Effective partitioning method for computing generalized inverses and their gradients

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

University of Niš, Faculty of Science and Mathematics,

Višegradska 33, 18000 Niš, Serbia

E-mail: dexterofnis@gmail.com, milan12t@ptt.rs, pecko@pmf.ni.ac.rs

Abstract

We extend the algorithm for computing $\{1\},\{1,3\},\{1,4\}$ inverses and their gradients from [11] to the set of multiple-variable rational and polynomial matrices. An improvement of this extension, appropriate to sparse polynomial matrices with relatively small number of nonzero coefficient matrices as well as in the case when the nonzero coefficient matrices are sparse, is introduced. For that purpose, we exploit two effective structures form [6], which make use of only nonzero addends in polynomial matrices, and define their partial derivatives. Symbolic computational package MATHEMATICA is used in the implementation. Several randomly generated test matrices are tested and the CPU times required by two used effective structures are compared and discussed.

AMS Subj. Class.: 15A09, 68Q40.

Key words: Generalized inverses; differentiation; rational matrices; polynomial matrices; sparse matrices; symbolic computation.

1 Introduction

The following Penrose equations are crucial in pseudoinverses definition:

$$(1)\quad AXA=A,\quad (2)\quad XAX=X,\quad (3)\quad (AX)^T=AX,\quad (4)\quad (XA)^T=XA.$$

For a subset $\ddagger$ of the elements from the set $\{1,2,3,4\}$, the set of matrices obeying the equations determined by the set $\ddagger$ is denoted by $A\{\ddagger\}$. A matrix from $A\{\ddagger\}$ is called an $\ddagger$-inverse of A and it is denoted by $A^{\ddagger}$. The Moore-Penrose inverse is the unique matrix satisfying all the equations (1)–(4), and it is denoted by $A^{\dagger}$.

A lot of methods were proposed to compute various generalized inverses of a matrix. These include methods arising from the Cayley-Hamilton theorem, the full-rank factorization and the singular value decomposition (see for instance, [1, 15]). Greville in [2] proposed a finite recursive algorithm for determining the Moore-Penrose inverse. Due to its ability to undertake sequential computing, this method has been extensively applied in statistical inference, filtering theory, linear estimation theory, optimization and also in analytical dynamics [4]. About a decade ago, Udwadia and Kalaba gave an alternative and a simple constructive proof of Grevilles formula [12]. A generalization of Greville's method to the weighted Moore-Penrose inverse is introduced in [14]. The results in [14] are established by using a new technique.

Symbolic computation of generalized inverses and their gradients is one of the most interesting areas of computer algebra. Matrix differentiation is of considerable importance in statistics. It is especially useful in connection with the maximum likelihood estimation of the parameters in a statistical model. The maximum likelihood estimates of the model's parameters satisfy the equations (known as the likelihood equations) obtained by equating to zero the first-order partial derivatives (with respect to model's parameters) of the logarithm of the so-called likelihood function [3]. In many important cases, the likelihood function involves

*Corresponding author

the determinant and/or inverse of a matrix. The gradient of the pseudo-inverse may be needed for sensitivity analysis, optimizations or in the nonlinear least squares problems.

There is a lot of extensions of the partitioning method to sets of rational and polynomial matrices. The algorithm for the computation of the Moore-Penrose inverse of the one-variable polynomial and/or rational matrix, based on the Greville's partitioning algorithm, was introduced in [8]. The extension of results from [8] to the set of the two-variable rational and polynomial matrices is introduced in the paper [7]. The Wang's partitioning method from [14], aimed in the computation of the weighted Moore-Penrose inverse, is extended to sets of one-variable rational and polynomial matrices in the paper [9]. Also the efficient algorithm for computing the weighted Moore-Penrose inverse, appropriate for the polynomial matrices where only a few polynomial coefficients are nonzero, is established in [6]. Udwadia and Kalaba derived in [13] a constructive procedure for determining different types of generalized inverses for constant matrices. These results are extended to one-variable rational and polynomial matrices in [10].

An efficient method for direct simultaneous computation of the Moore-Penrose inverse in conjunction with its gradient is derived in [5]. Layton in [5] used the approach to simply differentiate terms arising from the Grevile's partitioning method. The resulting algorithm in [5] is usable and efficient because of its unique property that it requires only elementary matrix operations, such as addition, subtraction and multiplication. In the paper [11] the Layton's method is combined with the representation of the Moore-Penrose inverse of one-variable polynomial matrix from [8]. In consequence, the algorithm for computing the gradient of the Moore-Penrose inverse for one-variable polynomial matrix is developed. Moreover, using the representation of various types of pseudo-inverses from [10], more general algorithms for computing partial derivatives of $\{1\}$, $\{1,3\}$ and $\{1,4\}$ inverses of one-variable rational and polynomial matrices are derived in [11].

As usual, let $\mathbb{R}$ be the set of real numbers, $\mathbb{R}^{m\times n}$ be the set of complex $m\times n$ matrices, and $\mathbb{R}_r^{m\times n}=\{X\in \mathbb{R}^{m\times n} \mid \operatorname{rank}(X)=r\}$. Let $A\in\mathbb{R}(s_1,s_2,\dots,s_p)^{m\times n}$ be arbitrary multi-variable rational or polynomial matrix. In order to make a notation shorter we denote $S=(s_1,s_2,\dots,s_p)$ and write $A(S)$ instead of $A(s_1,s_2,\dots,s_p)$. Let $A_i(S)$ be the submatrix of $A(S)$ consisting of first i columns of $A(S)$. If i-th column of $A(S)$ is denoted by $a_i(S)$, then it is obvious that $A_i(S)$ is partitioned as $A_i(S)=[A_{i-1}(S)\,|\,a_i(S)]$, $i=2,\dots,n$, assuming $A_1(S)=[a_1(S)]$. The set of polynomials (resp. rational functions) with complex coefficients in the variables S are denoted by $\mathbb{R}[S]$ (resp. $\mathbb{R}(S)$). The set of $m\times n$ matrices with elements in $\mathbb{R}[S]$ (resp. $\mathbb{R}(S)$) are denoted by $\mathbb{R}[S]^{m\times n}$ (resp $\mathbb{R}(S)^{m\times n}$). Also by $\mathbf{O}$, we denote an appropriate zero matrix and by $\mathbf{0}$, appropriate zero vector.

