Available online at www.sciencedirect.com
ScienceDirect
Computers and Mathematics with Applications 55 (2008) 1720–1734
An International Journal
computers & mathematics
with applications
www.elsevier.com/locate/camwa

Effective partitioning method for computing weighted Moore–Penrose inverse

Marko D. Petković*, Predrag S. Stanimirović, Milan B. Tasić

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

Received 16 February 2007; received in revised form 10 July 2007; accepted 17 July 2007

Abstract

We introduce a method and an algorithm for computing the weighted Moore–Penrose inverse of multiple-variable polynomial matrix and the related algorithm which is appropriated for sparse polynomial matrices. These methods and algorithms are generalizations of algorithms developed in [M.B. Tasić, P.S. Stanimirović, M.D. Petković, Symbolic computation of weighted Moore–Penrose inverse using partitioning method, Appl. Math. Comput. 189 (2007) 615–640] to multiple-variable rational and polynomial matrices and improvements of these algorithms on sparse matrices. Also, these methods are generalizations of the partitioning method for computing the Moore–Penrose inverse of rational and polynomial matrices introduced in [P.S. Stanimirović, M.B. Tasić, Partitioning method for rational and polynomial matrices, Appl. Math. Comput. 155 (2004) 137–163; M.D. Petković, P.S. Stanimirović, Symbolic computation of the Moore–Penrose inverse using partitioning method, Internat. J. Comput. Math. 82 (2005) 355–367] to the case of weighted Moore–Penrose inverse. Algorithms are implemented in the symbolic computational package MATHEMATICA.

Keywords: Weighted Moore–Penrose inverse; Rational matrices; Polynomial matrices; Sparse matrices; Symbolic computation

1. Introduction

Let $\mathbf{C}^{m\times n}$ be the set of $m \times n$ complex matrices, and $\mathbf{C}_r^{m\times n}$ be the set of $m \times n$ complex matrices of rank r: $\mathbf{C}_r^{m\times n} = \{X \in \mathbf{C}^{m\times n} \mid \operatorname{rank}(X) = r\}$. For any matrix $A \in \mathbf{C}^{m\times n}$ and positive definite Hermitian matrices M and N of the orders m and n respectively, consider the following equations in X, where $*$ denotes the conjugate and transpose:

$$(1)\ AXA = A \qquad (2)\ XAX = X$$
$$(3)\ (MAX)^* = MAX \qquad (4)\ (NXA)^* = NXA.$$

The matrix X satisfying these equations is called the weighted Moore–Penrose inverse of A [4], and it is denoted by $X = A^{\dagger}_{MN}$. In the partial case $M = I_m$, $N = I_n$, the matrix $X = A^{\dagger}_{MN}$ comes to the Moore–Penrose inverse $A^{\dagger}$ of A [4].

* Corresponding author.
E-mail addresses: dexter_of_nis@neobee.net (M.D. Petković), pecko@pmf.ni.ac.yu (P.S. Stanimirović), milan12t@ptt.yu (M.B. Tasić).

0898-1221/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2007.07.014

As usual, $\mathbf{C}[s_1, \ldots, s_p]$ (resp. $\mathbf{C}(s_1, \ldots, s_p)$) denotes the polynomials (resp. rational functions) with complex coefficients in the variables $s_1, \ldots, s_p$. The matrices of format $m \times n$ with elements in $\mathbf{C}[s_1, \ldots, s_p]$ (resp. $\mathbf{C}(s_1, \ldots, s_p)$) are denoted by $\mathbf{C}[s_1, \ldots, s_p]^{m\times n}$ (resp $\mathbf{C}(s_1, \ldots, s_p)^{m\times n}$). An appropriate identity matrix is denoted by I.

The computation of the Moore–Penrose inverse of one-variable polynomial and/or rational matrices, based on the Leverrier–Faddeev algorithm, is investigated in [5–10]. The implementation of this algorithm in the symbolic computational language MAPLE, is described in [11]. An algorithm for computing the Moore–Penrose inverse of two-variable rational and polynomial matrix is introduced in [12]. A quicker and less memory-expensive effective algorithm for computing the Moore–Penrose inverse of one-variable and two-variable polynomial matrix, with respect to those introduced in [8] and [12], is presented in [13]. This algorithm is efficient when elements of the input matrix are polynomials with only few nonzero addends.

Papers [9,14–17] deal with a computation of the *Drazin inverse*. A generalization of these algorithms, introduced in [18], generates the wide class of outer inverses of a rational or polynomial matrix.

An interpolation algorithm for computing the Moore–Penrose inverse of a given one-variable polynomial matrix, based on the Leverrier–Faddeev method, is presented in [19]. Algorithms for computing the Moore–Penrose and the Drazin inverse of one-variable polynomial matrices based on the evaluation–interpolation technique and the Fast Fourier transform are introduced in [20]. Corresponding algorithms for two-variable polynomial matrices are introduced in [21].

In this paper we consider the set of rational and polynomial matrices and various variants of the partitioning method for computing generalized inverses. Greville's *partitioning method* for numerical computation of generalized inverses is introduced in [22]. Two different proofs for Greville's method were presented in [23,24]. A simple derivation of the Greville's result has been given by Udwadia and Kalaba [25]. In [26] Fan and Kalaba used the approach of determination of the Moore–Penrose inverse of matrices using dynamic programming and Belman's principle of optimality. Wang in [27] generalizes Greville's method to the weighted Moore–Penrose inverse.

Many numerical algorithms for computing the Moore–Penrose inverse lack numerical stability. The Greville's algorithm requires more operations and consequently it accumulates more rounding errors (see for example [28]). Moreover, it is well-known that the Moore–Penrose inverse is not necessarily a continuous function of the elements of the matrix. The existence of this discontinuity presents further problems in the pseudoinverse computation. It is therefore clear that cumulative round-off errors should be totally eliminated, which is possible only by means of the symbolic implementation. In the symbolic implementation variables are stored in the "exact" form or can be left "unassigned" (without numerical values), resulting in no loss of accuracy during the calculation [8].

An algorithm for computing the Moore–Penrose inverse of one-variable polynomial and/or rational matrices, based on the Greville's partitioning algorithm, is introduced in [2]. An extension of results from [2] to the set of two-variable rational and polynomial matrices is introduced in the paper [3]. In our recent paper [1] we propose an algorithm for computing the weighted Moore–Penrose of one-variable rational and polynomial matrix. In this work we generalized the results from [1] in the following two ways:

- extend algorithms from [1] to the set of multi-variable rational and polynomial matrices with complex coefficients,
- make algorithms from [1] more effective on sparse matrices with a relatively small number of nonzero elements.

The structure of the paper is as follows. In the second section we extend the algorithm for computing the weighted Moore–Penrose from [27] to the set of multiple-variable rational matrices with complex coefficients. The main results are given in the third and the fourth sections. In Section 3 we adapt the previous algorithm to the set of polynomial matrices. In the fourth section we consider two effective structures which exploit only nonzero addends in polynomial matrices and improve previous results on the set of sparse matrices. In the last section we presented an illustrative example and compared various algorithms.

2. Weighted Moore–Penrose inverse for multi-variable rational matrices

Let $A(s_1, \ldots, s_p)$ be a complex rational matrix. For the sake of simplicity, we will introduce new variables $s_{2p+1-i} = \overline{s_i}$. Also we will denote the vector of all variables $s_1, \ldots, s_{2p}$ by $S = (s_1, \ldots, s_{2p})$ and further we will denote $A(s_1, \ldots, s_p)$ as $A(S)$.

By $A_i(S)$ we denote the submatrix of $A(S)$ consisting of its first i columns, and by $a_i(S)$ is denoted the ith column of $A(S)$:

$$A_i(S) = [A_{i-1}(S) \mid a_i(S)], \quad i = 2, \ldots, n, \ A_1(S) = a_1(S). \tag{2.1}$$

We will consider positive definite Hermitian matrices $M(S) \in \mathbf{C}(S)^{m\times m}$ and $N(S) \in \mathbf{C}(S)^{n\times n}$. The leading principal submatrix $N_i(S) \in \mathbf{C}(S)^{i\times i}$ of $N(S)$ is partitioned as

$$N_i(S) = \begin{bmatrix} N_{i-1}(S) & l_i(S) \\ l_i^*(S) & n_{ii}(S) \end{bmatrix}, \quad i = 2, \ldots, n, \tag{2.2}$$

where $l_i(S) \in \mathbf{C}(S)^{(i-1)\times 1}$ and $n_{ii}(S)$ is the complex polynomial. By $N_1(S)$ we denote the polynomial $n_{11}(S)$.

In the following lemma we generalize the representations of the weighted Moore–Penrose inverse from [24,1] to the set of rational matrices of multiple complex variables $\mathbf{C}(S)^{m\times n}$.

