Applied Mathematics and Computation 217 (2011) 9828–9834

Contents lists available at ScienceDirect

Applied Mathematics and Computation

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

An improved method for the computation of the Moore–Penrose inverse matrix

Vasilios N. Katsikis [a,*], Dimitrios Pappas [b], Athanassios Petralias [b]

[a] General Department of Mathematics, Technological Education Institute of Piraeus, Aigaleo, 12244 Athens, Greece
[b] Athens University of Economics and Business, Department of Statistics, 76 Patission Str., 10434 Athens, Greece

ARTICLE INFO

Keywords:
Moore–Penrose inverse matrix
Tensor-product matrix

ABSTRACT

In this article we provide a fast computational method in order to calculate the Moore–Penrose inverse of singular square matrices and of rectangular matrices. The proposed method proves to be much faster and has significantly better accuracy than the already proposed methods, while works for full and sparse matrices.

1. Introduction

Let T be an $n \times n$ real matrix. It is known that if T is singular, then its unique generalized inverse $T^{\dagger}$ (known as the Moore–Penrose inverse) is defined. In this case, whenever 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 a recent article [9], the first two authors provided a new method for the fast computation of the generalized inverse of full rank rectangular matrices and of square matrices with at least one zero row or column. In order to reach this goal, a special type of tensor product of two vectors was used, usually defined in infinite dimensional Hilbert spaces. In this work we extend our method so that it can be used in any kind of matrices, square or rectangular, full rank or not. The numerical experiments show that the proposed method is competitive in terms of accuracy and much faster than the commonly used methods, and can also be used for large sparse matrices.

There are several methods for computing the Moore–Penrose inverse matrix (cf. [2,4,5,8,9,11,13,14]). One of the most commonly used methods is the Singular Value Decomposition (SVD) method. This method is very accurate but time-intensive since it requires a large amount of computational resources, especially in the case of large matrices. On a recent work, Toutounian and Ataei [14] presented an algorithm based on the conjugate Gram-Schmidt process and the Moore–Penrose inverse of partitioned matrices, the `CGS-MPi` algorithm and they concluded that this algorithm is a robust and efficient tool for computing the Moore–Penrose inverse of large sparse and rank deficient matrices. Also, the recent work [13], of Petković and Stanimirović proposes a new iterative method, which is derived from the second Penrose equation, for the computation of the Moore–Penrose inverse. Finally, in terms of speed, an interesting computational method for the Moore–Penrose inverse is the one proposed by Courrieu in [4].

In the present manuscript, we construct a very fast and reliable method (see the `qrginv` function in the Appendix) in order to estimate the Moore–Penrose inverse matrix. The computational effort required for the `qrginv` function (see Tables 1–11) in order to obtain the Moore–Penrose inverse is substantially lower, particularly for large matrices, compared to those provided by the SVD method and the methods presented by Toutounian and Ataei in [14], by Petković and Stanimirović in [13] and by Courrieu in [4]. In addition, we obtain reliable and very accurate approximations in each tested cases. In order to test our method, we use random singular matrices (see Section 4.1) as well as a collection of singular test matrices

* Corresponding author.
E-mail addresses: vaskats@gmail.com (V.N. Katsikis), dpappas@aueb.gr, pappdimitris@gmail.com (D. Pappas), petralia@aueb.gr (A. Petralias).

0096-3003/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2011.04.080

Table 1
Error and Computational Time Results; random singular matrices.

