2008

Fast computing of the Moore-Penrose inverse matrix

Vasilios N. Katsikis
Dimitrios Pappas
dpappas@aeub.gr

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

Recommended Citation
DOI: https://doi.org/10.13001/1081-3810.1287

This Article is brought to you for free and open access by Wyoming Scholars Repository. It has been accepted for inclusion in Electronic Journal of Linear Algebra by an authorized editor of Wyoming Scholars Repository. For more information, please contact scholcom@uwyo.edu.
FAST COMPUTING OF THE MOORE-PENROSE INVERSE MATRIX∗

VASILIOS N. KATSIKIS† AND DIMITRIOS PAPPAS‡

Abstract. In this article a fast computational method is provided in order to calculate the Moore-Penrose inverse of full rank $m \times n$ matrices and of square matrices with at least one zero row or column. Sufficient conditions are also given for special type products of square matrices so that the reverse order law for the Moore-Penrose inverse is satisfied.

Key words. Moore-Penrose inverse matrix, Computational methods, Tensor-product matrix, Reverse order law.

AMS subject classifications. 15A09.

1. Introduction. Let $T$ be a $n \times n$ real matrix. It is known that when $T$ is singular, then its unique generalized inverse $T^\dagger$ (known as the Moore-Penrose inverse) is defined. In the case when $T$ is a real $m \times n$ matrix, Penrose showed that there is a unique matrix satisfying the four Penrose equations, called the generalized inverse of $T$. A lot of work concerning generalized inverses has been carried out, in finite and infinite dimension (e.g., [2, 11]).

In this article, we provide a method for the fast computation of the generalized inverse of full rank matrices and of square matrices with at least one zero row or column. In order to reach our goal, we use a special type of tensor product of two vectors, that is usually used in infinite dimensional Hilbert spaces. Using this type of tensor product, we also give sufficient conditions for products of square matrices so that the reverse order law for the Moore-Penrose inverse ([1, 4, 5]) is satisfied.

There are several methods for computing the Moore-Penrose inverse matrix (cf. [2]). One of the most commonly used methods is the Singular Value Decomposition (SVD) method. This method is very accurate but also time-intensive since it requires a large amount of computational resources, especially in the case of large matrices. In the recent work of P. Courrieu [3], an algorithm for fast computation of Moore-Penrose inverse matrices is presented based on a known reverse order law (eq. 3.2

∗Received by the editors May 26, 2008. Accepted for publication November 20, 2008. Handling Editor: Michael Tsatsomeros.
†Department of Mathematics, National Technical University of Athens, Zographou 15780, Athens, Greece (vaskats@mail.ntua.gr, vaskats@gmail.com). Supported during his postdoctoral studies by the State Scholarship Foundation (IKY).
‡Athens University of Economics and Business, Department of Statistics, 76 Patission St., GR10434, Athens, Greece (dpappas@aeub.gr).
from [11]), and on a full-rank Cholesky factorization of possibly singular symmetric positive matrices. This is a fast algorithmic process; however, even in the case of rank deficient matrices, which is the proposed case by the author for the application of this method, the computation error is large compared to that of the SVD method (see Table 4.2).

In the present manuscript, we construct a very fast and reliable method (see the \texttt{ginv} function in the Appendix) in order to estimate the Moore-Penrose inverse matrix of a rank-$n$ tensor-product matrix. The computational effort required for the \texttt{ginv} function (see Figure 4.1) in order to obtain the generalized inverse is substantially lower, particularly for large matrices, compared to those provided by the other two methods (the SVD method and Courrieu’s method). In addition, we obtain reliable and very accurate approximations in each one of the tested cases (Table 4.2). Also, from Theorem 3.1, that will be shown in paragraph 3, it is evident that the proposed method (\texttt{ginv} function), can also be used in the case of full-rank rectangular matrices.

In what follows, we make use of the high-level language Matlab both for calculations of the generalized inverse of a tensor-product matrix, as well as for testing the reliability of the obtained results. Specifically, the Matlab 7.4 (R2007a) [9, 10] Service Pack 3 version of the software was used on an AMD Athlon(tm) 64 Processor 3000+ system running at 1.81GHz with 1.5 GB of RAM memory using the Windows XP Professional Version 2002 Service Pack 2 Operating System.