For the sake of simplicity, by $X_i(S)$ we denote the weighted Moore–Penrose inverse corresponding to $M(S)$ and submatrices $A_i(S)$, $N_i(S)$: $X_i(S) = A_i(S)^{\dagger}_{MN_i}$, for each $i = 2, \ldots, n$. Similarly $X_1(S) = a_1(S)^{\dagger}_{M,N_1}$.

Lemma 2.1. *Let $A(S) \in \mathbf{C}(S)^{m\times n}$, assume that $M(S) \in \mathbf{C}(S)^{m\times m}$, $N(S) \in \mathbf{C}(S)^{n\times n}$ are positive definite Hermitian matrices, and let $A_i(S)$ be the submatrix of $A(S)$ consisting of its first i columns, as it is defined in (2.1). Assume that the leading principal submatrix $N_i(S) \in \mathbf{C}(S)^{i\times i}$ is partitioned as in (2.2). Then the matrices $X_i(S)$ can be computed in this way:*

$$X_1(S) = \begin{cases} \left(a_1^*(S)M(S)a_1(S)\right)^{-1} a_1^*(S)M(S), & a_1(S) \neq 0, \\ a_1^*(S), & a_1(S) = 0, \end{cases} \tag{2.3}$$

$$X_i(S) = \begin{bmatrix} X_{i-1}(S) - \left(d_i(S) + (I - X_{i-1}(S)A_{i-1}(S))\, N_{i-1}^{-1}(S)l_i(S)\right)b_i^*(S) \\ b_i^*(S) \end{bmatrix}, \quad i = 2, \ldots, n, \tag{2.4}$$

where the vectors $d_i(S)$, $c_i(S)$ and $b_i^*(S)$ are defined by

$$d_i(S) = X_{i-1}(S)a_i(S) \tag{2.5}$$

$$c_i(S) = a_i(S) - A_{i-1}(S)d_i(S) = (I - A_{i-1}(S)X_{i-1}(S))\, a_i(S) \tag{2.6}$$

$$b_i^*(S) = \begin{cases} \left(c_i^*(S)M(S)c_i(S)\right)^{-1} c_i^*(S)M(S), & c_i(S) \neq 0 \\ \delta_i^{-1}(S)\left(d_i^*(S)N_{i-1}(S) - l_i(S)^*\right)X_{i-1}(S), & c_i(S) = 0, \end{cases} \tag{2.7}$$

and where in $b_i^*(S)$ is

$$\begin{aligned} \delta_i(S) = {} & n_{ii}(S) + d_i^*(S)N_{i-1}(S)d_i(S) - \left(d_i^*(S)l_i(S) + l_i^*(S)d_i(S)\right) \\ & - l_i^*(S)\,(I - X_{i-1}(S)A_{i-1}(S))\, N_{i-1}^{-1}(S)l_i(S). \end{aligned} \tag{2.8}$$

Also in [24], the authors used a block representation of the inverse $N_i^{-1}(S)$, which we also generalized to the set of rational matrices.

Lemma 2.2. *Let $N_i(S)$ be the partitioned matrix defined in (2.2). Assume that $N_i(S)$ and $N_{i-1}(S)$ are both nonsingular. Then*

$$N_i^{-1}(S) = \begin{cases} \begin{bmatrix} N_{i-1}(S) & l_i(S) \\ l_i^*(S) & n_{ii}(S) \end{bmatrix}^{-1} = \begin{bmatrix} E_{i-1}(S) & f_i(S) \\ f_i^*(S) & h_{ii}(S) \end{bmatrix}, & i = 2, \ldots, n, \\ n_{11}^{-1}(S), \quad i = 1, \end{cases} \tag{2.9}$$

where

$$h_{ii}(S) = \left(n_{ii}(S) - l_i^*(S)N_{i-1}^{-1}(S)l_i(S)\right)^{-1} \tag{2.10}$$

$$f_i(S) = -h_{ii}(S)N_{i-1}^{-1}(S)l_i(S) \tag{2.11}$$

$$E_{i-1}(S) = N_{i-1}^{-1}(S) + h_{ii}^{-1}(S)f_i(S)f_i^*(S). \tag{2.12}$$

In view of Lemmas 2.1 and 2.2, respectively, we present the following algorithms for computing the weighted Moore–Penrose inverse and the inverse matrix $N_i^{-1}(S) \in \mathbf{C}(S)^{i\times i}$. These algorithms are generalizations of corresponding algorithms from [1] to the set of multiple-variable rational matrices with complex coefficients.

Algorithm 2.1. Input: $A(S) \in \mathbf{C}(S)^{m\times n}$ and positive definite matrices $M(S) \in \mathbf{C}(S)^{m\times m}$ and $N(S) \in \mathbf{C}(S)^{n\times n}$.

Step 1. Initial value: Compute $X_1(S) = a_1(S)^\dagger$ defined in (2.3).
Step 2. Recursive step: For each $i = 2, \ldots, n$ compute $X_i(S)$ performing the following four steps:
  Step 2.1. Compute $d_i(S)$ using (2.5).
  Step 2.2. Compute $c_i(S)$ using (2.6).
  Step 2.3. Compute $b_i^*(S)$ by means of (2.7) and (2.8).
  Step 2.4. Applying (2.4) compute $X_i(S)$.
Step 3. The stopping criterion: $i = n$. Return $X_n(S)$.

Algorithm 2.2. Let $N_i(S) = \begin{bmatrix} N_{i-1}(S) & l_i(S) \\ l_i^*(S) & n_{ii}(S) \end{bmatrix}$ be the leading principal submatrix of positive definite matrix $N \in \mathbf{C}(S)^{n\times n}$. Then the inverse matrix $N^{-1}(S)$ can be computed as follows:

Step 1. Initial values: $N_1^{-1}(S) = n_{11}^{-1}(S)$.
Step 2. Recursive step: For $i = 2, \ldots, n$ perform the following steps:
  Step 2.1. Compute $h_{ii}(S)$ using (2.10).
  Step 2.2. Compute $f_i(S)$ using (2.11).
  Step 2.3. Compute $E_{i-1}(S)$ using (2.12).
  Step 2.4. Compute $N_i^{-1}(S)$ using (2.9).
Step 3. For $i = n$ return the inverse matrix $N^{-1}(S) = N_n^{-1}(S)$.

We used MATHEMATICA function *Together* in order to enable simplifications of rational expressions (this function joins rational addends together and cancels common multipliers in the numerator and denominator).

3. Weighted Moore–Penrose inverse for multi-variable polynomial matrices

Now suppose that $A(S) \in \mathbf{C}[S]^{m\times n}$ is a multi-variable polynomial matrix. We can represent it in the following polynomial form:

$$A(S) = \sum_{i_1=0}^{d_1} \cdots \sum_{i_{2p}=0}^{d_{2p}} A_{i_1,\ldots,i_{2p}} s_1^{i_1} \cdots s_{2p}^{i_{2p}} = \sum_{I=0}^{Q} A_I S^I, \tag{3.1}$$

where $I = (i_1, \ldots, i_{2p})$, $A_I = A_{i_1,\ldots,i_{2p}}$ are constant $m \times n$ matrices, $S^I = s_1^{i_1} s_2^{i_2} \cdots s_{2p}^{i_{2p}}$, $Q = (d_1, \ldots, d_{2p}) = deg A(S)$. Here d_i is the degree of the matrix polynomial with respect to the variable s_i in $A(S)$.

If by $\overline{J}$ we denote $\overline{J} = (j_{2p}, \ldots, j_1)$, where $J = (j_1, \ldots, j_{2p})$ then it can be easily checked that it holds for $A^*(S) = \sum_{J=0}^{\overline{Q}} A_J^* S^J$.

An application of Algorithm 2.1 to the multiple-variable polynomial matrix $A(S)$ gives the following result.

Theorem 3.1. *Let us consider $A(S) \in \mathbf{C}[S]^{m\times n}$ of the form (3.1) and positive definite Hermitian matrices $M(S) \in \mathbf{C}(S)^{m\times m}$ and $N(S) \in \mathbf{C}(S)^{n\times n}$. Assume that the leading principal submatrix $N_i(S) \in \mathbf{C}(S)^{i\times i}$ of $N(S)$ is partitioned as in (2.2). Then the weighted Moore–Penrose inverse $A_{MN_i}^\dagger(S) \in \mathbf{C}^{i\times m}[S]$ corresponding to the first i columns in $A(S)$ is of the form*

$$X_i(S) = A_{MN_i}^\dagger(S) = \frac{Z_i(S)}{Y_i(S)}, \quad i = 1, \ldots, n, \tag{3.2}$$

where $Z_i(S) \in \mathbf{C}^{m\times i}[S]$ and $Y_i(S) \in \mathbf{C}[S]$, can be computed from $Z_{i-1}(S)$, $Y_{i-1}(S)$, $A_{i-1}(S)$ and $a_i(S)$ using exact recurrence relations.

