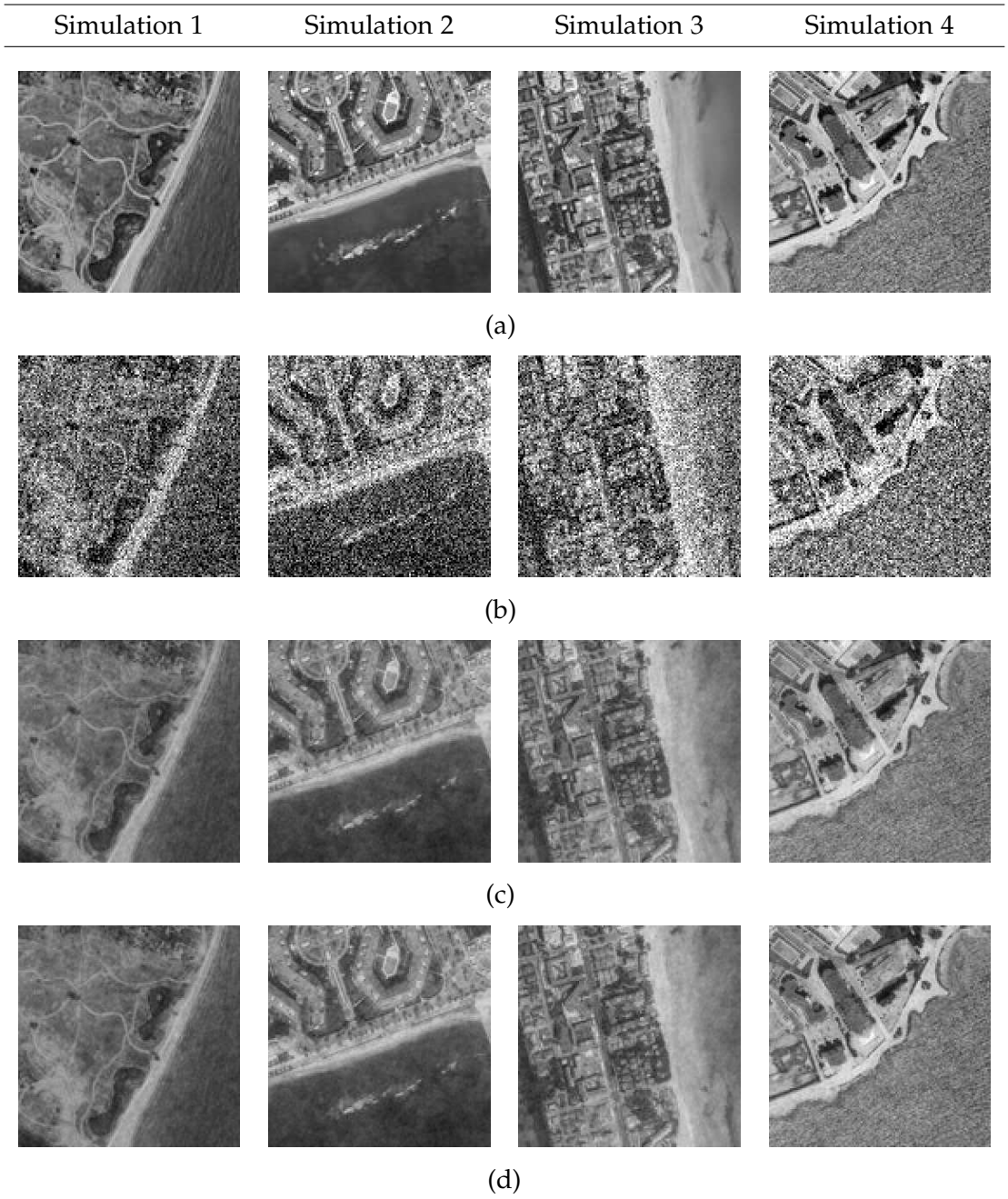


Figure 3: Illustration of the estimation of four noisy images by MATLAB's pinv command and Algorithm 1. (a) Source image $\overline{X}$. (b) Noisy observed image $\overline{Y}$. (c) Estimation $\widetilde{X}$ using MATLAB's pinv command. (d) Estimation $\widetilde{X}$ using Algorithm 1.