2. Preliminaries and notation. We shall denote by $\mathbb{R}^{m \times n}$ the linear space of all $m \times n$ real matrices. For $T \in \mathbb{R}^{m \times n}$, $R(T)$ will denote the range of $T$ and $N(T)$ the kernel of $T$. The generalized inverse $T^\dagger$ (known as the Moore-Penrose inverse) is the unique matrix that satisfies the following four Penrose equations:

$$TT^\dagger = (TT^\dagger)^*, \quad T^\dagger T = (T^\dagger T)^*, \quad TT^\dagger T = T, \quad T^\dagger TT^\dagger = T^\dagger,$$

where $T^*$ denotes the transpose matrix of $T$.

It is easy to see that $R(T^\dagger) = N(T)^\perp$, where $TT^\dagger$ is the orthogonal projection onto $R(T)$, and that $T^\dagger T$ is the orthogonal projection onto $N(T)^\perp$. It is well known that $R(T^\dagger) = R(T^*)$. The number $r = \dim R(T)$ is called the rank of $T$ and shall be denoted by $r(T)$ and $\langle , \rangle$ denotes the usual inner-product in $\mathbb{R}^n$.

3. The generalized inverse of a tensor-product matrix. According to [12], for each $x \in \mathbb{R}^k$, we consider the mapping

$$e \otimes f : \mathbb{R}^k \rightarrow \mathbb{R}^k \text{ with } (e \otimes f)(x) = \langle x, e \rangle f,$$

and assume that $\{e_1, ..., e_n\}$ and $\{f_1, ..., f_n\}$ are two collections of orthonormal vectors and linearly independent vectors of $\mathbb{R}^k$, $n < k$, respectively. Then, every rank-$n$
operator $T$ can be written in the following form

$$T = \sum_{i=1}^{n} e_i \otimes f_i.$$  

In the following, we shall refer to this type of tensor product as the tensor-product of the collections $\{e_1, ..., e_n\}$ and $\{f_1, ..., f_n\}$. The adjoint operator $T^*$ of $T$ is the rank-$n$ operator $T^* = \sum_{i=1}^{n} f_i \otimes e_i$.

The tensor-product of two collections of vectors, as defined above, is a linear operator. Therefore, it has a corresponding matrix representation $T$. In order to describe this representation, let us denote by $\{e_1, ..., e_n\}$ the first $n$ vectors of the standard basis of $\mathbb{R}^k$, and suppose that the $f_i$’s are in the form :

$$f_i = (f_{i1}, f_{i2}, ..., f_{ik}), \quad i = 1, 2, ..., n$$

then the corresponding matrix $T$ has the vectors $f_i$ as columns. So,

$$T = \begin{pmatrix}
  f_{11} & f_{21} & ... & f_{n1} & 0 & ... & 0 \\
  f_{12} & f_{22} & ... & f_{n2} & 0 & ... & 0 \\
  \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
  f_{1k} & f_{2k} & ... & f_{nk} & 0 & ... & 0
\end{pmatrix}.$$  

We shall refer to this matrix $T$ as the tensor-product matrix of the given collections.

In order to compute the Moore-Penrose inverse of the corresponding tensor-product matrix, we use the following theorem. For the sake of completeness we give a sketch of its proof.

**Theorem 3.1.** [8, Theorem 3.2] Let $\mathcal{H}$ be a Hilbert space. If $T = \sum_{i=1}^{n} e_i \otimes f_i$ is a rank-$n$ operator then its generalized inverse is also a rank-$n$ operator and for each $x \in \mathcal{H}$, it is defined by the relation

$$T^\dagger x = \sum_{i=1}^{n} \lambda_i(x) e_i,$$

where the functions $\lambda_i$ are the solution of an appropriately defined $n \times n$ linear system.

**Proof.** If $T = \sum_{i=1}^{n} e_i \otimes f_i$ then $\mathcal{R}(T) = [f_1, f_2, ..., f_n]^2$ and $\mathcal{R}(T^\dagger) = \mathcal{R}(T^*) = [e_1, e_2, ..., e_n]$. Therefore, for each $x \in \mathcal{H}$, we have $T^\dagger x = \sum_{i=1}^{n} \lambda_i(x) e_i$. Hence, in order to determine $T^\dagger$ one must calculate the functions $\lambda_i, \quad i = 1, 2, ..., n$. It holds that $T^* x = T^* T T^\dagger x$, thus we have