Proof. We will prove theorem by the induction. In the case $i = 1$ exact relations for $Z_1(S)$ and $Y_1(S)$ can be derived from (2.3):

$$a_1(S) = A_1(S) = 0 \Rightarrow Z_1(S) = 0, \quad Y_1(S) = 1$$
$$a_1(S) = A_1(S) \neq 0 \Rightarrow Z_1(S) = a_1^*(S)M(S), \quad Y_1(S) = a_1^*(S)M(S)a_1(S).$$

Consider now the inductive step. From the inductive hypothesis we can write $X_{i-1}(S) = \frac{Z_{i-1}(S)}{Y_{i-1}(S)}$. Then $X_i(S)$ can be computed by using Step 2 of Algorithm 2.1. From Steps 2.1 and 2.2 we have:

$$d_i(S) = X_{i-1}(S)a_i(S) = \frac{Z_{i-1}(S)a_i(S)}{Y_{i-1}(S)} = \frac{D_i(S)}{Y_{i-1}(S)}$$
$$c_i(S) = a_i(S) - A_{i-1}(S)d_i(S) = \frac{a_i(S)Y_{i-1}(S) - A_{i-1}(S)D_i(S)}{Y_{i-1}(S)} = \frac{C_i(S)}{Y_{i-1}(S)}.$$

If $C_i(S) \neq 0$, according to the Step 2.3 of Algorithm 2.1 we have:

$$b_i^*(S) = \frac{\frac{C_i^*(S)}{Y_{i-1}^*(S)}M(S)}{\frac{C_i^*(S)}{Y_{i-1}^*(S)}M(S)\frac{C_i(S)}{Y_{i-1}(S)}} = \frac{Y_{i-1}(S)C_i^*(S)M(S)}{C_i^*(S)M(S)C_i(S)} = \frac{V_i(S)}{W_i(S)}.$$

Otherwise, we need to evaluate first the expression $\delta_i(S)$. From (2.8) we obtain:

$$\delta_i(S) = n_{ii}(S) + \frac{D_i^*(S)}{Y_{i-1}^*(S)}N_{i-1}(S)\frac{D_i(S)}{Y_{i-1}(S)} - \left(\frac{D_i^*(S)}{Y_{i-1}^*(S)}l_i(S) + l_i^*(S)\frac{D_i(S)}{Y_{i-1}(S)}\right) - l_i^*(S)\frac{\phi_i(S)}{\psi_i(S)}. \tag{3.3}$$

Here we used the inductive hypothesis together with temporary polynomial matrix $\phi_i(S) \in \mathbf{C}[S]^{(i-1)\times 1}$ and polynomial $\psi_i(S)$ are defined by:

$$\begin{aligned}(I - X_{i-1}(S)A_{i-1}(S))\,N_{i-1}^{-1}(S)l_i(S) &= \frac{Y_{i-1}(S)I - Z_{i-1}(S)A_{i-1}(S)}{Y_{i-1}(S)} \cdot \frac{\widetilde{N}_{i-1}(S)}{\breve{N}_{i-1}(S)} \cdot l_i(S) \\ &= \frac{Y_{i-1}(S)\widetilde{N}_{i-1}(S)l_i(S) - Z_{i-1}(S)A_{i-1}(S)\widetilde{N}_{i-1}(S)l_i(S)}{Y_{i-1}(S)\breve{N}_{i-1}(S)} \\ &= \frac{\phi_i(S)}{\psi_i(S)}. \end{aligned} \tag{3.4}$$

Also, we use $N_{i-1}^{-1}(S) = \frac{\widetilde{N}_i(S)}{\breve{N}_i(S)}$, where $\widetilde{N}_i(S) \in \mathbf{C}[S]^{(i-1)\times(i-1)}$ and $\breve{N}_i(S) \in \mathbf{C}[S]$ are defined in the next theorem. By collecting addends under the same denominator in (3.3) we can write $\delta_i(S)$ in the form:

$$\delta_i(S) = \frac{\widetilde{\Delta}_i(S)}{\breve{\Delta}_i(S)},$$

where:

$$\begin{aligned}\widetilde{\Delta}_i(S) &= n_{ii}(S)\breve{N}_{i-1}(S)Y_{i-1}^*(S)Y_{i-1}(S) + \breve{N}_{i-1}(S)D_i^*(S)N_{i-1}(S)D_i(S) \\ &\quad - \left(Y_{i-1}(S)D_i^*(S)l_i(S) + Y_{i-1}^*(S)D_i(S)l_i^*(S)\right)\breve{N}_{i-1}(S) - l_i^*(S)\phi_i(S)Y_{i-1}^*(S) \\ \breve{\Delta}_i(S) &= Y_{i-1}^*(S)Y_{i-1}(S)\breve{N}_{i-1}(S).\end{aligned}$$

Now we apply Step 2.3 in the case $C_i(S) = 0$ and evaluate $b_i^*(S)$:

$$\begin{aligned}b_i^*(S) &= \frac{\widetilde{\Delta}_i(S)}{\breve{\Delta}_i(S)}\left(\frac{D_i^*(S)}{Y_{i-1}^*(S)}N_{i-1}(S) - l_i(S)^*\right)\frac{Z_{i-1}(S)}{Y_{i-1}(S)} \\ &= \frac{\widetilde{N}_{i-1}(S)\left(D_i^*(S)N_{i-1}(S) - Y_{i-1}^*(S)l_i^*(S)\right)Z_{i-1}(S)}{\breve{\Delta}_i(S)} = \frac{V_i(S)}{W_i(S)}.\end{aligned}$$

Let us now rewrite expression (2.4) in the following way:

$$X_i(S) = \begin{bmatrix} \dfrac{Z_{i-1}(S)}{Y_{i-1}(S)} - \left(\dfrac{D_i(S)}{Y_{i-1}(S)} + \dfrac{\phi_i(S)}{\psi_i(S)}\right)\dfrac{V_i(S)}{W_i(S)} \\ \dfrac{V_i(S)}{W_i(S)} \end{bmatrix}$$

$$= \frac{1}{W_i(S)\psi_i(S)} \begin{bmatrix} W_i(S)\breve{N}_{i-1}(S)Z_{i-1}(S) - \left(D_i(S)\breve{N}_{i-1}(S) + \phi_i(S)\right)V_i(S) \\ \breve{N}_{i-1}(S)Y_{i-1}(S)V_i(S) \end{bmatrix}.$$

From the last expression we obviously have that it holds for:

$$Z_i = \begin{bmatrix} W_i(S)\breve{N}_{i-1}(S)Z_{i-1}(S) - \left(D_i(S)\breve{N}_{i-1}(S) + \phi_i(S)\right)V_i(S) \\ \breve{N}_{i-1}(S)Y_{i-1}(S)V_i(S) \end{bmatrix} = \begin{bmatrix} \Theta_i(S) \\ \Psi_i(S) \end{bmatrix}$$

$$Y_i = W_i(S)\psi_i(S).$$

This completes the proof of the theorem. □

Theorem 3.2. *Let the leading principal submatrix $N_i(S)$ of the positive definite matrix $N(S) \in \mathbf{C}[s]^{n\times n}$ be partitioned as in (2.2). Then the inverse $N_i^{-1}(S)$ is of the form:*

$$N_i^{-1}(S) = \frac{\widetilde{N}_i(S)}{\breve{N}_i(S)} = \frac{1}{\breve{N}_i(S)} \begin{bmatrix} E_{i-1}(S) & F_i(S) \\ F_i^*(S) & H_{ii}(S) \end{bmatrix},$$

where $E_{i-1}(S) \in \mathbf{C}^{(i-1)\times(i-1)}$, $F_i^*(S) \in \mathbf{C}^{(i-1)\times 1}$ and scalar $H_{ii}(S) \in \mathbf{C}[s]$ can be computed from $N_{i-1}(S)$, $l(S)$, $n_{ii}(S)$, $\widetilde{N}_{i-1}(S)$ and $\breve{N}_{i-1}(S)$ using exact recurrence relations.