Rank\Cond	Method	Time	$\|TT^{\dagger}T - T\|_2$	$\|T^{\dagger}TT^{\dagger} - T^{\dagger}\|_2$	$\|TT^{\dagger} - (TT^{\dagger})^*\|_2$	$\|T^{\dagger}T - (T^{\dagger}T)^*\|_2$
2^8\1.18e+18	pinv	0.219279	1.2119e−12	5.5815e−13	2.9899e−13	3.4618e−13
	CGS-MPi	1.492632	2.1273e−8	7.4393e−7	2.0098e−11	5.2990e−7
	geninv	0.052244	1.4507e−8	5.6808 e−7	1.3893e−10	2.3454e−7
	stan	0.132442	1.8460 + 5	1.3845 e+4	3.0860e−14	1.4865e−14
	qrginv	0.034830	8.7988e−13	5.2886e−13	1.5282e−12	1.4353e−13
2^9\3.34e+17	pinv	1.293886	3.0693e−12	2.3141e−12	1.4977e−12	7.3344e−13
	CGS-MPi	21.631733	1.6152e−8	9.7965e−9	9.6874e−11	5.1869e−9
	geninv	0.232794	2.3342e−7	1.2370 e−7	8.0713e−10	5.5621e−8
	stan	0.547468	1.3642 + 6	1.0231 e+5	1.0062e−13	3.8067e−14
	qrginv	0.156921	2.7375e−12	1.5195e−12	2.8013e−12	3.6188e−13
2^{10}\2.90e+17	pinv	16.433	8.2360e−12	3.1784e−12	1.6267e−12	1.6215e−12
	CGS-MPi	828.880	1.5733e−8	3.7021e−7	2.9058e−10	1.6509e−7
	geninv	2.797	1.1847e−6	2.4071 e−6	2.3203e−9	8.8235e−7
	stan	5.613	1.0485 + 7	7.8639 e+5	2.7732e−13	1.0762e−13
	qrginv	1.745	8.4960e−12	2.2247e−12	7.6749e−12	7.4694e−13
2^{11}\1.38e+18	pinv	189.89	4.1122e−11	2.9050e−11	9.3230e−12	9.4529e−12
	CGS-MPi	9384.84	6.9999e−7	4.1987e−5	5.6624e−9	1.1585e−5
	geninv	10.24	6.5988e−5	6.8342 e−5	8.0995e−8	1.7684e−5
	stan	18.64	8.2211 + 7	6.1658 e+6	7.4454e−13	2.9862e−13
	qrginv	6.08	4.1104e−11	2.4683e−11	5.9973e−11	6.3105e−12
2^{12}\1.24e+18	pinv	1668.40	1.1937e−10	7.5177e−11	2.4279e−11	5.6894e−11
	CGS-MPi	No results	No results	No results	No results	No results
	geninv	133.68	8.3697e−4	7.9488 e−3	4.8176e−7	1.4262e−3
	stan	232.21	6.5092 + 8	4.8819 e+7	2.1038e−12	8.4248e−13
	qrginv	79.53	1.1753e−10	4.3408e−11	2.3549e−10	1.0131e−11

Note: The rank and the condition number of the tested random singular matrices are reported in the first column. The `CGS-MPi` algorithm was not able to produce numerical results for matrix with rank 2^{12} even after 3 days running.

Table 2
Error and Computational Time Results for `chow` (Rank = 199, Cond = 8.01849e+135).

Method	Time	$\|TT^{\dagger}T - T\|_2$	$\|T^{\dagger}TT^{\dagger} - T^{\dagger}\|_2$	$\|TT^{\dagger} - (TT^{\dagger})^*\|_2$	$\|T^{\dagger}T - (T^{\dagger}T)^*\|_2$
pinv	5.2412e−4	4.7411e−13	1.8966e−13	2.3129e−13	2.2617e−13
CGS-MPi	No results	No results	No results	No results	No results
genginv	1.6065e−4	1.8971e−9	5.9457e−12	1.3883e−11	6.8856e−10
stan	4.1595e−4	1.5819e+5	1.1864e+4	6.3546e−13	4.8618e−13
qrginv	1.1138e−4	5.0120e−13	1.0788e−13	5.2690e−13	1.6915e−13

Note: The `CGS-MPi` algorithm was not able to produce numerical results for the chow matrix, even after 3 days running.

Table 3
Error and Computational Time Results for `cycol` (Rank = 50, Cond = 2.05e+48).

Method	Time	$\|TT^{\dagger}T - T\|_2$	$\|T^{\dagger}TT^{\dagger} - T^{\dagger}\|_2$	$\|TT^{\dagger} - (TT^{\dagger})^*\|_2$	$\|T^{\dagger}T - (T^{\dagger}T)^*\|_2$
pinv	2.7674e−4	2.9655e−13	5.6901e−16	1.3866e−14	1.2511e−14
CGS-MPi	2.8988e−3	4.0189e−14	5.4226e−17	7.1682e−16	1.1086e−15
genginv	8.3933e−5	9.2263e−14	1.6665e−16	1.7769e−15	2.6541e−15
stan	7.4580e−4	5.1429e+3	3.8572e+2	1.7314e−14	1.6589e−14
qrginv	1.1473e−4	4.4052e−14	7.1448e−17	1.5076e−15	1.2248e−15