Main results of the present article are summarized in the following.

- Algorithms from [10], which give recursive rules for computation of $A_i^{\ddagger}(s)$ in terms of $A_{i-1}^{\ddagger}(s)$, $s\in\mathbb{R}$, are extended from the single-variable polynomial matrix case to the multi-variable polynomial matrix case.

- Algorithm which gives recurrent relations between the partial derivatives $\frac{\partial A_i^{\ddagger}(S)}{\partial s_k}$ and $\frac{\partial A_{i-1}^{\ddagger}(S)}{\partial s_k}$, established in [11], is improved in the case when a great number of coefficient matrices vanishes to zero matrix as well as in the case when the nonzero coefficient matrices are sparse. In order to define effective algorithm for computing partial derivatives of generalized inverses, it is necessary to define partial derivatives of effective structures defined in the papers [6] and [7].

Generally, the present paper is a continuation of the papers [6, 7, 9, 10, 11] on multi-variable polynomial matrices and sparse matrices.

Extension of results from [10, 11] to multi-variable polynomial matrices is presented in the second section. In this way, a finite recursive algorithm for symbolic computation of generalized inverses $A_i^{\ddagger}(S)$ and their gradients is derived. Algorithms effectively applicable to sparse polynomial matrices are developed in the third section. Implementation, evaluated in the package MATHEMATICA, is exploited in development of several illustrative examples in the last section. A comparison of two used effective structures is given.

2 Multi-variable polynomial matrix case

Udwadia and Kalaba [13] proved the following theorem which gives the expressions for compute generalized inverses of partitioned matrix $A_k=[A_{k-1}\mid a_k]$ where $A_{k-1}\in\mathbb{R}^{m\times k}$, $a_k\in\mathbb{R}^{m\times 1}$ and $k\in\{2,3,\dots,n\}$.

Theorem 2.1. (Udwadia and Kalaba, 1999) *Denote by* $\ddagger$ *any of following generalized inverses:* $\{1\}$, $\{1,3\}$, $\{1,4\}$ *or* $\{1,2,3,4\}$. *Also let* $c_k = (I - A_k A_k^{\ddagger})a_k$ *and* $d_k = A_k^{\ddagger} a_k$. *Then*

$$A_k^{\ddagger} = \begin{bmatrix} A_{k-1}^{\ddagger} - d_k b_k^T \\ b_k^T \end{bmatrix}$$

where b_k *is given by:*

1. *If* $c_k = \mathbf{0}$ *then*

$$b_k = \frac{(A_{k-1}^{\ddagger})^T d_k}{1 + d_k^T d_k}, \qquad \ddagger = \{1,2,3,4\} \text{ or } \ddagger = \{1,4\},$$

and b_k *is an arbitrary vector from* $\mathbb{R}^{m\times 1}$ *if* $\ddagger = \{1,3\}$ *or* $\{1\}$.

2. *Otherwise, if* $c_k \neq \mathbf{0}$ *then*

$$b_k = \frac{c_k^T (I - A_{k-1} A_{k-1}^{\ddagger})}{c_k^T c_k}, \qquad \ddagger = \{1,4\} \text{ or } \ddagger = \{1\},$$

and

$$b_k = \frac{c_k^T}{c_k^T c_k}, \qquad \ddagger = \{1,3\} \text{ or } \ddagger = \{1,2,3,4\}$$

Previous theorem gives the recursive method for computing $A^{\ddagger} = A_n^{\ddagger}$. Note that for $k = 1$ and $a_1 \neq \mathbf{0}$ we have [10]

$$a_1^{\ddagger} = \frac{a_1^T}{a_1^T a_1}, \qquad \ddagger = \{1,3\} \text{ or } \ddagger = \{1,2,3,4\}$$

and

$$a_1^{\ddagger} = \frac{r_1^T}{r_1^T a_1}, \qquad \ddagger = \{1\} \text{ or } \ddagger = \{1,4\}.$$

Moreover, for $a_1 = \mathbf{0}$ trivially holds $a_1^{\ddagger} = \mathbf{0}$.

Based on the Theorem 2.1, Stanimirović and Tasić [10] developed an algorithm for recursive computation of generalized inverses of one variable rational or polynomial matrix $A(s)$. The computation of the derivative of $A^{\ddagger}(s)$ is shown by same authors in [11].

Generalizations of results form [10, 11] to the multi-variable polynomial or rational matrices gives the following algorithm.

Algorithm 2.1 Moore-Penrose inverse, $\{1\}, \{1,3\}, \{1,4\}$-inverses and corresponding gradients.

Require: Initial value: matrix $A(S) \in \mathbb{R}(S)^{m\times n}$, inverse $\ddagger$ and $r_1(S), \ldots, r_n(S) \in \mathbb{R}(S)^{m\times 1}$ arbitrary vectors.

if $a_1(S) = \mathbf{0}$ **then**
  $A_1^{\ddagger}(S) = a_1^T(S)$; $\quad \frac{\partial A_1^{\ddagger}(S)}{\partial s_k} = \frac{\partial a_1^T(S)}{\partial s_k}$.
else
  if $\ddagger = \{1,2,3,4\}$ or $\ddagger = \{1,3\}$ **then**
    $A_1^{\ddagger}(S) = \frac{a_1^T(S)}{a_1^T(S)a_1(S)}$;
    $\frac{\partial A_1^{\ddagger}(S)}{\partial s_k} = \frac{a_1^T(S)a_1(S)\frac{\partial a_1^T(S)}{\partial s_k} - \frac{\partial a_1^T(S)}{\partial s_k}a_1(S)a_1^T(S) - a_1^T(S)\frac{\partial a_1(S)}{\partial s_k}a_1^T(S)}{(a_1^T(S)a_1(S))^2}$.
  end if
  if $\ddagger = \{1,4\}$ or $\ddagger = \{1\}$ **then**
    $A_1^{\ddagger}(S) = \frac{r_1^T(S)}{r_1^T(S)a_1(S)}$;
    $\frac{\partial A_1^{\ddagger}(S)}{\partial s_k} = \frac{r_1^T(S)a_1(S)\frac{\partial r_1^T(S)}{\partial s_k} - \frac{\partial r_1^T(S)}{\partial s_k}a_1(S)r_1^T(S) - r_1^T(S)\frac{\partial a_1(S)}{\partial s_k}r_1^T(S)}{(r_1^T(S)a_1(S))^2}$.
  end if