Proof. As in the proof of the previous theorem we will use induction and Lemma 2.2 (Algorithm 2.2). The case $i = 1$ is again trivial and we have:

$$\widetilde{N}_1(S) = 1, \qquad \breve{N}_1(S) = n_{11}(S).$$

Let us consider now the inductive step and suppose that $N_{i-1}^{-1}(S) = \frac{\widetilde{N}_{i-1}(S)}{\breve{N}_{i-1}(S)}$. From the relation (2.10) we have:

$$\frac{1}{H_{ii}(S)} = n_{ii}(S) - l_i^*(S)\frac{\widetilde{N}_{i-1}(S)}{\breve{N}_{i-1}(S)}l_i(S)$$

$$= \frac{\breve{N}_{i-1}(S)n_{ii}(S) - l_i^*(S)\widetilde{N}_{i-1}(S)l_i(S)}{\breve{N}_{i-1}(S)} = \frac{\breve{H}_i(S)}{\breve{N}_{i-1}(S)}. \tag{3.5}$$

Therefore, we can write $H_{ii}(S) = \frac{\breve{N}_{i-1}(S)}{\breve{H}_i(S)}$. Using the relation (2.11) we can represent $f_i(S)$ in the following way:

$$f_i(S) = -\frac{\widetilde{N}_{i-1}(S)}{\breve{H}_i(S)} \cdot l_i^*(S) \cdot \frac{\widetilde{N}_{i-1}(S)}{\breve{N}_{i-1}(S)} = -\frac{l_i^*(S)\widetilde{N}_{i-1}(S)}{\breve{H}_i(S)} = \frac{\widetilde{F}_i(S)}{\breve{H}_i(S)}.$$

Furthermore using the fact that $\widetilde{N}_{i-1}(S)$ is symmetric and positive definite, we can conclude that $\widetilde{F}_i^*(S) = \widetilde{N}_{i-1}(S)l_i(S)$ which further implies that:

$$f_i^*(S) = \frac{\widetilde{F}_i^*(S)}{\breve{H}_i^*(S)} = \frac{\widetilde{N}_{i-1}(S)l_i(S)}{\breve{H}_i(S)}.$$

We also used that $\breve{H}_i(S) = \breve{H}_i^*(S)$ which can be easily proven from (3.5). From (2.12) we can conclude:

$$E_{i-1} = \frac{\widetilde{N}_{i-1}(S)}{\breve{N}_{i-1}(S)} + \frac{\breve{H}_i(S)}{\breve{N}_{i-1}(S)}\frac{\widetilde{F}_i(S)}{\breve{H}_i(S)}\frac{\widetilde{F}_i^*(S)}{\breve{H}_i(S)}$$

$$= \frac{\widetilde{N}_{i-1}(S) - \widetilde{F}_i(S)\widetilde{F}_i^*(S)}{\breve{N}_{i-1}(S)\breve{H}_i(S)} = \frac{\widetilde{E}_{i-1}(S)}{\breve{N}_{i-1}(S)\breve{H}_i(S)}.$$

Finally, we can represent $N_i^{-1}(S)$ in the following matrix form:

$$N_i^{-1}(S) = \begin{bmatrix} \dfrac{\widetilde{E}_{i-1}(S)}{\breve{N}_{i-1}(S)\breve{H}_i(S)} & \dfrac{\widetilde{F}_i(S)}{\breve{H}_i(S)} \\ \dfrac{\widetilde{F}_i^*(S)}{\breve{H}_i(S)} & \dfrac{\breve{N}_{i-1}(S)}{\breve{H}_i(S)} \end{bmatrix} = \frac{1}{\breve{H}_i(S)\breve{N}_{i-1}(S)} \begin{bmatrix} \widetilde{E}_{i-1}(S) & \breve{N}_{i-1}(S)\widetilde{F}_i(S) \\ \breve{N}_{i-1}(S)\widetilde{F}_i^*(S) & \breve{N}_{i-1}(S)^2 \end{bmatrix} = \frac{\widetilde{N}_i(S)}{\breve{N}_i(S)}.$$

This completes the proof of the theorem. □

Now it is easy to construct corresponding algorithms from the Theorems 3.1 and 3.2.

4. Effective method

In practice we often work with polynomial matrices $A(S)$ with a relatively small number of nonzero coefficients. In that case, the previous algorithm is not effective because many operations are redundant. To avoid this problem we will construct two appropriate sparse structures for the representation of the polynomial matrix $A(S)$ and the corresponding effective algorithm for computing $A^{\dagger}_{MN}(S)$. The first sparse representation is denoted by **Eff** and its improvement by **Eff′**, while the second structure is denoted by **Ef**.

The main idea in the first considered sparse structure is to exploit only nonzero coefficient matrices $A_I = A_{i_1,\ldots,i_{2p}} \neq 0$ of the polynomial matrix $A(S)$ given in the form (3.1).

Definition 4.1. The *effective sparse structure* of the polynomial matrix $A(S)$, defined in (3.1), is equal to:

$$\mathbf{Eff}_A = \{(J, A_J) \mid A_J \neq 0, 0 \leq J \leq \deg A(S)\}. \tag{4.1}$$

Also define *the index set* of this effective structure by:

$$\mathrm{Ind}_A = \{J \mid A_J \neq 0, 0 \leq J \leq \deg A(S)\}. \tag{4.2}$$

Define operations $+, -, \cdot$ and * on sparse structures as:

$$\begin{aligned} &\mathbf{Eff}_A + \mathbf{Eff}_B = \mathbf{Eff}_{A+B}, \qquad \mathbf{Eff}_A - \mathbf{Eff}_B = \mathbf{Eff}_{A-B}, \\ &\mathbf{Eff}_A \cdot \mathbf{Eff}_B = \mathbf{Eff}_{A\cdot B}, \qquad \mathbf{Eff}_A^* = \mathbf{Eff}_{A^*}. \end{aligned} \tag{4.3}$$

Denote by $e_A = |\mathbf{Eff}_A| = |\mathrm{Ind}_A|$ the *size* of the structure $\mathbf{Eff}_A$.

Obviously we have

$$A(S) \cdot B(S) = \sum_{\substack{I \in \mathrm{Ind}_A \\ J \in \mathrm{Ind}_B}} A_I B_J S^{I+J},$$

where

$$S^{I+J} = s_1^{i_1+j_1} \ldots s_{2p}^{i_{2p}+j_{2p}}.$$

If $C(S) = A(S)B(S)$ then the elements of $\mathbf{Eff}_C$ are pairs (K, C_K) where C_K is defined as the following sum of matrix products:

$$C_K = \sum_{\substack{I \in \mathrm{Ind}_A, \\ K-I \in \mathrm{Ind}_B}} A_I B_{K-I}, \tag{4.4}$$

where $C_K \neq 0$. Therefore it holds for $e_C \leq e_A + e_B$ and $\mathbf{Eff}_C = \mathbf{Eff}_A \cdot \mathbf{Eff}_B$ can be computed in the time $O(e_A \cdot e_B)$.

Similarly it holds for computing the sum $C(S) = A(S) + B(S)$. Elements of $\mathbf{Eff}_C$ are pairs (K, C_K) where values C_K are defined by

$$C_K = \begin{cases} A_K, & A_K \neq 0,\ B_K = 0 \\ B_K, & B_K \neq 0,\ A_K = 0 \\ A_K + B_K, & A_K \neq 0,\ B_K \neq 0 \end{cases} \tag{4.5}$$

and satisfy $C_K \neq 0$. As in the previous case we can conclude that $e_C \leq \max\{e_A, e_B\}$ and $\mathbf{Eff}_C$ can be computed in time $O(\max\{e_A, e_B\})$.

Index sets corresponding to addition and multiplication of sparse matrices are equal to:

$$\mathrm{Ind}_{A+B} = \mathrm{Ind}_A \cup \mathrm{Ind}_B, \qquad \mathrm{Ind}_{AB} = \mathrm{Ind}_A + \mathrm{Ind}_B.$$

In view of (4.3), we compute $\mathbf{Eff}_A^* = \{(I, A_I^*) \mid (I, A_I) \in \mathbf{Eff}_A\}$ in time $O(e_A)$.

Usually, coefficient matrices A_I in the polynomial representation (3.1), i.e. in the sparse representation (4.1) are sparse. Using this fact we can significantly improve our sparse structure **Eff** by using an appropriate structure for these constant coefficient matrices.

Definition 4.2. For the constant matrix $A = [a_{ij}] \in \mathbf{C}^{m\times n}$, denote the following sparse structure:

$$\mathbf{Sp}_A = \left\{(i, j, a_{ij}) \mid a_{ij} \neq 0\right\}. \tag{4.6}$$

Denote by $s_A = |\mathbf{Sp}_A|$ the size of the structure $\mathbf{Sp}_A$.