(see Section 4.2) with "large" condition number (ill-conditioned matrices) from the Matrix Computation Toolbox (see [6]). We also tested the proposed method on sparse matrices from the Matrix Market collection [10] (see Section 4.3). In what follows, we make use of the high-level language Matlab both for calculations of the generalized inverse matrix, as well as, for testing the reliability of the obtained results. Specifically, the Matlab 7.4 (R2007a) Service Pack 3 version of the software was used on an Intel Core i7 920 Processor system running at 2.67 GHz with 6 GB of RAM memory using the Windows XP Professional 64-bit 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 $T \in \mathbb{R}^{m\times n}$, the generalized inverse $T^{\dagger} \in \mathbb{R}^{n\times m}$ (known as the Moore–Penrose inverse) is the unique matrix that satisfies the following four Penrose equations,

Table 4
Error and Computational Time Results for `gearmat` (Rank = 199, Cond = 3.504 e+17).

Method	Time	$\|TT^{\dagger}T-T\|_2$	$\|T^{\dagger}TT^{\dagger}-T^{\dagger}\|_2$	$\|TT^{\dagger}-(TT^{\dagger})^*\|_2$	$\|T^{\dagger}T-(T^{\dagger}T)^*\|_2$
pinv	1.1863e−3	1.8999e−14	3.0333e−13	8.0629e−14	7.5617e−14
CGS-MPi	1.1157e−2	2.3795e−13	3.3921e−13	6.1137e−14	5.4783e−13
genginv	2.9073e−4	1.3048e−11	2.4943e−11	7.9926e−12	2.9379e−11
stan	7.3467e−2	4.5781e−10	3.5717e−6	8.9387e−15	3.5810e−15
qrginv	2.0842e−4	2.8700e−15	2.6279e−13	7.4168e−14	1.9234e−14

Table 5
Error and Computational Time Results for `kahan` (Rank = 199, Cond = 2.30018 e+24).

Method	Time	$\|TT^{\dagger}T-T\|_2$	$\|T^{\dagger}TT^{\dagger}-T^{\dagger}\|_2$	$\|TT^{\dagger}-(TT^{\dagger})^*\|_2$	$\|T^{\dagger}T-(T^{\dagger}T)^*\|_2$
pinv	1.3929e−3	1.4320e−13	4.1553e−9	3.7954e−9	9.0701e−14
CGS-MPi	8.0045e−3	4.0113e+18	4.9217e+35	2.7207e+2	5.7649e+18
genginv	1.4155e−4	2.4407e−1	3.4031e+0	1.9729e−12	3.3729e+0
stan	3.4375e−1	1.2426e−13	1.9475e−4	1.0156e−9	2.3945e−14
qrginv	1.4566e−4	2.1280e−5	1.8223e−9	6.9232e−1	6.9645e−15

Table 6
Error and Computational Time Results for `lotkin` (Rank = 19, Cond = 8.97733 e+21).

Method	Time	$\|TT^{\dagger}T-T\|_2$	$\|T^{\dagger}TT^{\dagger}-T^{\dagger}\|_2$	$\|TT^{\dagger}-(TT^{\dagger})^*\|_2$	$\|T^{\dagger}T-(T^{\dagger}T)^*\|_2$
pinv	7.2083e−4	8.7529e−5	1.0348e+8	1.0047e−3	1.1494e−3
CGS-MPi	1.1164e−3	2.9598e+8	1.2525e+14	6.9268e−9	7.4479e+7
genginv	8.2907e−5	4.0527e−2	7.4873e+3	1.4280e−4	1.0614e+3
stan	1.9109e−3	2.7961e+1	1.3335e−1	2.6709e−16	3.7295e−15
qrginv	1.3118e−4	8.3470e−6	4.8973e−8	4.6290e−2	3.5464e−11

Table 7
Error and Computational Time Results for `prolate` (Rank = 117, Cond = 5.61627 e+17).