end if
for $i = 2$ to n **do**
  $d_i(S) = A^{\ddagger}_{i-1}(S)a_i(S)$;
  $\frac{\partial d_i(S)}{\partial s_k} = \frac{\partial A^{\ddagger}_{i-1}(S)}{\partial s_k}a_i(S) + A^{\ddagger}_{i-1}(S)\frac{\partial a_i(S)}{\partial s_k}$;
  $c_i(S) = a_i(S) - A_{i-1}(S)d_i(S)$;
  $\frac{\partial c_i(S)}{\partial s_k} = \frac{\partial a_i(S)}{\partial s_k} - \frac{\partial A_{i-1}(S)}{\partial s_k}d_i(S) - A_{i-1}(S)\frac{\partial d_i(S)}{\partial s_k}$;
  if $c_i \neq \mathbf{0}$ **then**
    if $\ddagger = \{1,2,3,4\}$ or $\ddagger = \{1,3\}$ **then**
      $b_i(S) = \frac{c_i(S)}{c_i^T(S)c_i(S)}$;
      $\frac{\partial b_i^T(S)}{\partial s_k} = \frac{c_i^T(S)c_i(S)\frac{\partial c_i^T(S)}{\partial s_k} - \frac{\partial c_i^T(S)}{\partial s_k}c_i(S)c_i^T(S) - c_i^T(S)\frac{\partial c_i(S)}{\partial s_k}c_i^T(S)}{\left(c_i^T(S)c_i(S)\right)^2}$;
    end if
    if $\ddagger = \{1,4\}$ or $\ddagger = \{1\}$ **then**
      $b_i(S) = \frac{\left(I - A_{i-1}(S)A^{\ddagger}_{i-1}(S)\right)^T c_i}{c_i(S)^T c_i(S)}$
      $\frac{\partial b_i^T(S)}{\partial s_k} = \frac{\frac{\partial c_i(S)^T}{\partial s_k} - c_i(S)^T A_{i-1}(S)\frac{\partial (A^{\ddagger}_{i-1}(S))}{\partial s_k} - c_i(S)^T\frac{\partial A_{i-1}(S)}{\partial s_k}A^{\ddagger}_{i-1}(S) - \frac{\partial c_i(S)^T}{\partial s_k}A_{i-1}(S)A^{\ddagger}_{i-1}(S)}{c_i^T(S)c_i(S)}$
      $- \frac{\left(\frac{\partial c_i^T(S)}{\partial s_k}c_i(S) + c_i^T(S)\frac{\partial c_i(S)}{\partial s_k}\right)\left(c_i(S)^T - c_i(S)^T A_{i-1}(S)A^{\ddagger}_{i-1}(S)\right)}{\left(c_i^T(S)c_i(S)\right)^2}$.
    end if
  else
    if $\ddagger = \{1,2,3,4\}$ or $\ddagger = \{1,4\}$ **then**
      $b_i(S) = \frac{(A^{\ddagger}_{i-1}(S))^T d_i(S)}{1 + d_i(S)^T d_i(S)}$
      $\frac{\partial b_i^T(S)}{\partial s_k} = \frac{\left(1 + d_i^T(S)d_i(S)\right)\left(\frac{\partial d_i^T(S)}{\partial s_k}A^{\ddagger}_{i-1}(S) + d_i^T(S)\frac{\partial (A^{\ddagger}_{i-1}(S))}{\partial s_k}\right) - \left(d_i^T(S)A^{\ddagger}_{i-1}(S)\right)\left(\frac{\partial d_i^T(S)}{\partial s_k}d_i(S) + d_i^T(S)\frac{\partial d_i(S)}{\partial s_k}\right)}{\left(1 + d_i^T(S)d_i(S)\right)^2}$.
    end if
    if $\ddagger = \{1,3\}$ or $\ddagger = \{1\}$ **then**
      $b_i(S) = r_i(S)$; $\frac{\partial b_i^T(S)}{\partial s_k} = \frac{\partial r_i^T(S)}{\partial s_k}$
    end if
  end if
  $A^{\ddagger}_i(S) = \begin{bmatrix} A^{\ddagger}_{i-1}(S) - d_i(S)b_i^T(S) \\ b_i^T(S) \end{bmatrix}$
  $\frac{\partial A^{\ddagger}_i(S)}{\partial s_k} = \begin{bmatrix} \frac{\partial (A^{\ddagger}_{i-1}(S))}{\partial s_k} - \frac{\partial d_i(S)}{\partial s_k}b_i^T(S) - d_i(S)\frac{\partial b_i^T(S)}{\partial s_k} \\ \frac{\partial b_i^T(S)}{\partial s_k} \end{bmatrix}$.
end for
return The stopping criterion: $A(S)^{\ddagger} = A^{\ddagger}_n(S)$; $\frac{\partial A(S)^{\ddagger}}{\partial s_k} = \frac{\partial A^{\ddagger}_n(S)}{\partial s_k}$.

Now assume that $A(S) \in \mathbb{R}[S]^{m\times n}$ is a multi-variable polynomial matrix. We can represent the matrix A in the polynomial form

$$A(S) = \sum_{i_1=0}^{d_1} \cdots \sum_{i_p=0}^{d_p} A_{i_1,\ldots,i_p} s_1^{i_1} \cdots s_p^{i_p} = \sum_{I=0}^{Q} A_I S^I, \tag{2.1}$$

where $I = (i_1, \ldots, i_p)$, $A_I = A_{i_1,\ldots,i_p}$ are constant $m \times n$ matrices (called the coefficient matrices), $S^I = s_1^{i_1} s_2^{i_2} \cdots s_p^{i_p}$, $Q = (d_1, \ldots, d_p) = \deg A(S)$. Here $d_i = \deg(A(S), s_i)$ is the degree of $A(S)$ with respect to variable s_i in (2.1).