$^2[f_1, ..., f_n]$ denotes the closed linear span generated by the vectors $f_1, ..., f_n$. 

\[
\sum_{i=1}^{n} \langle x, f_i \rangle e_i = T^* x = T^* T^\dagger x = \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_j(x) \langle f_i, f_j \rangle e_i.
\]

The last relation leads to the following \( n \times n \) linear system:

\[
\langle x, f_i \rangle = \sum_{j=1}^{n} \lambda_j(x) \langle f_i, f_j \rangle, \quad i = 1, 2, \ldots, n.
\]

The determinant of the above system is the Gram determinant of the linearly independent vectors \( f_1, \ldots, f_n \) and hence, for each \( x \in \mathcal{H} \), it has a unique solution where the unknowns are the functions \( \lambda_i, i = 1, 2, \ldots, n \). \( \square \)

Hence, one has to derive a procedure that will accurately implement the ideas of Theorem 3.1 in order to determine the generalized inverse of a tensor-product matrix. In other words, our main concern is to calculate the corresponding \( \lambda_i \) in the expansion

\[
T^\dagger x = \sum_{i=1}^{n} \lambda_i(x) e_i
\]

so that we can provide the generalized inverse \( T^\dagger \). In order to reach our goal, the high-level computational environment of Matlab is employed in our study.

4. The computational method.

4.1. Method presentation and examples. The first step of our approach consists of constructing two functions named \texttt{ginv} and \texttt{ginvetest} (see Appendix). The function \texttt{ginv} first calculates the corresponding Gram matrix of the linearly independent vectors \( f_1, \ldots, f_n \) and then it solves the appropriately defined \( n \times n \) linear system. In particular, for each \( j = 1, \ldots, n \), the \texttt{ginv} function provides the corresponding \( \lambda_i(e_j) \) (see Theorem 3.1) in the expansion

\[
T^\dagger e_j = \sum_{i=1}^{n} \lambda_i(e_j) e_i
\]

in order to determine the generalized inverse of a given tensor-product matrix \( T \). Therefore, for each \( j = 1, 2, \ldots, n \) we have \( \lambda_1(e_j), \lambda_2(e_j), \ldots, \lambda_n(e_j) \) as a solution of the corresponding linear system. Then, from (4.1) the generalized inverse \( T^\dagger \) is having the following form

\[\text{That is, a } n \times n \text{ matrix } [a_{ij}], \text{ where } a_{ij} = \langle f_i, f_j \rangle, \text{ for each } i, j = 1, \ldots, n.\]
Fast Computing of the Moore-Penrose Inverse Matrix

\[
T^\dagger = \begin{pmatrix}
\lambda_1(e_1) & \lambda_1(e_2) & \ldots & \lambda_1(e_k) \\
\lambda_2(e_1) & \lambda_2(e_2) & \ldots & \lambda_2(e_k) \\
\vdots & \vdots & \ddots & \vdots \\
\lambda_n(e_1) & \lambda_n(e_2) & \ldots & \lambda_n(e_k) \\
0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 \\
\end{pmatrix}.
\]

Suppose that \(T\) is the corresponding matrix representation of a rank-\(n\) operator, then \(T\) is a \(k \times k\) matrix whose first \(n\) columns are linearly independent vectors of \(\mathbb{R}^k, n < k\) and all the other columns consists of zeros.

**Remark 4.1.** It is clear, from Theorem 3.1, that our computational method will restrict in the case of full-rank rectangular and rank-deficient square matrices.

The `ginvtest` function allows us to compare the accuracy of the proposed method (`ginv` function) to that of the SVD method (Matlab `pinv` function) and on a recent, fast computational method by P. Courrieu [3] (`geninv` function). The `geninv` function is based on a known reverse order law (eq. 3.2 from [11]) and on a full-rank Cholesky factorization of possibly singular symmetric positive matrices. The accuracy of the results was examined in error matrices, with the matrix 2-norm, corresponding to the four properties characterizing the Moore-Penrose inverse (i.e., \(TT^\dagger = (TT^\dagger)^*, T^\dagger T = (T^\dagger T)^*, TT^\dagger T = T, T^\dagger TT^\dagger = T^\dagger\)). The results of the `ginvtest` function are organized in a 3 \(\times\) 4 matrix. The first row of this matrix contains the error results for the SVD method, the second row contains the error results for the Courrieu's method and the third row contains the error results of the proposed method.