Method	Time	$\|TT^{\dagger}T-T\|_2$	$\|T^{\dagger}TT^{\dagger}-T^{\dagger}\|_2$	$\|TT^{\dagger}-(TT^{\dagger})^*\|_2$	$\|T^{\dagger}T-(T^{\dagger}T)^*\|_2$
pinv	4.3844e−4	1.7488e−4	7.3987e+9	9.5686e−3	8.0930e−3
CGS-MPi	6.8297e−3	2.2182e+15	7.8853e+30	6.2484e−1	4.7960e+15
genginv	1.2530e−4	1.6289e+15	1.1005e+31	8.1164e+12	6.4573e+15
stan	7.8016e−2	9.2500e−1	6.9375e−2	5.0460e−18	5.0069e−18
qrginv	1.3988e−4	1.3837e−6	6.1407e−7	4.7715e−2	6.5880e−11

Table 8
Error and Computational Time Results for `hilb` (Rank = 20, Cond = 1.17164 e+19).

Method	Time	$\|TT^{\dagger}T-T\|_2$	$\|T^{\dagger}TT^{\dagger}-T^{\dagger}\|_2$	$\|TT^{\dagger}-(TT^{\dagger})^*\|_2$	$\|T^{\dagger}T-(T^{\dagger}T)^*\|_2$
pinv	6.7244e−4	7.7794e−5	3.1844e+9	3.2661e−3	5.1459e−3
CGS-MPi	1.0509e−3	3.9674e+6	2.8112e+12	5.5335e−10	3.0959e+6
genginv	8.2197e−5	8.9410e−1	1.7080e+5	2.3277e−1	1.3477e+4
stan	7.9753e−2	1.3920e+0	1.0440e−1	1.6256e−17	1.5872e−17
qrginv	1.2952e−4	7.8780e−6	8.8150e−9	1.0053e−1	6.9411e−12

Table 9
Error and Computational Time Results for `magic` (Rank = 3, Cond = 4.92358 e+19).

Method	Time	$\|TT^{\dagger}T-T\|_2$	$\|T^{\dagger}TT^{\dagger}-T^{\dagger}\|_2$	$\|TT^{\dagger}-(TT^{\dagger})^*\|_2$	$\|T^{\dagger}T-(T^{\dagger}T)^*\|_2$
pinv	7.7709e−4	1.2110e−8	2.4919e−19	9.7018e−14	4.2360e−14
CGS-MPi	9.5926e−3	1.7557e+18	8.3337e+19	2.9330e+2	6.3290e+14
genginv	6.9553e−5	9.7944e−4	3.7051e−12	1.3245e−12	7.3242e−8
stan	4.3725e−4	4.8004e+18	3.6003e+17	2.1323e−4	3.5543e−4
qrginv	1.1164e−4	1.0034e−8	3.9479e−19	1.8214e−13	4.5669e−14

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

where T^{*} denotes the transpose matrix of T. Let $\mathcal{R}(T)$ be the range of T, the number $r = \dim\mathcal{R}(T)$ is called the rank of T and $\langle\sim,\sim\rangle$ denotes the usual inner-product in $\mathbb{R}^n$.

According to [12], for each $x \in \mathbb{R}^k$, we consider the mapping

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

Assume that $\{e_1,\ldots,e_n\}$ and $\{f_1,\ldots,f_n\}$ are two collections of orthonormal vectors and linearly independent vectors of $\mathbb{R}^k$ with $n < k$, respectively. Then, every rank-n operator T can be written in the form $T = \sum_{i=1}^{n} e_i \otimes f_i$. We shall refer to this type of tensor products as the *tensor-product of the collections* $\{e_1,\ldots,e_n\}$ and $\{f_1,\ldots,f_n\}$. 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. 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:

Theorem 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 λ_i are the solution of an appropriately defined $n \times n$ linear system.

3. The computational method

In [9], based on Theorem 1, the authors developed an algorithm (the `ginv` function) for computing the generalized inverse of full rank matrices, and of square matrices with at least one zero row or column and the rest of the matrix full rank. In other words, our main concern was to calculate the corresponding λ_i in the expansion

Table 10
Error and Computational Time Results for `vand` (Rank = 34, Cond = 1.16262 e+20).