Theorem 2.2. *Consider the matrix $A(S) \in \mathbb{R}[S]^{m\times n}$ of the form* (2.1)*. The following statements are valid:*

(1) *Generalized inverse $A^{\ddagger}_i(S) \in \mathbb{R}[S]^{i\times m}$ and its partial derivative $\frac{\partial A^{\ddagger}_i(s)}{\partial s_k}$, corresponding to the first i*

columns in $A(S)$ have the general form

$$A_i^{\ddagger}(S) = \frac{X_i(S)}{Y_i(S)},\ i = 1, \ldots, n, \tag{2.2}$$

$$\frac{\partial A_i^{\ddagger}(S)}{\partial s_k} = \frac{Y_i(S)\frac{\partial X_i(S)}{\partial s_k} - \frac{\partial Y_i(S)}{\partial s_k} X_i(S)}{Y_i(S)^2},\ i = 1, \ldots, n. \tag{2.3}$$

where $X_i(S) \in \mathbb{R}[S]^{i\times m}$ and $Y_i(S) \in \mathbb{R}[S]$.

(2) *Matrix $X_i(S)$ and polynomial $Y_i(S)$ can be computed from $X_{i-1}(S)$, $Y_{i-1}(S)$, $A_{i-1}(S)$ and $a_i(S)$ (or $r_i(S)$), using exact recurrence relations, for each $i \geq 2$.*

(3) *Partial derivatives $\frac{\partial X_i(s)}{\partial s_k}$ and $\frac{\partial Y_i(s)}{\partial s_k}$ can be computed from $X_{i-1}(S)$, $Y_{i-1}(S)$, $A_{i-1}(S)$, $a_i(S)$ (or $r_i(S)$) and partial derivatives $\frac{\partial X_{i-1}(s)}{\partial s_k}$, $\frac{\partial Y_{i-1}(s)}{\partial s_k}$ and $\frac{\partial a_i(s)}{\partial s_k}$ (or $\frac{\partial r_i(s)}{\partial s_k}$), using exact recurrence relations and no derivative operation, for each $i \geq 2$.*

Proof. We prove theorem by the induction. Exact relations for $X_1(S)$ and $Y_1(S)$ can be derived after the investigations of two possible cases: $a_1(S) = \mathbf{0}$ and $a_1(S) \neq \mathbf{0}$.

In the case $a_1(S) = A_1(S) = \mathbf{0}$ it suffices to use $X_1(S) = \mathbf{0},\ Y_1(S) = 1$. Indeed, since

$$\frac{\partial X_1(S)}{\partial s_k} = \frac{\partial Y_1(S)}{\partial s_k} = 0,$$

the proof follows from Step 2 of Algorithm 2.1.

In the case $a_1(S) = A_1(S) \neq 0$ we observe two different cases, (a) and (b).

(a) Assume that $\ddagger = \{1, 3\}$ or $\ddagger = \{1, 2, 3, 4\}$. Algorithm 2.1 computes $a_1(S)$ in steps 5 and 6. Note that $a_1^T(S)$ is the numerator and $a_1^T(S)a_1(S)$ is denominator of $a_1^{\ddagger}(S)$, given by step 5. Since $X_1(S)$ and $Y_1(S)$ are polynomials, we conclude that $X_1(S) = a_1^T(S)$, $Y_1(S) = a_1^T(S)a_1(S)$ and

$$\frac{\partial X_1(S)}{\partial s_k} = \frac{\partial a_1^T(S)}{\partial s_k}, \quad \frac{\partial Y_1(S)}{\partial s_k} = \frac{\partial a_1^T(S)}{\partial s_k} a_1(S) + a_1^T(S)\frac{\partial a_1(S)}{\partial s_k}$$

(b) Otherwise, let $\ddagger = \{1, 4\}$ or $\ddagger = \{1\}$. Algorithm 2.1 then executes steps 9 and 10 of Algorithm 2.1. From the same reasons as in the previous case, we have

$$X_1(S) = r_1^T(S), \quad Y_1(S) = r_1^T(S)a_1(S)$$
$$\frac{\partial X_1(S)}{\partial s_k} = \frac{\partial r_1^T(S)}{\partial s_k}, \quad \frac{\partial Y_1(S)}{\partial s_k} = \frac{\partial r_1^T(S)}{\partial s_k} a_1(S) + r_1^T(S)\frac{\partial a_1(S)}{\partial s_k},$$

Consider now the inductive step. From the inductive hypothesis we can write

$$A_{i-1}^{\ddagger}(S) = \frac{X_{i-1}(S)}{Y_{i-1}(S)} \ \text{ and } \ \frac{\partial A_{i-1}^{\ddagger}(S)}{\partial s_k} = \frac{Y_{i-1}(S)\frac{\partial X_{i-1}(S)}{\partial s_k} - \frac{\partial Y_{i-1}(S)}{\partial s_k} X_{i-1}(S)}{Y_{i-1}(S)^2}.$$

Then $A_i^{\ddagger}(S)$ can be computed by using Step 2 of Algorithm 2.1. From Steps 14 and 16 we have:

$$d_i(S) = A_{i-1}^{\ddagger}(S)a_i(S) = \frac{X_{i-1}(S)a_i(S)}{Y_{i-1}(S)} = \frac{D_i(S)}{Y_{i-1}(S),}$$
$$c_i(S) = \left(I - A_{i-1}(S)\frac{X_{i-1}(S)}{Y_{i-1}(S)}\right) a_i(S)$$
$$= \frac{(Y_{i-1}(S)I - A_{i-1}(S)X_{i-1}(S))\, a_i(S)}{Y_{i-1}(S)} = \frac{M_{i-1}(S)a_i(S)}{Y_{i-1}(S)} = \frac{C_i(S)}{Y_{i-1}(S)},$$

where we denoted by $M_{i-1}(S) = Y_{i-1}(S)I - A_{i-1}(S)X_{i-1}(S)$. Note that

$$\frac{\partial D_i(S)}{\partial s_k} = \frac{\partial X_{i-1}(S)}{\partial s_k} a_i(S) + X_{i-1}(S)\frac{\partial a_i(S)}{\partial s_k},$$
$$\frac{\partial M_i(S)}{\partial s_k} = \frac{\partial Y_{i-1}(S)}{\partial s_k} I - \frac{\partial A_{i-1}(S)}{\partial s_k} X_{i-1}(S) - A_{i-1}(S)\frac{\partial X_{i-1}(S)}{\partial s_k},$$
$$\frac{\partial C_i(S)}{\partial s_k} = \frac{\partial M_{i-1}(S)}{\partial s_k} a_i(S) + M_{i-1}(S)\frac{\partial a_i(S)}{\partial s_k}.$$