The computational effort required to obtain the generalized inverse of a tensor-product matrix under different parameter configurations (i.e., number of vectors and dimensions) of the `ginv` function is substantially less. This is particularly true for large matrices, when a comparison is made between the results provided by the proposed method and those provided by the other two methods (SVD method, Courrieu's method). In particular, we have used the Matlab function `rand` in order to produce \(m \times n\) matrices of values derived by a pseudorandom, scalar value drawn from a uniform distribution in the unit interval. Functions `ginv` and `ginvtest` (see Appendix) must be stored for further use. In our work, we stored them in a Matlab-accessible directory named `work`. Note that, `ginv` and `ginvtest` define functions that accept an input \(A\), where \(A\) denotes a full rank matrix, in a different case the program responds with an explicit warning. We illustrate an example for a rank-8 tensor-product matrix, where \(e_i, f_i \in \mathbb{R}^9, i = 1, 2, \ldots, 9\). For the purpose of monitoring the performance, we also present the execution times as well as the accuracy of the proposed method, the
SVD method and Courrieu’s method (Table 4.1). The execution times have been recorded using the Matlab function profile.

**EXAMPLE 4.2.** Consider the tensor-product matrix $T = \sum_{i=1}^{8} e_i \otimes f_i$, generated by the linearly independent vectors,

$$
\begin{pmatrix}
    f_1 \\
    f_2 \\
    f_3 \\
    f_4 \\
    f_5 \\
    f_6 \\
    f_7 \\
    f_8
\end{pmatrix} =
\begin{pmatrix}
    11983 \\
    12956 \\
    147 \\
    145 \\
    10584 \\
    0 \\
    64
\end{pmatrix}
\begin{pmatrix}
    5634 \\
    1951 \\
    133 \\
    270 \\
    130 \\
    35 \\
    419
\end{pmatrix}
\begin{pmatrix}
    4018 \\
    4361 \\
    222 \\
    230 \\
    5456 \\
    181 \\
    209
\end{pmatrix}
\begin{pmatrix}
    3791 \\
    3431 \\
    328 \\
    215 \\
    124 \\
    37 \\
    123
\end{pmatrix}
\begin{pmatrix}
    3661 \\
    2328 \\
    282 \\
    147 \\
    434 \\
    40 
\end{pmatrix}
\begin{pmatrix}
    2806 \\
    2553 \\
    237 \\
    213 \\
    143 
\end{pmatrix}
\begin{pmatrix}
    2675 \\
    1936 \\
    285 \\
    218 \\
    149 
\end{pmatrix}
\begin{pmatrix}
    3587 \\
    10214 \\
    241 \\
    234 
\end{pmatrix}
\begin{pmatrix}
    3955 \\
    8009 
\end{pmatrix}
$$

and the vectors $e_1, ..., e_8$ of the standard basis of $\mathbb{R}^8$.

Then, $T$ can be represented as a $9 \times 9$ matrix whose first 8 columns are the vectors $f_1, f_2, ..., f_8$ and the last one (column) consists of zeros.

We proceed with computing the Moore-Penrose inverse of $T$ as follows:

In the command window of Matlab we type a matrix $A$ that contains the nonzero block of $T$, i.e. $A$ has the vectors $f_i, i = 1, 2, ..., 8$ as columns. Then, we invoke the `ginv` function by typing in the command window:

```matlab
>> ginv(A)
```

The results, then, are as follows:

```
ans =

229/642 311/2358 -563/6505 -145/1541 122/4031 -242/4405 135/11093 -104/11765 443/4180
270/1654 280/3443 -95/9808 -62/21251 302/2125 -205/8221 114/18613 -505/3503 -160/11331
-60/1276 -287/1355 145/2445 131/4034 -303/5586 55/7714 683/16884 180/3313 -165/11332
284/7305 366/2720 -177/2924 -102/5061 -166/12675 -73/2879 243/9074 -107/7315 325/6527
```

therefore the Moore-Penrose inverse $T^\dagger$ of $T$ is the following matrix:

$$
T^\dagger =
\begin{pmatrix}
    -912 \\
    17887 \\
    229 \\
    6452 \\
    370 \\
    1443 \\
    10176 \\
    1042 \\
    10144 \\
\end{pmatrix}
\begin{pmatrix}
    129 \\
    1054 \\
    111 \\
    5650 \\
    226 \\
    3444 \\
    7155 \\
    3772 \\
\end{pmatrix}
\begin{pmatrix}
    332 \\
    1069 \\
    140 \\
    5505 \\
    93 \\
    980 \\
    2209 \\
    2069 \\
\end{pmatrix}
\begin{pmatrix}
    340 \\
    4663 \\
    113 \\
    1541 \\
    52 \\
    1131 \\
    4034 \\
    2069 \\
\end{pmatrix}
\begin{pmatrix}
    169 \\
    2955 \\
    122 \\
    4031 \\
    52 \\
    813 \\
    5366 \\
    2449 \\
\end{pmatrix}
\begin{pmatrix}
    217 \\
    13563 \\
    1242 \\
    4045 \\
    282 \\
    8211 \\
    36064 \\
    4969 \\
\end{pmatrix}
\begin{pmatrix}
    232 \\
    7799 \\
    143 \\
    4405 \\
    11093 \\
    11147 \\
    18664 \\
    11147 \\
\end{pmatrix}
\begin{pmatrix}
    104 \\
    1645 \\
    1061 \\
    4045 \\
    11765 \\
    2619 \\
    3201 \\
\end{pmatrix}
\begin{pmatrix}
    -375 \\
    2668 \\
    441 \\
    8101 \\
    1480 \\
    2809 \\
    11332 \\
\end{pmatrix}
\begin{pmatrix}
    0 \\
    0 \\
    0 \\
    0 \\
    0 \\
    0 \\
    0 \\
\end{pmatrix}
$$
In order to test the accuracy of the three methods (i.e., SVD method (pinv), Courrieu’s method (geninv) and the proposed method (ginv)) we invoke the `ginvtest` function by using the command:

\[ >> \text{ginvtest}(A) \]

The cumulative results (execution times, accuracy) are presented in Table 4.1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (seconds)</th>
<th>(|TT^\dagger T - T|_2)</th>
<th>(|T^\dagger TT^\dagger - T^\dagger|_2)</th>
<th>(|TT^\dagger - (TT^\dagger)^*|_2)</th>
<th>(|T^\dagger T - (T^\dagger T)^*|_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVD method (Matlab pinv)</td>
<td>0.004</td>
<td>6.7251 \times 10^{-14}</td>
<td>9.3345 \times 10^{-16}</td>
<td>1.4001 \times 10^{-14}</td>
<td>5.6939 \times 10^{-14}</td>
</tr>
<tr>
<td>Courrieu’s method (geninv)</td>
<td>0.005</td>
<td>4.6811 \times 10^{-12}</td>
<td>2.1126 \times 10^{-14}</td>
<td>2.2553 \times 10^{-12}</td>
<td>8.5285 \times 10^{-14}</td>
</tr>
<tr>
<td>Proposed method (ginv)</td>
<td>0.002</td>
<td>9.6576 \times 10^{-15}</td>
<td>1.0375 \times 10^{-14}</td>
<td>1.5797 \times 10^{-15}</td>
<td>7.872 \times 10^{-14}</td>
</tr>
</tbody>
</table>

It is apparent that the `ginv` function provides a practical, accurate and substantially faster numerical way for the calculation of the Moore-Penrose inverse of a tensor-product matrix.

**Remark 4.3.** The Moore-Penrose inverse of a rectangular matrix of size \(m \times n\) is a rectangular matrix of size \(n \times m\). This can be easily verified in example 4.2 since the given matrix is a \(9 \times 9\) matrix with a \(9 \times 8\) nonzero block and its generalized inverse is a \(9 \times 9\) matrix with a \(8 \times 9\) nonzero block.

It is clear that according to Theorem 3.1, the choice of using the proposed `ginv` function depends on the fact that the tensor-product matrix \(T\) is rank deficient\(^4\). Hence, the proposed method can also be used for the computation of any given \(m \times n\) full-rank real matrix, where \(m > n\). Also, in the case when \(m < n\), one can compute the transpose matrix \(T^*\) and then make use of the formula \((T^*)^\dagger = (T^\dagger)^*\). So, in order to simplify the procedure, the `ginv` function performs all necessary transpositions in all cases. Thus, the `ginv` function can also be used, directly, for the computation of any given \(m \times n\) full-rank real matrix. In addition, since the user, does not know, in general, a priori whether or not the argument he/she provides is rank deficient the `ginv` function provides an explicit warning in this case.