Similarly as in the case of $\mathbf{Eff}_A$, we can define elementary operations on these sparse structures:

$$\mathbf{Sp}_A + \mathbf{Sp}_B = \{(i, j, a_{ij} + b_{ij}) \mid (i, j, a_{ij}) \in \mathbf{Sp}_A \vee (i, j, b_{ij}) \in \mathbf{Sp}_B,\ a_{ij} + b_{ij} \neq 0\}$$
$$\mathbf{Sp}_A \cdot \mathbf{Sp}_B = \left\{(i, j, c_{ij}) \mid c_{ij} = \sum a_{ik}b_{kj} \neq 0, (i, k, a_{ik}) \in \mathbf{Sp}_A \wedge (k, j, b_{kj}) \in \mathbf{Sp}_B\right\}$$
$$\mathbf{Sp}_A^* = \{(j, i, a_{ij}^*) \mid (i, j, a_{ij}) \in \mathbf{Sp}_A\}.$$

In this way, we have the following improvement of the structure **Eff**:

$$\begin{aligned} \mathbf{Eff}'_A &= \left\{(J, \mathbf{Sp}_{A_J}) \mid A_J \neq 0,\ 0 \leq J \leq \deg A(S)\right\} \\ &= \left\{\left(J, \{i, j, (A_J)_{ij} \mid (A_J)_{ij} \neq 0\}\right) \mid A_J \neq 0,\ 0 \leq J \leq \deg A(S)\right\}. \end{aligned} \tag{4.7}$$

It can be seen that the complexity of computing $\mathbf{Sp}_A + \mathbf{Sp}_B$ is $O(s_A + s_B)$ and for $\mathbf{Sp}_A^*$ is $O(s_A)$. In the case of multiplication the complexity depends on concrete implementation. Suppose that $A \in \mathbf{C}^{m\times n}$ and $B \in \mathbf{C}^{n\times p}$. If the triples are sorted lexicographically in $\mathbf{Sp}_A$ and in $\mathbf{Sp}_B$ then for every $(i, k, A_{ik}) \in \mathbf{Sp}_A$ we need to find all $(k, j, B_{kj}) \in \mathbf{Sp}_B$, i.e. all triples in $\mathbf{Sp}_B$ which begin by k. If we denote this number by $s_B^{(k)}$:

$$s_B^{(k)} = |\{(k, j, b_{kj}) \in \mathbf{Sp}_B \mid j = 1, \ldots, p\}|$$

then the complexity of multiplication $\mathbf{Sp}_A \cdot \mathbf{Sp}_B$ is:

$$O\left(\sum_{(i,k,a_{ik})\in\mathbf{Sp}_A} s_B^{(k)} + m \cdot p\right). \tag{4.8}$$

The last addend in (4.8) comes from the fact that we need to construct the sparse structure $\mathbf{Sp}_C$ for the matrix $C = AB \in \mathbf{C}^{m\times p}$.

We implemented the sparse structure **Sp** in MATHEMATICA as the structure `SparseArray`. MATHEMATICA offers a sparse representation for matrices, vectors, and tensors with `SparseArray` [29,30]. Both of the expressions

`SparseArray`$[\{i_1, j_1\} \to v_1, \{i_2, j_2\} \to v_2, \ldots,]$
`SparseArray`$[\{\{i_1, j_1\}, \{i_2, j_2\}, \ldots\} \to \{v_1, v_2, \ldots\}]$

represent the `SparseArray` with elements in positions $\{i_k, j_k\}$ having values v_k.

Operations on sparse matrices are all equivalent to the operations on dense matrices [29,30]: `Plus`(`+`) for matrix addition, `Dot` (`.`) for matrix multiplication, `Times` (`*`) for multiplication by scalar, etc.

Therefore, in our implementation we have

$$\mathbf{Eff}'_A = \{(J, \texttt{SparseArray}[A_J]) \mid A_J \neq 0, 0 \leq J \leq \deg A(S)\}.$$

The shown fact that basic operations are the same for dense and sparse matrices allows us to use the same procedures for basic operations on **Eff** in cases when **Sp** is embedded in **Eff** and when it is not. In procedural programming of languages we can decide to use **Sp** or not in the beginning of algorithm, depending on the structure of input matrices $A(S)$, $M(S)$ and $N(S)$. Similarly, it is possible to change the choice of one between these two variants of the structure **Eff** during the algorithm implementation.

In the second type of the sparse structure for polynomial matrices we represent the matrix $A(S)$ in the form $A(S) = [a_{ij}(S)]$, where $a_{ij}(S)$ are scalar polynomials, and construct effective sparse structures $\mathbf{Eff}_{a_{ij}}$ for each $a_{ij}(S)$. Effective structure $\mathbf{Eff}_a$ for the scalar polynomial $a(S) = \sum_{I=1}^{\deg a(S)} a_I S^I$ is defined similarly as in the matrix case (4.1):

$$\mathbf{Eff}_a = \{(J, a_J) \mid a_J \neq 0, 1 \leq J \leq \deg a(S)\}.$$

Such sparse representation we denote by **Ef**, and have $\mathbf{Ef}_A = [\mathbf{Eff}_{a_{ij}}]$. If we use notations $\mathbf{ef}_A = \sum_{i=1}^{m} \sum_{j=1}^{n} e_{a_{ij}}$, then the complexity for the addition is

$$O\left(\sum_{i=1}^{m} \sum_{j=1}^{n} e_{a_{ij}} + e_{b_{ij}}\right) = O(\mathbf{ef}_A + \mathbf{ef}_B).$$

After the notations $\mathbf{row}(B, k) = \sum_{j=1}^{p} e_{b_{kj}}$ and $\mathbf{col}(A, k) = \sum_{i=1}^{m} e_{a_{ik}}$ we conclude that the complexity of the matrix multiplication $C = AB$ is equal to

$$\begin{aligned} O\left(\sum_{i=1}^{m} \sum_{k=1}^{n} \sum_{j=1}^{p} e_{a_{ik}} e_{b_{kj}}\right) &= O\left(\sum_{k=1}^{n} \sum_{i=1}^{m} e_{a_{ik}} \mathbf{row}(A, k)\right) \\ &= O\left(\sum_{k=1}^{n} \mathbf{col}(A, k) \mathbf{row}(B, k)\right) \end{aligned}$$

for the multiplication.

Polynomials in MATHEMATICA are represented in the internal form using the little modified **Ef** sparse structure. For example, two-variable polynomial $p(s_1, s_2) = 4s_1^9 s_2^{10} + s_2^3 + s_1^2 s_2^2 + 3s_1^3 s_2 + s_2 + 2s_1^2 + 3s_1 + 10$ is represented in the following MATHEMATICA internal form:

```
Plus[
10,
Times[3, s1],
Times[2, Power[s1, 2]],
s2,
Times[3, Power[s1, 3], s2],
Times[Power[s1, 2], Power[s2, 2]],
Power[s2, 3],
Times[4, Power[s1, 9], Power[s2, 10]]
].
```

The last expression is obtained by using MATHEMATICA function `FullForm[E]` which returns an internal representation of the expression E [29,30]. This internal form of the polynomial $p(S)$, at the top level is the list with length $\mathbf{ef}_p$ with the head `Plus`. Each element of this list contains the exponent $J = (j_1, j_2)$ and the value p_J (values $j_1 = 0, 1$ and $j_2 = 0, 1$ and are not shown), hence the length of each element is $O(1)$. Also the size of whole structure is $O(\mathbf{ef}_{p(s)})$. Therefore, we can use this natural polynomial representation in MATHEMATICA and built-in elementary operators to implement the effective partitioning method using the **Ef** structure. The complexity of these built-in operations are the same as the corresponding operations defined for **Ef** structure.

The next algorithm is the effective partitioning method for computing the weighted Moore–Penrose inverse of polynomial matrices, suitable for sparse matrices. Generally, the same method can be used with both two presented sparse structures. Therefore, we will denote *general sparse structure* with $\mathcal{E}$, which can be exchanged either by **Eff** or **Ef**. Also by $\mathcal{O}$ we will denote the general effective structure of an appropriate zero matrix. We will use the same symbol for the effective structure of the number 0.

Algorithm 4.1 (*Computing the Weighted Moore–Penrose Inverse* $A(S)^{\dagger}_{M(S),N(S)}$ *of Sparse Matrix* $A(S)$). Input: Effective structures of matrices $A(S)$, $M(S)$, $N(S)$.

Step 1. In the case $\mathcal{E}_{a_1} \neq \mathcal{O}$ compute initial values:

$$\mathcal{E}_{Z_1} = \mathcal{E}^*_{a_1} \cdot \mathcal{E}_M, \qquad \mathcal{E}_{Y_1} = \mathcal{E}^*_{a_1} \cdot \mathcal{E}_M \cdot \mathcal{E}_{a_1}.$$

If $\mathcal{E}_{a_1} = \mathcal{O}$, then set $\mathcal{E}_{Z_1} = \mathcal{O}$ and $\mathcal{E}_{Y_1} = \mathcal{E}_1$, where $\mathcal{E}_1$ is the corresponding sparse structure of the number 1.