Therefore,

$$\frac{\partial d_i(S)}{\partial s_k} = \frac{Y_{i-1}(S)\frac{\partial D_i(S)}{\partial s_k} - \frac{\partial Y_{i-1}(S)}{\partial s_k} D_i(S)}{Y_{i-1}(S)^2}, \quad \frac{\partial c_i(S)}{\partial s_k} = \frac{Y_{i-1}(S)\frac{\partial C_i(S)}{\partial s_k} - \frac{\partial Y_{i-1}(S)}{\partial s_k} C_i(S)}{Y_{i-1}(S)^2}.$$

In the case $C_i(S) \neq \mathbf{0}$ we have two possibilities, corresponding to the class of pseudoinverses we compute (holds directly from steps 20, 21, 24 and 25 of Algorithm 2.1):

(a) If $\ddagger = \{1,4\}$ or $\ddagger = \{1\}$ then

$$b_i(S) = \frac{\left(I - A_{i-1}(S)\frac{X_{i-1}(S)}{Y_{i-1}(S)}\right)^T \frac{C_i(S)}{Y_{i-1}(S)}}{\frac{C_i^T(S)C_i(S)}{Y_{i-1}^T(S)Y_{i-1}(S)}} = \frac{M_{i-1}^T(S)C_i(S)}{C_i^T(S)C_i(S)} = \frac{V_i(S)}{W_i(S)},$$
$$\frac{\partial V_i(S)}{\partial s_k} = \frac{\partial M_{i-1}^T(S)}{\partial s_k} C_i(S) + M_{i-1}^T(S)\frac{\partial C_i(S)}{\partial s_k}, \quad \frac{\partial W_i(S)}{\partial s_k} = \frac{\partial C_i^T(S)}{\partial s_k} C_i(S) + C_i^T(S)\frac{\partial C_i(S)}{\partial s_k}.$$

(b) If $\ddagger = \{1,3\}$ or $\ddagger = \{1,2,3,4\}$ then

$$b_i(S) = \frac{\frac{C_i(S)}{Y_{i-1}(S)}}{\frac{C_i^T(S)C_i(S)}{Y_{i-1}^T(S)Y_{i-1}(S)}} = \frac{Y_{i-1}^T(S)C_i(S)}{C_i^T(S)C_i(S)} = \frac{V_i(S)}{W_i(S)},$$
$$\frac{\partial V_i(S)}{\partial s_k} = \frac{\partial Y_{i-1}^T(S)}{\partial s_k} C_i(S) + Y_{i-1}^T(S)\frac{\partial C_i(S)}{\partial s_k}, \quad \frac{\partial W_i(S)}{\partial s_k} = \frac{\partial C_i^T(S)}{\partial s_k} C_i(S) + C_i^T(S)\frac{\partial C_i(S)}{\partial s_k}.$$

Otherwise if $C_i(S) = \mathbf{0}$ we have (holds directly from steps 29, 30 and 33 of Algorithm 2.1):

(a) If $\ddagger = \{1,3\}$ or $\ddagger = \{1\}$ then $V_i(S)$ can be chosen arbitrarily and $W_i(S) = 1$.

(b) If $\ddagger = \{1,4\}$ or $\ddagger = \{1,2,3,4\}$ then

$$b_i(S) = \frac{\frac{X_{i-1}^T(S)}{Y_{i-1}^T(S)}\frac{D_i(S)}{Y_{i-1}(S)}}{1 + \frac{D_i^T(S)D_i(S)}{Y_{i-1}^T(S)Y_{i-1}(S)}} = \frac{X_{i-1}^T(S)D_i(S)}{Y_{i-1}^T(S)Y_{i-1}(S) + D_i^T(S)D_i(S)} = \frac{V_i(S)}{W_i(S)},$$
$$\frac{\partial V_i(S)}{\partial s_k} = \frac{\partial X_{i-1}^T(S)}{\partial s_k} D_i(S) + X_{i-1}^T(S)\frac{\partial D_i(S)}{\partial s_k},$$
$$\frac{\partial W_i(S)}{\partial s_k} = \frac{\partial Y_{i-1}^T(S)}{\partial s_k} Y_{i-1}(S) + Y_{i-1}^T(S)\frac{\partial Y_{i-1}(S)}{\partial s_k} + \frac{\partial D_i^T(S)}{\partial s_k} D_i(S) + D_i^T(S)\frac{\partial D_i(S)}{\partial s_k}.$$

The gradient $\frac{\partial b_i(S)}{\partial s_k}$ is in all cases equal to

$$\frac{\partial b_i(S)}{\partial s_k} = \frac{W_i(S)\frac{\partial V_i(S)}{\partial s_k} - \frac{\partial W_{i-1}(S)}{\partial s_k} V_i(S)}{W_i(S)^2}.$$

Now from Step 36 we obtain

$$A_i^{\ddagger}(S) = \begin{bmatrix} \frac{X_{i-1}(S)}{Y_{i-1}(S)} - \frac{D_i(S)}{Y_{i-1}(S)}\frac{V_i^T(S)}{W_i^T(S)} \\ \frac{V_i^T(S)}{W_i^T(S)} \end{bmatrix} = \frac{1}{Y_{i-1}(S)W_i^T(S)} \begin{bmatrix} X_{i-1}(S)W_i^T(S) - D_i(S)V_i^T(S) \\ Y_{i-1}(S)V_i^T(S) \end{bmatrix}.$$