**4.2. Comparison Results.** In this section, we compare the performance of the proposed method (`ginv`) to that of the other two algorithms, namely, the SVD method (pinv) and Courrieu’s method (geninv). In addition, the accuracy of the results was examined with the matrix 2-norm in error matrices corresponding to the four properties characterizing the Moore-Penrose inverse (i.e., \(TT^\dagger = (TT^\dagger)^*\), \(T^\dagger T = \)

\(^4\)An \(n \times n\) matrix \(T\) is rank deficient if \(\text{rank}(T) = m < n\).
enables us to perform fast and accurate estimations of the generalized inverse \((T^\dagger)^*, TT^\dagger T = T, T^\dagger TT^\dagger = T^\dagger\) and the cumulative results are presented in Table 4.2.

<table>
<thead>
<tr>
<th></th>
<th>Rank</th>
<th>(|TT^\dagger T - T|_2)</th>
<th>(|TT^\dagger - T^\dagger|_2)</th>
<th>(|TT^\dagger - (TT^\dagger)^*|_2)</th>
<th>(|TT^\dagger - T^\dagger T^\dagger T|_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVD method</td>
<td>2^8</td>
<td>1.763 \times 10^{-12}</td>
<td>1.0573 \times 10^{-12}</td>
<td>7.2466 \times 10^{-13}</td>
<td>6.1911 \times 10^{-13}</td>
</tr>
<tr>
<td>(Matlab pinv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Courrieu’s method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ginv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposed method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ginv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD method</td>
<td>2^9</td>
<td>4.7474 \times 10^{-12}</td>
<td>4.7250 \times 10^{-12}</td>
<td>1.7556 \times 10^{-12}</td>
<td>1.2273 \times 10^{-12}</td>
</tr>
<tr>
<td>(Matlab pinv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Courrieu’s method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ginv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposed method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ginv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD method</td>
<td>2^{10}</td>
<td>1.5329 \times 10^{-11}</td>
<td>9.9524 \times 10^{-12}</td>
<td>4.6853 \times 10^{-12}</td>
<td>5.3752 \times 10^{-12}</td>
</tr>
<tr>
<td>(Matlab pinv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Courrieu’s method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ginv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposed method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ginv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD method</td>
<td>2^{11}</td>
<td>7.9990 \times 10^{-10}</td>
<td>6.9937 \times 10^{-9}</td>
<td>1.5855 \times 10^{-10}</td>
<td>1.2520 \times 10^{-10}</td>
</tr>
<tr>
<td>(Matlab pinv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Courrieu’s method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ginv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposed method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ginv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD method</td>
<td>2^{12}</td>
<td>1.5743 \times 10^{-5}</td>
<td>5.8952 \times 10^{-6}</td>
<td>1.8897 \times 10^{-7}</td>
<td>1.9087 \times 10^{-9}</td>
</tr>
<tr>
<td>(Matlab pinv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Courrieu’s method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ginv)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposed method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The new numerical method, based on the introduction of the \texttt{ginv} function, enables us to perform fast and accurate estimations of the generalized inverse \(T^\dagger\) of a tensor-product matrix for a variety of dimensions. The tested matrices were obtained using the Matlab function \texttt{rand} and they were all rank deficient with rank \(2^n\) for \(n = 8, 9, 10, 11, 12\). Figure 4.1 shows the time efficiency curves, i.e., the rank of the tested matrix versus the computation time (in seconds). All algorithms were carefully implemented and tested in Matlab. After a deep analysis of the results in Figure 4.1 and Table 4.2 one can easily obtain the following conclusions:

5. Conclusions.

1. The \texttt{ginv} method is sensitive to numerical rounding errors, and it has been observed that it is inaccurate in the computation of generalized inverses of full-rank ill-conditioned matrices. This remark is also included in [3].
2. It is notable that for matrices with rank $2^{12}$ or higher, the SVD method was not able to produce a numerical result (Matlab produced an 'Out of memory' message).

3. It is evident, from Table 4.2, that using the \texttt{ginv} function we obtained a reliable approximation in all the tests that were conducted. At the same time, it is also clear that we have simplified the procedure to the extent that the interested user can reach a fast computational solution using a reduced amount of computational resources. Therefore, the proposed function allows us for a both fast and accurate computation of the Moore-Penrose inverse matrix.