Step 2. Recursive step: For $i = 2, \ldots, n$ perform the following steps

Step 2.1 Compute: $\mathcal{E}_{d_i} = \mathcal{E}_{Z_{i-1}} \cdot \mathcal{E}_{a_i}$

Step 2.2 Compute: $\mathcal{E}_{c_i} = \mathcal{E}_{a_i} \cdot \mathcal{E}_{Y_{i-1}} - \mathcal{E}_{A_{i-1}} \cdot \mathcal{E}_{d_i}$

Step 2.3 If $\mathcal{E}_{c_i} \neq \mathcal{O}$ then compute $\mathcal{E}_{V_i}$ and $\mathcal{E}_{W_i}$ using

$$\mathcal{E}_{V_i} = \mathcal{E}_{Y_{i-1}} \cdot \mathcal{E}^*_{c_i} \cdot \mathcal{E}_M$$
$$\mathcal{E}_{W_i} = \mathcal{E}^*_{c_i} \cdot \mathcal{E}_M \cdot \mathcal{E}_{c_i}.$$

Otherwise use the following formulae:

$$\mathcal{E}_{V_i} = \mathcal{E}_{\breve{\Delta}_i} \cdot (\mathcal{E}^*_{d_i} \cdot \mathcal{E}_{N_{i-1}} - \mathcal{E}^*_{l_i} \cdot \mathcal{E}_{Y_{i-1}}) \cdot \mathcal{E}_{Z_{i-1}}$$
$$\mathcal{E}_{W_i} = \mathcal{E}_{\widetilde{\Delta}_i} \cdot \mathcal{E}^*_{Y_{i-1}} \cdot \mathcal{E}_{Y_{i-1}},$$

where the structures $\breve{\Delta}_i$ and $\widetilde{\Delta}_i$ are defined by:

$$\mathcal{E}_{\breve{\Delta}_i} = \mathcal{E}_{\psi_i} \cdot \mathcal{E}^*_{Y_{i-1}} \cdot \mathcal{E}_{Y_{i-1}}$$
$$\mathcal{E}_{\widetilde{\Delta}_i} = (\mathcal{E}_{Y_{i-1}} \cdot (\mathcal{E}_{n_{ii}} \mathcal{E}_{Y_{i-1}} - \mathcal{E}_{d^*_i} \cdot \mathcal{E}_{l_i} - \mathcal{E}_{l^*_i} \cdot \mathcal{E}_{d_i}) + \mathcal{E}_{d^*} \cdot \mathcal{E}_{N_{i-1}} \cdot \mathcal{E}_{D_i}) \cdot \mathcal{E}_{\breve{N}_{i-1}} - \mathcal{E}_{Y_{i-1}} \cdot \mathcal{E}_{l^*_i} \cdot \mathcal{E}_{\varphi_i}.$$

We used sparse representations for temporary variables φ_i and ψ_i, defined in (3.4):

$$\mathcal{E}_{\varphi_i} = (\mathcal{E}_{Y_{i-1}} \cdot \mathcal{E}_I - \mathcal{E}_{Z_{i-1}} \cdot \mathcal{E}_{A_{i-1}}) \cdot \mathcal{E}_{\widetilde{N}_{i-1}} \cdot \mathcal{E}_{l_i}$$
$$\mathcal{E}_{\psi_i} = \mathcal{E}_{Y_{i-1}} \cdot \mathcal{E}_{\breve{N}_{i-1}}.$$

Step 2.4. Now compute $\mathcal{E}_{Z_i}$ and $\mathcal{E}_{Y_i}$ using:

$$Z_i = \begin{bmatrix} \Theta_i \\ \Psi_i \end{bmatrix}, \qquad \mathcal{E}_{Y_i} = \mathcal{E}_{\psi_i} \cdot \mathcal{E}_{W_i}.$$

Structures $\mathcal{E}_{\Theta_i}$ and $\mathcal{E}_{\Psi_i}$ are defined by:

$$\mathcal{E}_{\Theta_i} = \mathcal{E}_{Z_{i-1}} \cdot \mathcal{E}_{\breve{N}_{i-1}} \cdot \mathcal{E}_{W_i} - \mathcal{E}_{d_i} \cdot \mathcal{E}_{\breve{N}_{i-1}} \cdot \mathcal{E}_{V_i} - \mathcal{E}_{\varphi_i} \mathcal{E}_{V_i}$$
$$\mathcal{E}_{\Psi_i} = \mathcal{E}_{\psi_i} \cdot \mathcal{E}_{V_i}.$$

If we use **Ef** or **Eff** sparse structure, $\mathcal{E}_{Z_i}$ is equal respectively to:

$$\mathbf{Ef}_{Z_i} = \begin{bmatrix} \mathbf{Ef}_{\Theta_i} \\ \mathbf{Ef}_{\Psi_i} \end{bmatrix}$$

$$\mathbf{Eff}_{Z_i} = \left\{ \left(j, \begin{bmatrix} (\Theta_i)_j \\ (\Psi_i)_j \end{bmatrix} \right) \middle| (j, (\Theta_i)_j) \in \mathbf{Eff}_{\Theta_i}, (j, (\Psi_i)_j) \in \mathbf{Eff}_{\Psi_i} \right\} \cup \left\{ \left(j, \begin{bmatrix} (\Theta_i)_j \\ 0 \end{bmatrix} \right) \middle| (j, (\Theta_i)_j) \in \mathbf{Eff}_{\Theta_i}, (\Psi_i)_j = 0 \right\} \cup \left\{ \left(j, \begin{bmatrix} 0 \\ (\Psi_i)_j \end{bmatrix} \right) \middle| (\Theta_i)_j = 0, (j, (\Psi_i)_j) \in \mathbf{Eff}_{\Psi_i} \right\}. \tag{4.9}$$

Step 2.5. Find the polynomials $Z_i(S)$ and $Y_i(S)$ from its effective structures and compute:

$$X_i(S) = \frac{Z_i(S)}{Y_i(S)}, \tag{4.10}$$

Cancel the common multipliers in numerator $Z_i(S)$ and denominator $Y_i(S)$, recompute (if necessary) effective structures and continue with the next i.

Step 3. The stopping criterion is $i = n$. In this case is $A^{\dagger}_{M(S),N(S)}(S) = X_n(S)$.

Similarly we can derive a modification of the method introduced in Theorem 3.2 for computing the inverse matrix $N_i^{-1}(S)$ in the polynomial form:

$$N_i^{-1}(S) = \frac{\widetilde{N}_i(S)}{\breve{N}_i(S)}. \tag{4.11}$$

Algorithm 4.2 (*Effective Computation of* $N_i^{-1}(S)$*, For* $i = 1, \ldots, n$). Input: Effective structure of positive definite Hermitian polynomial matrix $N(S)$ of the order n. Notations are the same as in Theorem 3.2.

Step 1. Generate initial values: $\widetilde{N}_1 = I$ and $\breve{N}_1 = n_{11}$ and corresponding effective structures.
Step 2. Recursive step: For $i = 2, \ldots, n$ perform following steps:
Step 2.1. Compute: $\mathcal{E}_{\breve{H}_i} = \mathcal{E}_{n_{ii}} \cdot \mathcal{E}_{\breve{N}_{i-1}} - \mathcal{E}^*_{l_i} \cdot \mathcal{E}_{\widetilde{N}_{i-1}} \cdot \mathcal{E}_{l_i}$.
Step 2.2. Compute: $\mathcal{E}_{\widetilde{F}_i} = -\mathcal{E}_{\widetilde{N}_{i-1}} \cdot \mathcal{E}_{l_i}$.
Step 2.3. Compute: $\mathcal{E}_{\widetilde{E}_i} = \mathcal{E}_{\widetilde{N}_{i-1}} - \mathcal{E}_{F_i} \cdot \mathcal{E}^*_{F_i}$.
Step 2.4. Generate:

$$\widetilde{N}_i(S) = \begin{bmatrix} \widetilde{E}_{i-1}(S) & \breve{N}_{i-1}(S) \cdot \widetilde{F}_i(S) \\ \breve{N}_{i-1}(S) \cdot \widetilde{F}_i^*(S) & \breve{N}^2_{i-1}(S) \end{bmatrix}$$

$$\breve{N}_i(S) = \breve{N}_{i-1}(S) \cdot \breve{H}_i(S).$$

As in the previous algorithm, we have also two different representations for **Ef** and **Eff** sparse structures. These relations are similar to (4.8).
Step 3. Stop criterion for $i = n$. Inverse matrix $N_k^{-1}(S)$, for every $k = 1, \ldots, n$ is equal to:

$$N_k^{-1}(S) = \frac{\widetilde{N}_k(S)}{\breve{N}_k(S)}. \tag{4.12}$$

5. Examples

We implemented Algorithms 2.1, 2.2, 4.1 and 4.2 in the programming language MATHEMATICA. An implementation of the **Eff** sparse structure is also made. Functions `WPolyEf` and `WPolyEff` implement Algorithm 4.1 using respectively **Ef** and **Eff** sparse structures. All basic operations for **Eff** sparse structure (functions `Add`, `Sub`, `Muls`, `Mul` and `TE` corresponding to the addition, subtraction, multiplication by scalar, multiplication and conjugate-transposition respectively) are also implemented.