Method	Time	$\|TT^{\dagger}T - T\|_2$	$\|T^{\dagger}TT^{\dagger} - T^{\dagger}\|_2$	$\|TT^{\dagger} - (TT^{\dagger})^{*}\|_2$	$\|T^{\dagger}T - (T^{\dagger}T)^{*}\|_2$
pinv	6.0757e−4	3.3379e−4	1.7085e+8	2.2069e−3	1.8524e−3
CGS-MPi	1.1492e−3	6.4988e+11	5.2998e+20	1.0258e−5	2.3906e+11
genginv	1.0859e−4	2.3355e+7	6.6810e+13	8.7269e−2	6.6271e+7
stan	1.2877e−3	8.4260e+2	6.3195e+1	8.0285e−16	4.3501e−15
qrginv	1.2178e−4	1.2730e−5	2.6181e−7	5.3304e−1	5.5535e−11

Table 11
Error and Computational Time Results; matrix market sparse matrices.

Matrix	Method	Time	$\|TT^{\dagger}T - T\|_2$	$\|T^{\dagger}\,TT^{\dagger} - T^{\dagger}\|_2$	$\|TT^{\dagger} - (TT^{\dagger})\,^{*}\|_2$	$\|T^{\dagger}T - (T^{\dagger}T)^{*}\|_2$
WELL1033_Z (m = 1033)	CGS-MPi	0.0676	1.8834e−12	1.3715e−10	2.7751e−13	1.3977e−11
	geninv	0.0040	1.7076e−9	2.4503e−9	2.3320e−9	2.9404e−9
	stan	26.3367	4.3608e−13	1.5979e−12	6.9434e−12	2.9385e−13
	qrginv	0.0075	8.7761e−13	2.0632e−10	7.3052e−11	1.69030e−12
WELL1850_Z (m = 1850)	CGS-MPi	0.04999	1.2540e−12	8.1432e−12	1.0332e−13	2.7288e−12
	geninv	0.0376	8.9331e−10	2.1151e−9	1.0874e−9	2.0244e−9
	stan	237.3757	6.6400e−13	2.7182e−12	1.6547e−11	5.9684e−13
	qrginv	0.0381	1.2844e−12	1.0488e−10	7.2368e−11	2.0119e−12
ILCC1850_Z (m = 1850)	CGS-MPi	0.4825	2.7959e+4	3.3363e+8	1.3175e−11	4.2739e+4
	geninv	0.0372	1.0540e−8	2.8375e−7	9.8848e−9	2.4027e−8
	stan	252.9536	2.2914e−12	3.8153e−11	1.1974e−10	1.7151e−12
	qrginv	0.0970	9.9285e−12	3.9597e−8	5.1084e−9	6.9214e−11
GR-30–30_Z (m = 900)	CGS-MPi	1.0967	3.3165e−12	2.3569e−12	1.1702e−13	2.1527e−11
	geninv	0.0579	3.5823e−9	3.0613e−9	5.5006 e−8	2.0268e−10
	stan	174.4744	9.4645e−12	6.7195e−13	1.8536e−12	1.3377e−12
	qrginv	0.0363	1.2428e−11	4.7846e−10	2.7574e−10	7.3211e−12
WATT1_Z (m = 1856)	CGS-MPi	1.2534	1	1.7239e+7	1.6424e−16	3.8459
	geninv	0.3535	8.1348e+4	1.1447e+16	5.0625e+12	9.8191e+3
	stan	1630.1	2.2609e−14	7.7154e−4	4.6808e−11	1.1804e−11
	qrginv	0.0154	3.1547e−15	0.5452	7.4696e−9	1.4173e−6

Note: In parenthesis is denoted the row size (m) of each matrix.

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

where $\{e_1,\ldots,e_n\}$ are the first n vectors of the standard basis of $\mathbb{R}^k$, in order to compute the generalized inverse $T^{\dagger}$.

To extend this result for any kind of matrix, we make use of the QR factorization, as well as, the reverse order law for generalized inverses. The following proposition is a restatement of a part of Bouldin's theorem [1] which holds for operators and matrices (see also [3,7]).

Proposition 2. *Let A, B be bounded operators on $\mathcal{H}$ with closed ranges. 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 $A^{*}A$.*

Using the above proposition, we have the following result, which can be found also, in a similar form but with a different proof, in [2].