4. In line 9 of the \texttt{ginv} function we included a rank test in order to simplify the use of the proposed Matlab function for the interested user. It is notable that in our tests the rank test costs more than the 50\% of the computational time that the \texttt{ginv} function took to respond. Therefore, if we are in position to know more about our input data then, after a slight modification, it is clear that the \texttt{ginv} function provides a rapid method for computing generalized inverses.

6. The reverse order law. In this section, we introduce sufficient conditions so that the generalized inverse of the product of two square rank-$n$ matrices is the product of the generalized inverses of the corresponding matrices in reverse order. In general, the well known reverse order law, $(AB)^{-1} = B^{-1}A^{-1}$ which holds for operators and matrices is only known to hold for generalized inverses under certain conditions. A lot of work has been carried out with respect to conditions so that
the relationship \((AB)^\dagger = B^\dagger A^\dagger\) holds (e.g.,[1] [5], [7]). The following theorem is a restatement of a part of R. Bouldin’s theorem 3.1 [1] which is valid for operators and matrices.

**Theorem 6.1.** Let \(A, B\) be bounded operators with closed range on a Hilbert space \(\mathcal{H}\). Then \((AB)^\dagger = B^\dagger A^\dagger\) if and only if the following three conditions hold:

(i) The range of \(AB\) is closed,

(ii) \(A^\dagger A\) commutes with \(BB^*\),

(iii) \(BB^\dagger\) commutes with \(A^*A\).

Since in the case of matrices the range is always closed, the first condition of this theorem always holds.

**Proposition 6.2.** Let \(T_1 = \sum_{i=1}^{n} e_i \otimes f_i\) and \(T_2 = \sum_{i=1}^{n} e_i' \otimes f_i'\) be two tensor-product matrices. If \([f_1', ..., f_n'] = [e_1, ..., e_n]\), then \((T_1T_2)^\dagger = T_2^\dagger T_1^\dagger\).

**Proof.** It is well known that, \(R(TT^*) = R(T)\) and \(R(T^*T) = R(T^*)\). Therefore, \(R(T_2T_2^*) = R(T_2) = [f_1', ..., f_n']\) and \(R(T_1^*T_1) = R(T_1^*) = [e_1, ..., e_n]\). In order to prove condition (ii) of Theorem 6.1 we have to show that \(T_1^*T_2^*\) commutes with the matrix \(T_2T_2^*\). Indeed, since \(T_1^*\) is the projection matrix on \(R(T_1^*) = R(T_1^*) = [e_1, ..., e_n]\), this is equivalent to \([e_1, ..., e_n]\) be invariant from \(T_2T_2^*\). By the relation

\[
R(T_2T_2^*) = R(T_2) = [f_1', ..., f_n'] = [e_1, ..., e_n]
\]

it is evident that condition (ii) of Theorem 6.1 holds.

In order to prove condition (iii) of Theorem 6.1 we use similar arguments as before, i.e., we must show that \(T_2T_2^*\), which is the projection matrix on \(R(T_2) = [f_1', ..., f_n']\), commutes with \(T_1^*T_1\), or else that the subspace \([f_1', ..., f_n']\) is invariant under \(T_1^*T_1\), which holds from \(R(T_1^*T_1) = R(T_1^*) = [e_1, ..., e_n]\).

By the above proposition, it is clear that only the placement of the non-zero columns plays a role on the inverse order law. Note that the law does not depend on the actual entries of the first matrix. We justify the situation with a simple example:

**Example 6.3.** Let \(T_1\) and \(T_2\) be two tensor-product matrices of \(\mathbb{R}^{5\times 5}\), where \(T_1 = \sum_{i=1}^{3} e_i \otimes f_i\) is a rank-3 tensor-product matrix, with

\[
e_1 = (1, 0, 0, 0, 0), e_2 = (0, 1, 0, 0, 0), e_3 = (0, 0, 1, 0, 0),
\]

and

\[
f_1 = (1, 3, 5, 6, 7), f_2 = (2, 4, 3, 7, 6), f_3 = (1, 8, 7, 5, 6).
\]
Fast Computing of the Moore-Penrose Inverse Matrix

Then, \( T_1 = \begin{pmatrix} 1 & 2 & 1 & 0 & 0 \\ 3 & 4 & 8 & 0 & 0 \\ 5 & 3 & 7 & 0 & 0 \\ 6 & 7 & 5 & 0 & 0 \\ 7 & 6 & 6 & 0 & 0 \end{pmatrix} \).