7.4. Numerical experiment 4: Algorithm 1 vs. parallel implementation for solving linear equation systems

In this numerical experiment, we compare the efficiency of Algorithm 1, using sequential implementation, with a parallel implementation using distributed arrays for solving linear equation systems in MATLAB. We based this experiment on the computational implementation developed in [41]. The numerical example in [41] shows how to solve a system of linear equations of the form $Ax = b$ in parallel with command `mldivide` using distributed arrays. It follows from [41] that distributed arrays distribute data from the client workspace to a parallel pool in a local machine or a cluster. Each worker stores a portion of the array in its memory, but can also communicate with the other workers to access all segments of the array.

We consider $Ax = b$, where $A \in \mathbb{R}^{m\times 10000}$ and $b \in \mathbb{R}^m$ such that $b = \sum_{i=1}^{n} A(:,i)$, i.e., solution

of $Ax = b$ is given by $\widehat{x} = (1,\ 1, \ldots, 1)^T \in \mathbb{R}^{10000}$. Here, $m = \{50, 100, 200, 300, 600, 900, \ldots, 5700, 6000\}$, and $\operatorname{rank}(A) = \frac{m}{2}$, where $T = AA^T \in \mathbb{R}^{m\times m}$ is a rank-deficiented and well-determined numerical matrix. Moreover, Algorithm 1 uses machine precision as tolerance, i.e., $\text{tol} = 2^{-52}$, and for parallel implementation, we use 10 computational threads.

Figure 4 presents the diagram of execution time associated with Algorithm 1 and parallel implementation for solving linear equation system $Ax = b$. It follows from Figure 4 that the proposed method estimates the solution of the linear system in less time than the parallel implementation when $m \leqslant 2400$. However, if $m > 2400$, the parallel implementation using distributed arrays with 10 computational threads is faster than Algorithm 1. These results follow because when the dimension of data increases, the parallel implementation using distributed arrays becomes faster than the serial method in Algorithm 1 [40, 41].

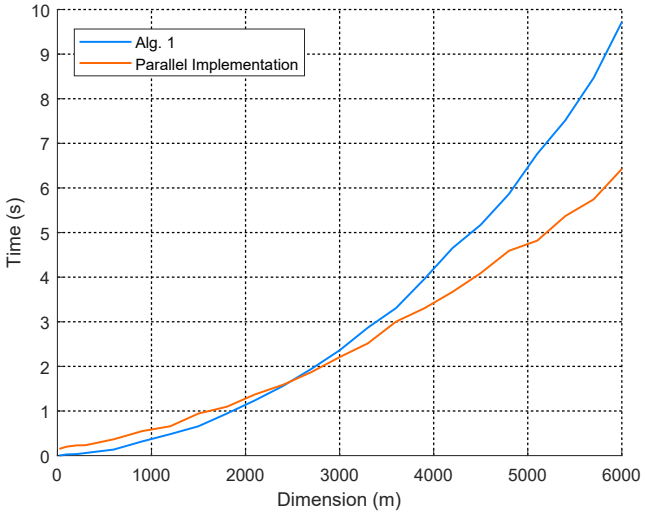


Figure 4: Diagrams of Algorithm 1 and parallel implementation using distributed arrays for solving linear equation system $Ax = b$. The matrix dimension m versus execution time (in seconds) between both methods to estimate solution of $Ax = b$.

7.5. Numerical experiment 5: Algorithm 1 using an iterative method to estimate singular values of A

In this experiment, we numerically illustrate the behavior of Remark 6.4 and Corollary 6.1, which pertain to using an iterative method to approximate the singular values of A. We consider the matrix

$$A = \begin{pmatrix} -1 & 1 & -1 & -2 \\ 0 & 0 & 0 & 4 \\ 2 & -2 & 2 & 0 \\ 0 & 0 & 0 & -2 \\ 1 & -1 & 1 & 0 \end{pmatrix},$$

where $\operatorname{rank}(A) = 2$. Moreover, $\sigma_1(A) = \sqrt{21+\sqrt{21}}$ and $\sigma_2(A) = \sqrt{21-\sqrt{21}}$. Let $\widehat{\sigma}_1(A) = 4.4159$ and $\widehat{\sigma}_2(A) = 3.0465$ be the approximations of $\sigma_1(A)$ and $\sigma_2(A)$, respectively, using three iterations of the QR algorithm to estimate the eigenvalues of $T = A^TA$. Note that $|\sigma_1(A) - \widehat{\sigma}_1(A)| = 0.6420$ and $|\sigma_2(A) - \widehat{\sigma}_2(A)| = 1.0053$. However, inequality (6.4) from Corollary 6.1 holds true. Therefore, if we use the value

$$\widehat{\alpha} = \frac{1}{2}\sqrt{\frac{\text{tol}}{\sum_{i=1}^{r}\frac{1}{\widehat{\sigma}_i^6(A)}}} \approx 1.3432 \times 10^{-4}$$

with $\text{tol} = 10^{-10}$, we obtain $\|A^\dagger - A_p(\widehat{\alpha})\|_{fr}^2 = 5.1549 \times 10^{-12} < \text{tol}$. Thus, Corollary 6.1 is numerically illustrated by this experiment.