Proposition 3. *Let $A = QR$ be the QR factorization of A. Then, $A^{\dagger} = R^{\dagger}Q^{*}$.*

Proof 1. We must prove that the conditions of Bouldin's theorem hold. The first condition always holds, since in the case of matrices the range is always closed. For the second condition, it is easy to see that since Q is a unitary matrix, $Q^{\dagger} = Q^{*} = Q^{-1}$ and so

$$Q^{\dagger}QRR^{*} = Q^{-1}QRR^{*} = IRR^{*} = RR^{*}I = RR^{*}Q^{\dagger}Q.$$

The third condition can be proved in a similar way. □

Using the QR factorization, the matrix R is upper triangular but not necessarily of full rank. So a variant of the QR method must be used, the QR with column pivoting as described in [15],

Theorem 4 (*[15, Theorem 3.3.11]*). *Let $A \in \mathbb{R}^{n\times m}$ be a matrix, with rank $(A) = r > 0$. Then there exist matrices $\widehat{A}, Q, R$ such that $\widehat{A}$ is obtained by A by permuting its columns, $Q \in \mathbb{R}^{n\times n}$ is orthogonal, $R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{n\times m}, R_{11} \in \mathbb{R}^{r\times r}$ is nonsingular and upper triangular and $\widehat{A} = QR$.*

Using the above theorem, we have that $AP = QR$, where P is a permutation matrix (therefore unitary). By Proposition 3 we have that $A^{\dagger} = PR^{\dagger}Q^{*}$.

To calculate the rank of R_{11}, one needs only the number of columns of R_{11} that have at least one value above a tolerance level in absolute terms. This tolerance is set equal to 10^{-13}, and turns out to provide accurate results. Note that, in Toutounian and Ataei [14] the corresponding tolerance is set equal to 10^{-5}. The implementation of all the above ideas are presented in the `qrginv` function (see the Appendix).

4. Numerical experiments

In this section we perform numerical experiments comparing Matlab's `pinv` function, Toutounian and Ataei' s method [14] (`CGS-MPi`), Courrie' s method [4] (`geninv`), Petković and Stanimirović s method [13] (`stan`) and the proposed method `qrginv`. Our proposed numerical method is based on the introduction of the `qrginv` function and enables us to perform fast and accurate estimations in all the tests that were conducted. All algorithms were carefully implemented and tested in Matlab.

Testing of `pinv`, `CGS-MPi`, `geninv`, `stan` and `qrginv` was performed separately for random singular matrices and for singular test matrices with "large" condition number from Higham's Matrix Computation Toolbox (`mctoolbox`), see [6]. From the comparative results, Sections 4.1 and 4.2, it is evident that using the proposed method (`qrginv`) the interested user can reach a fast computational solution using a reduced amount of computational resources while the accuracy of the results remains in a high level. Also, it is notable that in many cases the other methods could not obtain the Moore–Penrose inverse with acceptable accuracy.

As in [14], we also tested the proposed method in sparse matrices, Section 4.3, from the Matrix Market collection [10] and we obtained very fast and accurate results. In particular, in one of our tests (matrix `WATT1_Z`), the `CGS-MPi` and the `geninv` methods could not obtain the Moore–Penrose inverse while the `qrginv` method needed about $0.095 \cdot 10^{-6}$% the corresponding time needed by the `stan` function to calculate the Moore–Penrose inverse.

4.1. Random singular matrices

We are comparing the performance of the proposed method (`qrginv`) to that of the SVD method used by Matlab (`pinv`), the method proposed in [14] (`CGS-MPi algorithm`), Courrie's method [4] (`geninv`), Petković and Stanimirović s method

[13] (stan) on a series of random singular matrices with rank 2^n, $n = 8, 9, 10, 11, 12$. In addition, the accuracy of the results is examined with the matrix 2-norm in error matrices corresponding to the four properties characterizing the Moore–Penrose inverse.

In Table 1 we can see the time efficiency, as well as, the accuracy of the proposed method versus the other four methods (pinv, CGS-MPi, geninv and stan). The qrginv method needs about 3.2% up to 15.8% the corresponding time needed by the pinv function to calculate the Moore–Penrose inverse, depending on the matrix. On the other hand the CGS-MPi turns to be computationally demanding requiring from 43 times up to more than 1500 times the corresponding time needed by qrginv. Furthermore, the larger the rank of the matrix, the greater this difference, so that for a matrix with rank 2^{12}, the CGS-MPi algorithm was not able to produce a numerical result, even after 3 days running, while qrginv needs only up to 79 s. The geninv method becomes very competitive in terms of speed, but the accuracy of the results is low. The stan method was not able to produce correct Moore–Penrose inverses due to the large errors concerning the first two properties characterizing the Moore–Penrose inverse.