Let \( T_2 = \sum_{i=1}^{3} e'_i \otimes f'_i \) be another rank-3 tensor-product matrix, with

\[ e'_1 = (0, 0, 0, 0, 1), \quad e'_2 = (0, 0, 0, 1, 0), \quad e'_3 = (0, 0, 1, 0, 0), \]

and

\[ f'_1 = (1, 2, -2, 0, 0), \quad f'_2 = (4, 0, 2, 0, 0), \quad f'_3 = (0, 0, -1, 0, 0). \]

Then, \( T_2 = \begin{pmatrix} 0 & 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & -1 & 2 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \).

In order to prove that \([e_1, e_2, e_3] = [f'_1, f'_2, f'_3] \), we shall use the Matlab \texttt{rref} function which produces the reduced row echelon form of the given matrix using Gauss Jordan elimination with partial pivoting. In particular, we use the following code:

\[
d=[1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;1 2 -2 0 0;4 0 2 0 0;0 0 -1 0 0];
\]
\[
v=rref(d)
\]

Then, the results are as follows:

\[
v = \\
1 0 0 0 0 \\
0 1 0 0 0 \\
0 0 1 0 0 \\
0 0 0 0 0 \\
0 0 0 0 0 \\
0 0 0 0 0 \\
0 0 0 0 0
\]

which implies that \([e_1, e_2, e_3] = [f'_1, f'_2, f'_3] \).

On the other hand, using the \texttt{ginv} function it is easy to prove that
\[(T_1T_2)^\dagger = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1331 & 2695 & 1091 & -267 & 612 \\
-269 & -130 & 797 & -113 & 401 \\
2895 & 1049 & 9291 & -4024 & 26244 \\
149 & 263 & 253 & 2577 & -143 \\
2099 & 4247 & 2114 & 26801 & 4030
\end{pmatrix} = T_2^\dagger T_1^\dagger\]

7. Appendix: Matlab code of the 'ginv', 'ginvtest' functions.

The \texttt{ginv} function

function ginv = ginv(X)
%Returns the Moore-Penrose inverse of the argument
if isempty(X)
    quick return
    ginv = zeros(size(X'),class(X));
else
    \[n,m] = \text{size}(X)\];
    if rank(X) < min(n,m);
        error('matrix must be of full rank');
    else
        if n > m,
            \[C = X'X\];
            ginv = \text{C}\backslash X';
        else
            \[C = X'X';\]
            \[G = C\backslash X;\]
            ginv = G';
        end
    end
end

The \texttt{ginvtest} function

function Ginvtest = ginvtest(E)
%Returns the recorded errors for each one of the three %tested methods.
format short e
    \[G1 = \text{pinv}(E);\] %SVD method
G2 = geninv(E);  %Courrie's method
G3 = ginv(E);  %Proposed method

%The error matrices for each one of the previous methods.
D1 = E*G1*E-E;
D2 = G1*E*G1-G1;
D3 = (E*G1)'*E*G1;
D4 = (G1*E)'*G1*E;
F1 = E*G2*E-E;
F2 = G2*E*G2-G2;
F3 = (E*G2)'*E*G2;
F4 = (G2*E)'*G2*E;
R1 = E*G3*E-E;
R2 = G3*E*G3-G3;
R3 = (E*G3)'*E*G3;
R4 = (G3*E)'*G3*E;

%The accuracy of the algorithms were tested
%with the matrix 2-norm.
Pinverror1 = norm(D1,2);
Pinverror2 = norm(D2,2);
Pinverror3 = norm(D3,2);
Pinverror4 = norm(D4,2);
Geninverror1 = norm(F1,2);
Geninverror2 = norm(F2,2);
Geninverror3 = norm(F3,2);
Geninverror4 = norm(F4,2);
Ginverror1 = norm(R1,2);
Ginverror2 = norm(R2,2);
Ginverror3 = norm(R3,2);
Ginverror4 = norm(R4,2);
Ginvtest = [Pinverror1 Pinverror2 Pinverror3
Pinverror4;Geninverror1 Geninverror2 Geninverror3
Geninverror4;Ginverror1 Ginverror2 Ginverror3 Ginverror4];

REFERENCES