7.6. Comments on numerical experiments

This section presents some comments on the numerical simulations developed previously.

- Tables 2 and 4 in numerical experiment 1 reveal that the execution times of the methods `qrginv`, `imqrginv`, and `ats2` is greater than those of MATLAB's `pinv` command. Additionally, the results presented in Tables 1, 2, and 3 in numerical experiments 1 and 2 suggest that the method `geniv` is the slowest among the approaches considered in this paper for estimating the Moore-Pensore inverse. Importantly, these findings do not conflict with the conclusions drawn in previous scientific papers [5, 18, 34, 58], wherein it was established that these methods exhibit faster performance compared to MATLAB's `pinv` command. It is noteworthy that the numerical results in the cited papers were confined to square matrices with a maximum dimension of $m = 4096$ and $n = 4096$. In contrast, the results in Tables 2, 3, and 4 are derived from both rectangular and square matrices, where either m or n exceeds the value of 4096.

- The findings of numerical experiment 2 in Section 7.2 indicate that Algorithm 1 significantly reduces its execution time when all positive singular values are precomputed. Various expeditious techniques from the literature are available for estimating singular values of structured matrices. Notably, there exist methods for centrosymmetric matrices [67], pentadiagonal symmetric Toeplitz matrices [22], tridiagonal symmetric Toeplitz matrices [27], Sylvester–Kac matrices [9], k-tridiagonal matrices [59], and Hankel matrices [66]. Therefore, by employing the aforementioned approaches for singular value computation, Algorithm 1 is capable of efficiently calculating the Moore-Penrose inverse of these structured matrices, thereby further enhancing its computational speed.

- Numerical experiment 3 considers the MATLAB's `pinv` command and Algorithm 1 to estimate $Y^{\dagger}$. We do not consider the methods `geniv`, `qrginv`, `imqrginv`, and `ats2` because the aim of this experiment is to measure the accuracy of Algorithm 1. It is evident from numerical experiments 1 and 2 that Algorithm 1 is faster than the other methods. However, the accuracy was not the same. Therefore, we only consider MATLAB's `pinv` command and Algorithm 1 to illustrate that Algorithm 1 is as accurate as MATLAB's `pinv` in a real-life application.

- Numerical experiment 4 compares time execution of the sequential implementation in Algorithm 1 with a parallel implementation using distributed arrays for solving linear equation systems in MATLAB. In most cases, sequential implementation of a method is faster than parallel implementation. However, this experiment shows that the proposed Algorithm 1 is faster than a parallel implementation for solving a linear system $Ax = b$, for some particular cases. This result shows the efficiency and speed of Algorithm 1.

- In each numerical experiment presented in Section 7, we set tol $=$ `eps` in Algorithm 1. Additionally, according to Theorem 4.2, it is evident that $\|A^{\dagger} - A_p(\alpha)\|_{fr}^2 \leqslant$ tol, irrespective of the specific value assigned to tol. However, when we opt for tol $<$ `eps` in our numerical simulations, we observe that $\|A^{\dagger} - A_p(\alpha)\|_{fr}^2 \not\leqslant$ tol. This discrepancy arises from a floating-point error inherent in MATLAB computations, attributed to selecting a tolerance smaller than the machine precision. Consequently, we advocate for the choice of tol $\geqslant$ `eps` in Algorithm 1 during numerical simulations to mitigate the impact of floating-point errors and ensure the reliability of the results.

8. Conclusions and future work

This paper introduced a novel algorithm for estimating the Moore-Penrose inverse of a matrix A through Tikhonov regularization. Our approach assumed that the Gram matrix of A, as defined in (2.1), is a well-determined numerical rank matrix. The proposed method involved computing all positive singular values of matrix A. Additionally, we present an efficient and accurate technique for estimating these singular values. Furthermore, we demonstrate the application of our method in solving linear

discrete well-posed problems. Numerical simulations reveal that our technique exhibits lower execution times compared to existing methods in the literature. Notably, our proposed method is 25% to 95% faster than MATLAB's `pinv` command.