From the above discussion, it is evident that the proposed method (qrginv) produced a reliable approximation since the associated errors are greatly lower than the corresponding of the other methods. In addition, the qrginv method proved substantially faster in all the tests that were conducted. Therefore, in this case, the proposed method allows us for a both fast and accurate computation of the Moore–Penrose inverse matrix.

4.2. Singular test matrices

In this section we use a set of singular test matrices that includes 9 singular matrices, of size 200×200, obtained from the function matrix in the Matrix Computation Toolbox (mctoolbox) [6] (which includes test matrices from Matlab itself). The condition number of these matrices range from order 10^{15} to 10^{135}. For comparative purpose we also apply, as in the previous section, the four methods pinv, CGS-MPi, geninv and stan. Since the tested matrices are of relatively small size and so as to measure the time needed for each algorithm to compute the Moore–Penrose inverse accurately, each algorithm runs 100 distinct times. The reported time is the mean time over these 100 replications. For each case, the time responses together with the error results are presented in Tables 2–10.

The results show a clear ordering of the five methods for this set of test problems, with qrginv in the first place, followed by pinv, then by stan, then by geninv and then by CGS-MPi. The worst cases for pinv, with comparatively large errors for the $\|T^\dagger TT^\dagger - T^\dagger\|_2$ norm, are the lotkin, prolate, hilb, and vand matrices. The stan method produces significant errors in the cases of chow, cycol, magic and vand matrices for the $\|TT^\dagger T - T\|_2$ and the $\|T^\dagger TT^\dagger - T^\dagger\|_2$ norms and in the cases of lotkin and hilb matrices for the $\|TT^\dagger T - T\|_2$ norm. The geninv method has large errors in the cases of kahan, lotkin and hilb matrices for the $\|T^\dagger TT^\dagger - T^\dagger\|_2$ and the $\|T^\dagger T - (T^\dagger T)^*\|_2$ norms while the worst cases are the prolate and the vand matrices with comparatively large errors for all tested norms. The CGS-MPi algorithm has very large errors in the cases of kahan, lotkin, prolate, hilb, magic and vand matrices, for all or almost all tested norms. Moreover, the algorithm failed to produce a numerical result for the chow matrix, since the computed Moore–Penrose inverse included only NaNs. Nevertheless, we observe that there are also cases (in certain norms) that the CGS-MPi algorithm has smaller error than pinv. On the other hand, the proposed qrginv method gives very accurate results for all matrices and proves to be overall more efficient than the other four methods.

4.3. Matrix market sparse matrices

In this section we test the proposed algorithm on sparse matrices, from the Matrix Market collection [10]. We follow the same method and the same matrices as in [14], while the matrices are taken as rank deficient: $A\_Z = [A\ Z]$, where A is one of the chosen matrices, Z is a zero matrix of order $m \times 100$ and m takes values shown in Table 11. For an extensive presentation of the chosen matrices properties the reader may refer to [10].

In Table 11, the proposed method is tested against the CGS-MPi, the geninv and the stan methods. Note that, the CGS-MPi is proposed by Toutounian and Ataei [14] as suitable for large and sparse matrices. The pinv function of Matlab is not applicable in sparse matrices and thus omitted. The CGS-MPi algorithm has very large errors in the cases of ILCC1850_Z and WATT1_Z matrices. On the other hand the time responses are reasonable and in all the other cases the Moore–Penrose inverses obtained by the CGS-MPi were accurate enough. The geninv method has large errors in the case of WATT1_Z matrix but in all the other cases it is time efficient with relatively small errors. The major advantage of the stan method is the accuracy but the basic disadvantage is that it is time consuming. We observe that in sparse matrices as well, the proposed method gave fast and accurate results in all the tested matrices and seems to greatly outperform the other tested methods, both in terms of speed and accuracy.