Example 5.1. Let us find the weighted Moore–Penrose inverse of the following two-variable polynomial matrix $A(x, y)$:

$$A(x, y) = \begin{bmatrix} 1 - 3x & 5 + 9x - 10y & 16 + 8x + 2y \\ -7 + 9x - 8y & 8 + 5x - y & 4 + 2x + 3y \\ 7 - x - 8y & 16 - 2x - 6y & -3 - 2x - 4y \end{bmatrix}$$

with respect to the following matrices $M(x, y)$ and $N(x, y)$:

$$M(x, y) = \begin{bmatrix} -20 - x - \overline{x} & -8 - 7x - 4\overline{x} & -2(8 + 3x + 4\overline{x}) \\ -8 - 4x - 7\overline{x} & -20 + 7x + 7\overline{x} & 2(5x - \overline{x}) \\ -2(8 + 4x + 3\overline{x}) & -2(x - 5\overline{x}) & 7(-2 + x + \overline{x}) \end{bmatrix}$$

$$N(x, y) = \begin{bmatrix} 16 + 7x + 7\overline{x} & 7 - 6x - 2\overline{x} & 6 - 10x - 3\overline{x} \\ 7 - 2x - 6\overline{x} & -2(3 + 5x + 5\overline{x}) & -2(6 + 4x + 3\overline{x}) \\ 6 - 3x - 10\overline{x} & -2(6 + 3x + 4\overline{x}) & -3(-6 + x + \overline{x}) \end{bmatrix}.$$

The obtained weighted Moore–Penrose inverse is:

$$X(x,y) = A^{\dagger}_{M(x,y),N(x,y)}(x,y) = \left(60x^3 - 5yx^2 - 540x^2 + 51yx + 779x - 42y - 435\right)^{-1} \times \begin{bmatrix} -5x^2 + 51x - 42 & -3x^2 + 8x - 13 & -3x^2 + 33x - 4 \\ -30x^2 + 71x + 15 & 42x^2 - 5yx - 33x + y + 15 & -18x^2 - 63x + 10y + 105 \\ -2\left(10x^2 - 19x + 12\right) & 2\left(18x^2 - 2yx - 29x + 2y + 17\right) & -24x^2 + yx + 42x - 2y - 23 \end{bmatrix}.$$

Let us notice that degrees of intermediate results in Algorithms 4.1 and 4.2 are much greater than the degrees of A, M, N and X (maximum degree in this example are 874 and 122 of the variables x and y respectively). This is the reason why the algorithms for computing the weighted Moore–Penrose inverse for polynomial matrices are very slow (working time of the function `WPolyEff` for the last example is 172.922 s). As we will see in the sequel, when matrices A, M and N are sparse, corresponding intermediate results are also sparse. Therefore, sparse structures introduced in the previous section improve the working time of the implementation.

Algorithm 4.1 is tested on several random generated test examples. We tested variants of Algorithm 4.1 using **Ef** and **Eff** sparse structures separately. In this test, matrices $A(S)$, $M(S)$ and $N(S)$ were complex polynomial matrices of one variable s (i.e. they hold for $S = (s, \overline{s})$).

We did the testing for two different classes of matrices: sparse and dense. The measures representing sparsity of a given polynomial matrix are the same as in [19] (Definitions 6.1 and 6.2). We are now restating these two definitions and generalizing them to the multi-variable complex polynomial matrices.

Definition 5.1. For a given matrix $A(S) = [a_{ij}(S)] \in \mathbf{C}[S]^{m\times n}$ (polynomial or constant), **the first sparse number** $sp_1(A)$ is the ratio of the total number of nonzero elements and total number of elements in $A(S)$:

$$sp_1(A(S)) = \frac{\left|\{(i,j) \mid a_{ij}(S) \neq 0\}\right|}{m \cdot n}.$$

The first sparse number represents the density of nonzero elements and it is between 0 and 1.

m	n	d	Algorithm 4.1 with **Ef**	Algorithm 4.1 with **Eff**	m	n	d	Algorithm 4.1 with **Ef**	Algorithm 4.1 with **Eff**
$sp_1(A(S)) = 0.9, sp_2(A(S)) = 0.9$					$sp_1(A(S)) = 0.7, sp_2(A(S)) = 0.5$				
2	2	1	0.14	0.188	2	2	1	0.06	0.89
2	2	2	0.65	1.24	2	2	2	0.25	0.46
2	2	3	1.92	3.93	2	2	3	0.60	1.23
3	3	1	1.34	1.32	3	3	1	0.47	0.68
3	3	2	9.01	11.81	3	3	2	4.60	7.18
3	3	3	34.39	48.13	3	3	3	14.89	24.65
4	4	1	7.87	6.74	4	4	1	6.10	6.18
4	4	2	69.31	64.48	4	4	2	34.95	39.68
4	4	3	461.07	594.98	4	4	3	256.31	299.61
5	5	1	49.13	58.48	5	5	1	30.85	39.43
5	5	2	309.38	330.32	5	5	2	246.32	283.12
$sp_1(A(S)) = 1, sp_2(A(S)) = 0.2$					$sp_1(A(S)) = 0.2, sp_2(A(S)) = 0.2$				
2	2	1	0.04	0.112	2	2	1	0.032	0.105
2	2	2	0.11	0.263	2	2	2	0.069	0.190
2	2	3	0.422	1.303	2	2	3	0.187	0.713
3	3	1	0.281	0.972	3	3	1	0.185	0.675
3	3	2	1.367	3.505	3	3	2	0.628	2.944
3	3	3	5.808	18.449	3	3	3	1.031	3.275
4	4	1	1.613	5.549	4	4	1	0.987	4.344
4	4	2	12.134	27.113	4	4	2	6.087	25.263
4	4	3	55.139	107.27	4	4	3	27.466	176.581
5	5	1	7.475	13.582	5	5	1	3.294	15.853
5	5	2	84.712	139.681	5	5	2	42.159	171.416

Definition 5.2. For a given polynomial matrix $A(S) \in \mathbf{C}[S]^{m\times n}$ and $S = (s_1, \ldots, s_p)$, **the second sparse number** $sp_2(A(S))$ is the following ratio:

$$sp_2(A(S)) = \frac{\#\{(i, j, k_1, \ldots, k_p) \mid 0 \le k_j \le \deg\limits_{s_j} A(S), \operatorname{Coef}(a_{ij}(S), s_1^{k_1}\cdots s_p^{k_p}) \neq 0\}}{\deg\limits_{s_1} A \cdots \deg\limits_{s_p} A \cdot m \cdot n}.$$

By $\operatorname{Coef}(P(S), s_1^{k_1}\cdots s_p^{k_p})$ we denoted the coefficient corresponding to $s_1^{k_1}\cdots s_p^{k_p}$ in polynomial $P(S)$.

The second sparse number represents density of nonzero coefficients contained in elements $a_{ij}(S)$, and it is also between 0 and 1.

Results are presented in the next table (column d states for the degree of corresponding matrix polynomials $A(S)$, $M(S)$ and $N(S)$):

All presented processor times are in seconds and the sparse numbers for matrices $M(S)$ and $N(S)$ are the same as the corresponding sparse numbers for $A(S)$. Every processor time is obtained by averaging working times of 15 different randomly generated test cases. Testing was done on an Intel Pentium 4 processor at 2.6 GHz and MATHEMATICA 5.2. We can notice that Algorithm 4.1 with an **Ef** structure showed best timings on all test cases. We have already mentioned that an **Ef** sparse structure is already implemented in MATHEMATICA. In the implementation we used standard built-in operators for manipulation with matrices in **Ef** structure.