Moreover, we validate the efficiency and accuracy of our approach through a real-life application in image denoising, utilizing training data. As part of our future work, we plan to explore the computational implementation of Algorithm 1 in C++ using Armadillo [55] and Boost Multiprecision [38] libraries. This exploration aims to select a tolerance level below machine precision, thereby enhancing the computational accuracy of the method presented in this paper.

Acknowledgements

The author thanks Professor Oscar Rojo from *Universidad Católica del Norte*, Antofagasta, Chile, for his help in numerical experiment 2, and the reviewers of the paper for their insightful comments. This work was financially supported by *Vicerrectoría de Investigación y Extensión* from *Instituto Tecnológico de Costa Rica* (Research #1440042).

References

[1] Z. Allen-Zhu, Y. Li, *LazySVD: Even faster SVD decomposition yet without agonizing pain*, Adv. Neural Inf. Process Syst., **29** (2016), 9 pages. 1

[2] P. Arbenz, D. Kressner, D. Zürich, *Lecture notes on solving large scale eigenvalue problems*, D-MATH, EHT Zur., **2** (2012), 256 pages. 6, 6

[3] Armadillo, *Armadillo reference documentation of command* `rank`, [Accessed 25 January 2024]. 3

[4] S. Artidiello, A. Cordero, J. R. Torregrosa, M. P. Vassileva, *Generalized inverses estimations by means of iterative methods with memory*, Mathematics, **8** (2020), 13 pages. 1, 6.1

[5] A. Ataei, *Improved Qrginv algorithm for computing Moore-Penrose inverse matrices*, ISRN Appl. Math., **2014** (2014), 5 pages. 1, 7.1, 7.6

[6] J. C. A. Barata, M. S. Hussein, *The Moore–Penrose pseudoinverse: A tutorial review of the theory*, Braz. J. Phys., **42** (2012), 146–165. 1, 1

[7] A. Ben-Israel, *An iterative method for computing the generalized inverse of an arbitrary matrix*, Math. Comput., **19** (1965), 452–455. 1, 6.2, 7.1

[8] M. W. Benson, P. O. Frederickson, *Fast parallel algorithms for the Moore-Penrose pseudo-inverse*, SIAM, Philadelphia, PA, (1987). 1

[9] T. Boros, P. Rózsa, *An explicit formula for singular values of the Sylvester-Kac matrix*, Linear Algebra Appl., **421** (2007), 407–416. 7.6

[10] M. Brand, *Fast low-rank modifications of the thin singular value decomposition*, Linear Algebra Appl., **415** (2006), 20–30. 1

[11] D. Calvetti, L. Reichel, *Tikhonov regularization of large linear problems*, BIT, **43** (2003), 263–283. 5

[12] D. Calvetti, L. Reichel, *Tikhonov regularization with a solution constraint*, SIAM J. Sci. Comput., **26** (2004), 224–239. 5

[13] J. Chavarría-Molina, J. J. Fallas-Monge, P. Soto-Quiros, *Effective implementation to reduce execution time of a low-rank matrix approximation problem*, J. Comput. Appl. Math., **401** (2022), 17 pages. 1, 7.3

[14] H. Chen, Y. Wang, *A family of higher-order convergent iterative methods for computing the Moore-Penrose inverse*, Appl. Math. Comput., **218** (2011), 4012–4016. 1

[15] J. Chung, M. Chung, *Computing optimal low-rank matrix approximations for image processing*, In: 2013 Asilomar Conference on Signals, Systems and Computers, IEEE, (2013), 670–674. 7.3

[16] J. Chung, M. Chung, *An efficient approach for computing optimal low-rank regularized inverse matrices*, Inverse Probl., **30** (2014), 19 pages. 7.3

[17] A. Cordero, P. Soto-Quiros, J. R. Torregrosa, *A general class of arbitrary order iterative methods for computing generalized inverses*, Appl. Math. Comput., **409** (2021), 18 pages. 1, 6.1

[18] P. Courrieu, *Fast computation of Moore-Penrose inverse matrices*, Neural Inform. Process. Lett. Rev., **8** (2005), 25–29. 1, 7.1, 7.6

[19] H. Cragon, *Computer architecture and implementation*, Cambridge University Press, (2000). 7