From the last expression we can conclude that

$$X_i(S) = \begin{bmatrix} X_{i-1}(S)W_i^T(S) - D_i(S)V_i^T(S) \\ Y_{i-1}(S)V_i^T(S) \end{bmatrix}, \quad Y_i(S) = Y_{i-1}(S)W_i^T(S)$$

and therefore

$$\frac{\partial X_i(S)}{\partial s_k} = \begin{bmatrix} \frac{\partial X_{i-1}(S)}{\partial s_k}W_i^T(S) + X_{i-1}(S)\frac{\partial W_i^T(S)}{\partial s_k} - \frac{\partial D_i(S)}{\partial s_k}V_i^T(S) - D_i(S)\frac{\partial V_i^T(S)}{\partial s_k} \\ \frac{\partial Y_{i-1}(S)}{\partial s_k}V_i^T(S) + Y_{i-1}(S)\frac{\partial V_i^T(S)}{\partial s_k} \end{bmatrix},$$

$$\frac{\partial Y_i(S)}{\partial s_k} = \frac{\partial Y_{i-1}(S)}{\partial s_k}W_i^T(S) + Y_{i-1}(S)\frac{\partial W_i^T(S)}{\partial s_k}.$$

This completes the proof by mathematical induction of the part **(1)** of the theorem. Parts **(2)** and **(3)** directly hold according to the above expressions. □

From Theorem 2.2 we can conclude that $\frac{\partial A^{\ddagger}(S)}{\partial s_k}$ depends only on $\frac{\partial A(S)}{\partial s_k}$, $A(S)$ and $\frac{\partial R(S)}{\partial s_k}$ where $R(s) = [r_1(S)\ r_2(S)\ \cdots\ r_n(S)]$ is the matrix whose columns are arbitrary vectors used in steps 9 and 10 of Algorithm 2.1.

Based on Theorem 2.2, it is not hard to construct the algorithm for computing $A^{\ddagger}(S)$ and its partial derivatives.

Remark 2.1. At the end of this section, note that parts **(1)** and **(2)** of Theorem 2.2 ares valid also in the case of multi-variable rational and polynomial complex matrix, i.e. for $A \in \mathbb{C}[S]^{m\times n}$. Same holds for Algorithm 2.1 when $A \in \mathbb{C}(S)^{m\times n}$ is multi-variable rational complex matrix. In such case, the matrix A^* is rational or polynomial matrix with respect to the conjugated variables $\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_p$, i.e. $A^* \in \mathbb{R}(\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_p)^{m\times n}$. Hence by using conjugate-transpose operation on rational matrices, together with matrix addition and multiplication, results will be rational or polynomial matrices with respect to the variables $s_1, s_2, \ldots, s_p, \overline{s}_1, \overline{s}_2, \ldots, \overline{s}_p$. Therefore, in the case of complex matrices, we have to assume that $S = (s_1, s_2, \ldots, s_p, \overline{s}_p, \overline{s}_{p-1}, \ldots, \overline{s}_1)$, i.e. the number of variables is $2p$.

3 Effective method

Polynomial matrices which occur in many mathematical and scientific applications are sparse, in the sense that they have a comparatively small number of nonzero coefficients. Also, it is possible that polynomial entries of these matrices are sparse. To avoid executing redundant operations on zero matrices in the case when a substantial minority of nonzero elements (coefficients) is detected, appropriate sparse structures should be used in such things. We assume that algorithms on polynomial matrices are effective if the sparse data structure represents a matrix in a space proportional to the number of nonzero coefficients and matrix operations are performed in a time adequate to measured sparsity.

We restate the next two definitions which provide the quantitative measure of sparsity for multi-variable polynomial matrices.

Definition 3.1. [6] *For a given matrix $A(S) = [a_{ij}(S)] \in \mathbb{R}[S]^{m\times n}$ (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 $A(S)$:*

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

Definition 3.2. [6] *For a given polynomial matrix $A(S) \in R[S]^{m\times n}$ and $S = (s_1, \ldots, s_p)$,* **the second sparse number** *$sp_2(A(S))$ is the ratio*

$$sp_2(A(S)) = \frac{|\ \{(i,j,k_1,\ldots,k_p)\ |\ 0\leq k_j \leq \deg(A(S),s_j),\ \mathrm{Coef}(a_{ij}(S), s_1^{k_1}\cdots s_p^{k_p}) \neq 0\}\ |}{\deg(A(S),s_1)\cdots\deg(A(S),s_p)\cdot m\cdot n},$$

where $\mathrm{Coef}(P(S), s_1^{k_1}\cdots s_p^{k_p})$ *denotes the coefficient corresponding to $s_1^{k_1}\cdots s_p^{k_p}$ in polynomial $P(S)$.*

The first sparse number represents the density of nonzero elements. The second sparse number represents density of nonzero coefficients contained in elements $a_{ij}(S)$. Both sparse numbers are between 0 and 1.

In this section we briefly recall sparse storage formats **Eff** and **Ef** from [6] and consider the partial derivative computation on both structures. The result is the effective algorithm for computing $A^{\ddagger}(S)$ and corresponding partial derivatives.

The main idea in the sparse structure **Eff** is to exploit only nonzero coefficient matrices $A_I = A_{i_1,\dots,i_p} \neq \mathbf{O}$ of the polynomial matrix $A(S)$ given in the form (2.1).

Definition 3.3. [6] *The effective sparse structure of the polynomial matrix $A(S)$, defined in* (2.1)*, is equal to*

$$\mathbf{Eff}_A = \{(J, A_J) \mid A_J \neq \mathbf{O},\ \mathbf{0} \leq J \leq \deg A(S)\}\,. \tag{3.1}$$

Recall that by $\mathbf{0}$ we denote appropriate zero vector and by $\mathbf{O}$ we denote appropriate zero matrix. Also, the index set of this effective structure is defined by

$$\mathrm{Ind}_A = \{J \mid A_J \neq \mathbf{O},\ \mathbf{0} \leq J \leq \deg A(S)\}\,. \tag{3.2}$$

Define operations $+$, $-$, $\cdot$ and ^{T} on sparse structures by

$$\begin{aligned} \mathbf{Eff}_A + \mathbf{Eff}_B &= \mathbf{Eff}_{A+B}, \quad & \mathbf{Eff}_A - \mathbf{Eff}_B &= \mathbf{Eff}_{A-B}, \\ \mathbf{Eff}_A \cdot \mathbf{Eff}_B &= \mathbf{Eff}_{A\cdot B}, \quad & \mathbf{Eff}_A^T &= \mathbf{Eff}_{A^T}. \end{aligned} \tag{3.3}$$

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

It is not hard to prove that if $C(S) = A(S) \cdot B(S)$ then $e_C \leq e_A + e_B$ and hence the effective structure $\mathbf{Eff}_C$ can be computed in the time $O(e_A + e_B)$. Similarly, if $D(S) = A(S) + B(S)$ then $e_D \leq \max\{e_A, e_B\}$ which implies that $\mathbf{Eff}_D$ can be computed in time $O(\max\{e_A, e_B\})$. In view of (3.3), we compute $\mathbf{Eff}_A^T = \{(I, A_I^T) \mid (I, A_I) \in \mathbf{Eff}_A\}$ with complexity $O(e_A)$.