The first table (when $sp_1(A(S)) = sp_2(A(S)) = 0.9$) corresponds to dense matrices. In this case, sparse structures are not so effective because there are a lot of nonzero elements in all matrices and nonzero coefficients in polynomials. But we can notice a significant improvement in working time when **Ef** structure is applied against the case when **Eff** structure is applied. This difference mainly comes from the fact that **Ef** structure is implemented by MATHEMATICA built-in operations.

m	n	d	Algorithm 2.1	Algorithm 4.1 with **Eff**	Algorithm 4.1 with **Ef**	Algorithm 3.1 from [1]
$sp_1(A(S)) = 0.7, sp_2(A(S)) = 0.7$						
3	3	1	0.32	0.23	0.10	0.94
3	3	2	0.69	0.57	0.20	1.32
3	3	3	0.82	1.17	0.43	1.84
3	3	4	1.19	2.15	0.73	2.38
4	3	1	0.76	1.26	0.14	1.29
4	3	2	1.29	0.65	0.31	2.12
4	3	3	2.14	1.32	0.59	2.42
4	3	4	2.84	2.26	1.01	2.93
5	5	1	3.48	1.45	1.01	3.56
5	5	2	5.90	4.54	2.92	4.92
5	5	3	9.18	8.79	6.82	8.27
5	5	4	12.15	15.87	10.85	10.34
6	6	1	7.98	2.65	2.17	8.16
6	6	2	12.93	8.20	7.31	11.32
6	6	3	21.76	18.29	13.53	19.42

The second case (when $sp_1(A(S)) = 0.7$ and $sp_2(A(S)) = 0.5$) represents sparse matrices. We can notice that working times are significantly less than in the first case. Also here **Ef** structure produces less working times than **Eff**.

In the third and fourth cases (when $sp_1(A(S)) = 1$ and $sp_2(A(S)) = 0.2$, and $sp_1(A(S)) = sp_2(A(S)) = 0.2$, respectively) we deal with matrices whose entries are very sparse polynomials. Moreover, in the fourth case we work with matrices with only few nonzero elements. In the fourth case, smallest average working times are obtained for all considered matrix dimensions and degrees. Also we can notice that as sparse numbers decrease, the average working times also decrease (for constant matrix dimensions and degree). This holds for both sparse structures and verifies the theoretical results about sparse structures **Ef** and **Eff** in practice.

We also considered a simpler case: when all input matrices ($A(S)$, $M(S)$ and $N(S)$) and variables $s_1, \ldots, s_p$ are assumed to be real. In that case we have only p variables and conjugate-transpose operation reduces only to transpose. We also should suppose that matrices $M(S)$ and $N(S)$ are symmetric in that sense. Algorithms 4.1 and 4.2 remain

the same except that we should change the definition of conjugate-transpose operator (also the implementations in MATHEMATICA). This case is considered in [1] and Algorithms 4.1 and 4.2 are effective versions of corresponding Algorithms 3.1 and 3.2 in [1]. Here working times of the algorithms are significantly less, and also the inverses have much smaller degrees. Results obtained in this special case are presented in the following table:

It can be seen from the table that here in all cases **Ef** structure was better than **Eff** (both with using Algorithm 4.1). Both effective algorithms were significantly better than Algorithm 2.1 (for rational matrices) and Algorithm 3.1 from [1]. For smaller values of d, Algorithm 2.1 was better than Algorithm 3.1 from [1] due to the implementation details.

All presented results lead us to the same conclusion: the best choice for computing weighted Moore–Penrose inverse for polynomial matrices is Algorithm 4.1 with the sparse structure **Ef**.

6. Conclusion

We extend the algorithm for computing the weighted Moore–Penrose from [27] to the set of multiple-variable rational matrices with complex coefficients. We adapt the previous algorithm to the set of polynomial matrices. We consider two effective structures which make use of only nonzero addends in polynomial matrices and improve previous results on the set of sparse matrices. In the last section we presented an illustrative example and compared various algorithms.

References

[1] M.B. Tasić, P.S. Stanimirović, M.D. Petković, Symbolic computation of weighted Moore–Penrose inverse using partitioning method, Appl. Math. Comput. 189 (2007) 615–640.
[2] P.S. Stanimirović, M.B. Tasić, Partitioning method for rational and polynomial matrices, Appl. Math. Comput. 155 (2004) 137–163.
[3] M.D. Petković, P.S. Stanimirović, Symbolic computation of the Moore–Penrose inverse using partitioning method, Internat. J. Comput. Math. 82 (2005) 355–367.
[4] A. Ben-Israel, T.N.E. Grevile, Generalized Inverses, Theory and Applications, second ed., Canadian Mathematical Society, Springer, New York, 2003.
[5] S. Barnett, Leverrier's algorithm: A new proof and extensions, SIAM J. Matrix Anal. Appl. 10 (1989) 551–556.
[6] G. Fragulis, B.G. Mertzios, A.I.G. Vardulakis, Computation of the inverse of a polynomial matrix and evaluation of its Laurent expansion, Internat. J. Control 53 (1991) 431–443.
[7] N.P. Karampetakis, Generalized inverses of two-variable polynomial matrices and applications, Circuits Syst. Signal Process. 16 (1997) 439–453.
[8] N.P. Karampetakis, Computation of the generalized inverse of a polynomial matrix and applications, Linear Algebra Appl. 252 (1997) 35–60.
[9] P.S. Stanimirovic, N.P. Karampetakis, Symbolic implementation of Leverrier–Faddeev algorithm and applications, in: 8th IEEE Medit. Conference on Control and Automation, Patra, Greece, 2000.
[10] N.P. Karampetakis, P. Tzekis, On the computation of the generalized inverse of a polynomial matrix, in: 6th Medit. Symposium on New Directions in Control and Automation, 1998, pp. 1–6.
[11] J. Jones, N.P. Karampetakis, A.C. Pugh, The computation and application of the generalized inverse via Maple, J. Symbolic Comput. 25 (1998) 99–124.
[12] N.P. Karampetakis, Generalized inverses of two-variable polynomial matrices and applications, Circuits Syst. Signal Process. 16 (1997) 439–453.
[13] N.P. Karampetakis, P. Tzekis, On the computation of the generalized inverse of a polynomial matrix, IMA J. Math. Control Inform. 18 (2001) 83–97.
[14] F. Bu, Y. Wei, The algorithm for computing the Drazin inverses of two-variable polynomial matrices, Appl. Math. Comput. 147 (2004) 805–836.
[15] J. Ji, A finite algorithm for the Drazin inverse of a polynomial matrix, Appl. Math. Comput. 130 (2002) 243–251.
[16] N.P. Karampetakis, P.S. Stanimirović, On the computation of the Drazin inverse of a polynomial matrix, in: 1rst IFAC Symposium on System Structure and Control, Prague, Czech Republic, 2001.
[17] P.S. Stanimirovic, M.B. Tasić, Drazin inverse of one-variable polynomial matrices, Filomat, Niš 15 (2001) 71–78.
[18] P.S. Stanimirović, A finite algorithm for generalized inverses of polynomial and rational matrices, Appl. Math. Comput. 144 (2003) 199–214.
[19] M.D. Petković, P.S. Stanimirović, Computing generalized inverse of polynomial matrices by interpolation, Appl. Math. Comput. 172 (2006) 508–523.
[20] N.P. Karampentakis, S. Vologianidis, DFT calculation of generalized and Drazin inverse of polynomial matrix, Appl. Math. Comput. 143 (2003) 501–521.
[21] S. Vologiannidis, N.P. Karampetakis, Inverses of multivariable polynomial matrices by discrete Fourier transforms, Multidimens. Syst. Signal Process. 15 (2004) 341–361.
[22] T.N.E. Grevile, Some applications of the pseudo-inverse of matrix, SIAM Rev. 3 (1960) 15–22.
[23] S.L. Campbell, C.D. Meyer Jr., Generalized Inverses of Linear Transformations, Pitman, London, 1979.

[24] G.R. Wang, Y.L. Chen, A recursive algorithm for computing the weighted Moore–Penrose inverse $A^{\dagger}_{MN}$, J. Comput. Math. 4 (1986) 74–85.
[25] F.E. Udwadia, R.E. Kalaba, An alternative proof of the Greville formula, J. Optim. Theory Appl. 94 (1997) 23–28.
[26] Y. Fan, R. Kalaba, Dynamic programming and pseudo-inverses, Appl. Math. Comput. 139 (2003) 323–342.
[27] G.R. Wang, A new proof of Grevile's method for computing the weighted M–P inverse, J. Shangai Normal Univ. (Natural Science Edition) 3 (1985).
[28] N. Shinozaki, M. Sibuya, K. Tanabe, Numerical algorithms for the Moore–Penrose inverse of a matrix: Direct methods, Ann. Inst. Statist. Math. 24 (1) (1972) 193–203.
[29] S. Wolfram, Mathematica Book, Version 3.0, Wolfram Media and Cambridge University Press, 1996.
[30] S. Wolfram, The Mathematica Book, fourth ed., Wolfram Media, Cambridge University Press, 1999.