5. Concluding remarks

In this paper, we have presented a new method for calculating the Moore–Penrose inverse of singular square, rectangular, full or sparse matrices. This method is based on a previous work of the authors, [9], for the fast computation of the Moore–Penrose inverse of full rank rectangular matrices and of square matrices with at least one zero row or column. In this work, by using the QR factorization, as well as, the reverse order law for generalized inverses, we extend our method so that it can

be used in any kind of matrices. The comparative tests were conducted on random singular matrices, on singular matrices with large condition number from the Matrix Computation Toolbox, [6] and on sparse matrices from the Matrix Market collection, [10]. We compare the performance of the proposed method (`qrginv`) to that of three usual direct methods: Matlab's `pinv` function, Toutounian and Ataei's method [14], Courrie' s method [4] and an iterative well-established method: Petković and Stanimirović s method [13]. Numerical experiments (see Tables 1–11) show that the proposed method provides a substantially faster numerical way for calculating the Moore–Penrose inverse of a given matrix and a far more reliable approximation than that obtained by the other tested methods.

Acknowledgements

The authors thank the referees for their valuable comments that significantly improved the quality of the paper.

Appendix A. Matlab code of the qrginv function

```
function qrginv = qrginv(B)
[N,M] = size(B);
[Q,R,P] = qr(B);
r = sum(any(abs(R)>le-l3,2));
Rl = R(l:r,:);
R2 = ginv(Rl);
R3 = [R2 zeros(M,N-r)];
A = P*R3*Q';
qrginv = A;
```

In case the matrix of interest is sparse, the third line of this code must be replaced with

$$[\mathtt{Q}, \mathtt{R}, \mathtt{P}] = \mathtt{spqr}(\mathtt{B}),$$

since the function `qr` embedded in Matlab does not support sparse matrices under this format. The function `spqr` is a part of the SuiteSparse toolbox, built by Professor Timothy A. Davis, University of Florida and can be downloaded electronically from http://www.cise.ufl.edu/research/sparse/SuiteSparse/.

References

[1] R. Bouldin, The pseudo-inverse of a product, SIAM Journal on Applied Mathematics 24 (4) (1973) 489–495.
[2] A. Ben-Israel, T.N.E. Grenville, Generalized Inverses: Theory and Applications, Springer-Verlag, Berlin, 2002.
[3] T.N.E. Greville, Note on the generalized inverse of a matrix product, SIAM Review 8 (1966) 518–521.
[4] P. Courrieu, Fast Computation of Moore–Penrose Inverse matrices, Neural Information Processing-Letters and Reviews 8 (2005) 25–29.
[5] W. Guo, T. Huang, Method of elementary transformation to compute Moore–Penrose inverse, Applied Mathematics and Computation 216 (2010) 1614–1617.
[6] N.J. Higham, The Matrix Computation Toolbox, <http://www.ma.man.ac.uk/higham/mctoolbox>.
[7] S. Izumino, The product of operators with closed range and an extension of the reverse order law, Tohoku Mathematical Journal 34 (1982) 43–52.
[8] S. Karanasios, D. Pappas, Generalized inverses and special type operator Algebras, Facta Universitatis (NIS), Series Mathematics and Informatics 21 (2006) 41–48.
[9] V.N. Katsikis, D. Pappas, Fast computing of the Moore–Penrose inverse matrix, Electronic Journal of Linear Algebra 17 (2008) 637–650.
[10] Matrix Market, National Institute of Standards and Technology, Gaithersburg, MD. Available from: <http://arxiv.org///math.nist.gov/MatrixMarket>.
[11] M.A. Rakha, On the Moore–Penrose generalized inverse matrix, Applied Mathematics and Computation 158 (2004) 185–200.
[12] J. Ringrose, Compact Non Self Adjoint Operators, Van Nostrand, London, 1971.
[13] M.D. Petković, P.S. Stanimirović, Iterative method for computing the Moore–Penrose inverse based on Penrose equations, Journal of Computational and applied Mathematics 235 (2011) 1604–1613.
[14] F. Toutounian, A. Ataei, A new method for computing Moore–Penrose inverse matrices, Journal of Computational and applied Mathematics 228 (2009) 412–417.
[15] D. Watkins, Fundamentals of Matrix Computations, Wiley Interscience, New York, 2002.