Definition 3.4. *The partial derivative of the sparse structure* **Eff** *of the polynomial matrix $A(S)$, defined in* (2.1)*, is defined by $\partial_k \mathbf{Eff}_{A(S)} = \partial_{s_k} \mathbf{Eff}_{A(S)} = \mathbf{Eff}_{\frac{\partial A(S)}{\partial s_k}}$.*

Since

$$\frac{\partial A(S)}{\partial s_k} = \frac{\partial}{\partial s_k} \sum_{J \in \mathrm{Ind}_A} A_J s_1^{j_1} s_2^{j_2} \cdots s_p^{j_p} = \sum_{J \in \mathrm{Ind}_A, j_k \neq 0} j_k A_J s_1^{j_1} \cdots s_{k-1}^{j_{k-1}} s_k^{j_k - 1} s_{k+1}^{j_{k+1}} \cdots s_p^{j_p},$$

we conclude that $\partial_k \mathbf{Eff}_{A(S)}$ is equal to

$$\begin{aligned} \partial_k \mathbf{Eff}_{A(S)} &= \partial_{s_k} \mathbf{Eff}_{A(S)} = \mathbf{Eff}_{\frac{\partial A(S)}{\partial s_k}} \\ &= \{(J - e_k, j_k A_J) \mid J = (j_1, j_2, \dots, j_p) \in \mathrm{Ind}_A,\ j_k \neq 0,\ e_k = (\underbrace{0, \dots, 0}_{k-1}, 1, 0, \dots, 0)\}. \end{aligned} \tag{3.4}$$

It is clear that the derivative computation can be performed in the time complexity equal to $O(e_A)$.

Usually, coefficient matrices A_I in the polynomial representation (2.1), i.e. in the sparse representation (3.1) are sparse. Using this fact we can significantly improve our sparse structure **Eff** by using an appropriate structure for these constant coefficient matrices.

Definition 3.5. [6] *For the constant matrix $A = [a_{ij}] \in \mathbb{R}^{m \times n}$, define the following sparse structure:*

$$\mathbf{Sp}_A = \{(i, j, a_{ij}) \mid a_{ij} \neq 0\}\,. \tag{3.5}$$

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 matrix operations on $\mathbf{Sp}_A$ (see [6]). In this way, we have the following improvement of the structure **Eff**:

$$\begin{aligned} \mathbf{Eff}'_A &= \left\{(J, \mathbf{Sp}_{A_J}) \mid A_J \neq \mathbf{O},\ \mathbf{0} \leq J \leq \deg A(S)\right\} \\ &= \{(J, \{i, j, (A_J)_{ij} | (A_J)_{ij} \neq 0\}) \mid A_J \neq \mathbf{O},\ \mathbf{0} \leq J \leq \deg A(S)\}\,. \end{aligned} \tag{3.6}$$

It can be seen that the complexity of computing $\mathbf{Sp}_A+\mathbf{Sp}_B$ is $O(s_A+s_B)$ and for $\mathbf{Sp}_A^T$ is $O(s_A)$. In the case of multiplication the complexity depends on concrete implementation. Sparse structure **Sp** is implemented in MATHEMATICA as the structure `SparseArray`. Note that in MATHEMATICA, the basic matrix operations are performed in the same way on `SparseArray` structure and usual matrices [16]. Hence, the same code can be used in both cases. It is also possible in object-oriented programming languages, using the inheritance concept and virtual classes.

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 $a_{ij}(S)$ [6].

Definition 3.6. [6] *Effective structure for the scalar polynomial* $a(S)=\sum_{I=1}^{\deg a(S)} a_I S^I$ *is defined similarly as in the matrix case* (3.1)*:*

$$\mathbf{Ef}_a=\{(J,a_J)\mid a_J\neq 0,\ \mathbf{0}\leq J\leq \deg a(S)\}. \tag{3.7}$$

Sparse representation of the matrix $A(S)$ *is denoted by* $\mathbf{Ef}_{A(S)}=[\mathbf{Ef}_{a_{ij}(S)}]$.

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 $O(\mathbf{ef}_A+\mathbf{ef}_B)$ while the complexity of matrix multiplication is $O\left(\sum_{k=1}^n \mathbf{col}(A,k)\mathbf{row}(B,k)\right)$. Here we denoted by $\mathbf{row}(B,k)=\sum_{j=1}^p e_{b_{kj}}$ and $\mathbf{col}(A,k)=\sum_{i=1}^m e_{a_{ik}}$.

Partial derivative of the sparse structure **Ef** is introduced in the next definition.

Definition 3.7. *The partial derivative of the sparse structure* **Ef** *of the polynomial matrix* $A(S)$*, defined in* (2.1)*, is equal to* $\partial_k\mathbf{Ef}_{A(S)}=\partial_{s_k}\mathbf{Ef}_{A(S)}=\mathbf{Ef}_{\frac{\partial A(S)}{\partial s_k}}$.

As in the previous case, $\partial_k\mathbf{Ef}_{A(S)}$ can be computed using the following relation

$$\partial_k\mathbf{Ef}_{A(S)}=\{(J-1,j_ka_J)\mid a_J\neq 0,\ \mathbf{0}\leq J\leq \deg a(S)\}. \tag{3.8}$$

The following algorithm is the effective partitioning method for computing the generalized inverses of polynomial matrices and its partial derivatives. It is 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 also use the same symbol for the effective structure of the number 0.

Algorithm 3.2 Computing the $\{1\},\{1,3\},\{1,4\}$ or $\{1,2,3,4\}$ inverse $A^{\ddagger}(S)$ of sparse matrix $A(S)$ and corresponding partial derivative $\frac{\partial A^{\ddagger}(S)}{\partial s_k}$.