[20] B. N. Datta, *Numerical linear algebra and applications*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, (2010). 6.1

[21] P. Drineas, M. Mahoney, *Approximating a Gram matrix for improved kernel-based learning (extended abstract)*, In: Lecture Notes in Comput. Sci., Springer, Berlin, **3559** (2005), 323–337. 6.3

[22] M. Elouafi, *An eigenvalue localization theorem for pentadiagonal symmetric Toeplitz matrices*, Linear Algebra Appl., **435** (2011), 2986–2998. 7.6

[23] A.-J. Gallego, A. Pertusa, P. Gil, *Automatic ship classification from optical aerial images with convolutional neural networks*, Remote Sens., **10** (2018), 20 pages. 7.3

[24] G. H. Golub, M. Heath, G. Wahba, *Generalized cross-validation as a method for choosing a good ridge parameter*, Technometrics, **21** (1979), 215–223. 5

[25] T. N. E. Greville, *Some applications of the pseudoinverse of a matrix*, SIAM Rev., **2** (1960), 15–22. 1

[26] P. Guo, D. Zhao, M. Han, S. Feng, *Pseudoinverse learners: New trend and applications to big data*, In: INNS big data and deep learning conference, Springer, Cham, (2019), 158–168. 1

[27] J. Gutiérrez-Gutiérrez, *Eigenvalue decomposition for persymmetric Hankel matrices with at most three non-zero anti-diagonals*, Appl. Math. Comput., **234** (2014), 333–338. 7.6

[28] P. C. Hansen, *The truncated SVD as a method for regularization*, BIT, **27** (1987), 534–553. 1, 3

[29] N. J. Higham, *Cholesky factorization*, WIREs Comput. Stat., **1** (2009), 251–254. 6

[30] J. T. Holodnak, I. C. F. Ipsen, *Randomized approximation of the Gram matrix: exact computation and probabilistic bounds*, SIAM J. Matrix Anal. Appl., **36** (2015), 110–137. 6.3

[31] R. Hunger, *Floating point operations in matrix-vector calculus*, Tech. Univ. München, (2007). 6

[32] Julia, *Julia reference documentation of command* `rank`, [Accessed 25 January 2024]. 3

[33] V. N. Katsikis, D. Pappas, *Fast computing of the Moore-Penrose inverse matrix*, Electron. J. Linear Algebra, **17** (2008), 637–650. 1

[34] V. N. Katsikis, D. Pappas, A. Petralias, *An improved method for the computation of the Moore–Penrose inverse matrix*, Appl. Math. Comput., **217** (2011), 9828–9834. 1, 6.1, 7.1, 7.6

[35] S. López-Tapia, A. Lucas, R. Molina, A. K. Katsaggelos, *A single video super-resolution gan for multiple downsampling operators based on pseudo-inverse image formation models*, Digit. Signal Process., **104** (2020), 12 pages. 1

[36] S. Lu, X. Wang, G. Zhang, X. Zhou, *Effective algorithms of the Moore-Penrose inverse matrices for extreme learning machine*, Intell. Data Anal., **19** (2015), 743–760. 1

[37] T. Lyche, *Numerical linear algebra and matrix factorizations*, Springer, Cham, (2020). 1, 2.1

[38] J. Maddock, C. Kormanyos, *Boost multiprecision*, [Accessed 25 January 2024]. 8

[39] O. Makkonen, C. Hollanti, *Secure distributed gram matrix multiplication*, In: 2023 IEEE Information Theory Workshop (ITW), (2023), 192–197. 6.3

[40] MathWorks, *Distributing arrays to parallel workers*, [Accessed 25 January 2024]. 7.4

[41] MathWorks, *Use distributed arrays to solve systems of linear equations with direct methods*, [Accessed 25 January 2024]. 7.4

[42] MathWorks, *Matlab reference documentation of command* `rank`, [Accessed 25 January 2024]. 3

[43] MathWorks, *Matlab reference documentation of command* `mtimes`, [Accessed 25 January 2024]. 6.3

[44] MathWorks, *Matlab reference documentation of command* `isillconditioned`, [Accessed 25 January 2024]. 6

[45] S. Miljković, M. Miladinović, P. Stanimirović, I. Stojanović, *Application of the pseudoinverse computation in reconstruction of blurred images*, Filomat, **26** (2012), 453–465. 1

[46] NumPy, *Numpy reference documentation of command* `rank`, [Accessed 25 January 2024]. 3

[47] Octave, *Octave reference documentation of command* `rank`, [Accessed 25 January 2024]. 3

[48] M. Ostilli, *Cayley trees and Bethe lattices: A concise analysis for mathematicians and physicists*, Phys. A, **391** (2012), 3417–3423. 7.2

[49] R. Penrose, *On best approximate solutions of linear matrix equations*, Math. Proc. Camb. Philos. Soc., **52** (1956), 17–19. 7.3

[50] M. D. Petković, *Generalized schultz iterative methods for the computation of outer inverses*, Comput. Math. Appl., **67** (2014), 1837–1847. 6.2, 7.1

[51] M. D. Petković, M. A. Krstić, K. P. Rajković, *Rapid generalized schultz iterative methods for the computation of outer inverses*, J. Comput. Appl. Math., **344** (2018), 572–584.

[52] M. D. Petković, M. S. Petković, *Hyper-power methods for the computation of outer inverses*, J. Comput. Appl. Math., **278** (2015), 110–118. 6.2, 7.1

[53] V. Rajaraman, *IEEE standard for floating point numbers*, Resonance, **21** (2016), 11–30. 3

[54] O. Rojo, M. Robbiano, *An explicit formula for eigenvalues of Bethe trees and upper bounds on the largest eigenvalue of any tree*, Lin. Algebra Appl., **427** (2007), 138–150. 7.2

[55] C. Sanderson, R. Curtin, *Armadillo: a template-based C++ library for linear algebra*, J. Open Source Softw., **1** (2016), 2 pages. 8

[56] Scilab, *Scilab reference documentation of command* `rank`, [Accessed 25 January 2024]. 3

[57] P. Soto-Quiros, J. Jose Fallas-Monge, J. Chavarría-Molina, *A fast algorithm for image deconvolution based on a rank constrained inverse matrix approximation problem*, In: Proceedings of Sixth International Congress on Information and Communication Technology, Springer, Singapore, (2022), 165–176. 1, 7.3

[58] P. S. Stanimirović, D. Pappas, V. N. Katsikis, I. P. Stanimirović, *Full-rank representations of outer inverses based on the QR decomposition*, Appl. Math. Comput., **218** (2012), 10321–10333. 1, 7.1, 7.6

[59] A. Tănăsescu, P. G. Popescu, *A fast singular value decomposition algorithm of general* k*-tridiagonal matrices*, J. Comput. Sci., **31** (2019), 1–5. 7.6

[60] B. Telfer, D. Casasent, *Fast method for updating robust pseudoinverse and Ho-Kashyap associative processors*, IEEE Trans. Syst. Man. Cybern., **24** (1994), 1387–1390. 1

[61] A. Torokhti, P. Soto-Quiros, *Generalized Brillinger-like transforms*, IEEE Signal Process. Lett., **23** (2016), 843–847. 1
[62] G. Wang, Y. Wei, S. Qiao, *Generalized inverses: Theory and computations*, Springer, Singapore; Science Press Beijing, Beijing, (2018). 1
[63] J. Wang, P. Guo, X. Xin, *Review of pseudoinverse learning algorithm for multilayer neural networks and applications*, In: Advances in Neural Networks – ISNN 2018, Lecture Notes in Computer Science, Springer, Cham (2018), 99–106. 1
[64] R. E. White, *Computational Linear Algebra: with Applications and MATLAB® Computations*, Chapman & Hall/CRC, New York, (2023). 6.3
[65] L. Wu, *Regularization methods and algorithms for least squares and Kronecker product least squares problems*, ProQuest LLC, Ann Arbor, MI, (1997). 1, 3
[66] W. Xu, S. Qiao, *A fast symmetric SVD algorithm for square Hankel matrices*, Linear Algebra Appl., **428** (2008), 550–563. 7.6
[67] C. Xu, K. Wang, Z. Liu, *Singular value decomposition for central extended matrix*, In: 2011 International Conference on Computer Science and Service System (CSSS), IEEE, (2011), 736–738. 7.6
[68] G. Zielke, *Report on test matrices for generalized inverses*, Computing, **36** (1986), 105–162. 2.2
[69] D. D. Zontini, M. L. Mirkoski, J. A. Santos, *Error bounds in the computation of outer inverses with generalized Schultz iterative methods and its use in computing of Moore-Penrose inverse*, Appl. Math. Comput., **440** (2023), 10 pages. 6.2, 7.1