Require: Effective structure of matrix $A(S)$
and of the matrix $R(S)=\begin{bmatrix}r_1(S) & r_2(S) & \cdots & r_n(S)\end{bmatrix}$ in the case $\ddagger=\{1,3\}$ or $\ddagger=\{1\}$.

$\mathcal{E}_{dA}=\partial_k\mathcal{E}_A$, also compute $\mathcal{E}_{da_i}=\partial_k\mathcal{E}_{a_i}$ and $\mathcal{E}_{dA_i}=\partial_k\mathcal{E}_{A_i}$.
if $\ddagger=\{1,3\}$ or $\ddagger=\{1\}$ **then**
  $\mathcal{E}_{dR}=\partial_k\mathcal{E}_R$ and $\mathcal{E}_{dr_i}=\partial_k\mathcal{E}_{r_i}$.
end if
if $\mathcal{E}_{a_1}=\mathcal{O}$ **then**
  $\mathcal{E}_{X_1}=\mathcal{O},\quad \mathcal{E}_{Y_1}=\mathcal{E}_1,\quad \mathcal{E}_{dX_1}=\mathcal{O},\quad \mathcal{E}_{dY_1}=\mathcal{O}.$
else
  if $\ddagger=\{1,2,3,4\}$ or $\ddagger=\{1,3\}$ **then**
    $\mathcal{E}_{X_1}=\mathcal{E}_{a_1}^T,\quad \mathcal{E}_{Y_1}=\mathcal{E}_{a_1}^T\cdot\mathcal{E}_{a_1},\quad \mathcal{E}_{dX_1}=\mathcal{E}_{da_1}^T,\quad \mathcal{E}_{dY_1}=\mathcal{E}_{da_1}^T\cdot\mathcal{E}_{a_1}+\mathcal{E}_{a_1}^T\cdot\mathcal{E}_{da_1}.$
  end if
  if $\ddagger=\{1,4\}$ or $\ddagger=\{1\}$ **then**
    $\mathcal{E}_{X_1}=\mathcal{E}_{r_1}^T,\quad \mathcal{E}_{Y_1}=\mathcal{E}_{r_1}^T\cdot\mathcal{E}_{a_1},\quad \mathcal{E}_{dX_1}=\partial_k\mathcal{E}_{r_1}^T,\quad \mathcal{E}_{dY_1}=\partial_k\mathcal{E}_{r_1}^T\cdot\mathcal{E}_{a_1}+\mathcal{E}_{r_1}^T\cdot\mathcal{E}_{da_1}$, where $r_1(S)$ is an arbitrary vector.
  end if
end if

15: **for** $i = 2, \dots, n$ **do**

16: $\mathcal{E}_{d_i} = \mathcal{E}_{X_{i-1}} \cdot \mathcal{E}_{a_i}, \quad \mathcal{E}_{dd_i} = \mathcal{E}_{dX_{i-1}} \cdot \mathcal{E}_{a_i} + \mathcal{E}_{X_{i-1}} \cdot \mathcal{E}_{da_i}.$

17: $\mathcal{E}_{M_{i-1}} = \mathcal{E}_{Y_{i-1}} \cdot \mathcal{E}_I - \mathcal{E}_{A_{i-1}} \cdot \mathcal{E}_{X_{i-1}}, \quad \mathcal{E}_{dM_{i-1}} = \mathcal{E}_{dY_{i-1}} \cdot \mathcal{E}_I - \mathcal{E}_{dA_{i-1}} \cdot \mathcal{E}_{X_{i-1}} - \mathcal{E}_{A_{i-1}} \cdot \mathcal{E}_{dX_{i-1}},$

18: $\mathcal{E}_{c_i} = \mathcal{E}_{M_{i-1}} \cdot \mathcal{E}_{a_i}, \quad \mathcal{E}_{dc_i} = \mathcal{E}_{dM_{i-1}} \cdot \mathcal{E}_{a_i} + \mathcal{E}_{M_{i-1}} \cdot \mathcal{E}_{da_i}.$

19: **if** $\mathcal{E}_{c_i} \neq \mathcal{O}$ **then**

20: **if** $\ddagger = \{1, 4\}$ or $\ddagger = \{1\}$ **then**

21: $\mathcal{E}_{V_i} = \mathcal{E}_{M_{i-1}}^T \cdot \mathcal{E}_{c_i}, \quad \mathcal{E}_{W_i} = \mathcal{E}_{c_i}^T \cdot \mathcal{E}_{c_i},$

$\mathcal{E}_{dV_i} = \mathcal{E}_{dM_{i-1}}^T \cdot \mathcal{E}_{c_i} + \mathcal{E}_{M_{i-1}}^T \cdot \mathcal{E}_{dc_i}, \quad \mathcal{E}_{dW_i} = \mathcal{E}_{dc_i}^T \cdot \mathcal{E}_{c_i} + \mathcal{E}_{c_i}^T \cdot \mathcal{E}_{dc_i}.$

22: **end if**

23: **if** $\ddagger = \{1, 3\}$ or $\ddagger = \{1, 2, 3, 4\}$ **then**

24: $\mathcal{E}_{V_i} = \mathcal{E}_{Y_{i-1}}^T \cdot \mathcal{E}_{c_i}, \quad \mathcal{E}_{W_i} = \mathcal{E}_{c_i}^T \cdot \mathcal{E}_{c_i},$

$\mathcal{E}_{dV_i} = \mathcal{E}_{dY_{i-1}}^T \cdot \mathcal{E}_{c_i} + \mathcal{E}_{Y_{i-1}}^T \cdot \mathcal{E}_{dc_i}, \quad \mathcal{E}_{dW_i} = \mathcal{E}_{dc_i}^T \cdot \mathcal{E}_{c_i} + \mathcal{E}_{c_i}^T \cdot \mathcal{E}_{dc_i}.$

25: **end if**

26: **else**

27: **if** $\ddagger = \{1, 3\}$ or $\ddagger = \{1\}$ **then**

28: $\mathcal{E}_{V_i} = \mathcal{E}_{r_i}, \quad \mathcal{E}_{W_i} = \mathcal{E}_1, \quad \mathcal{E}_{dV_i} = \partial_k \mathcal{E}_{r_i}, \quad \mathcal{E}_{dW_i} = \mathcal{O}.$
($\mathcal{E}_{r_i}$ is an effective structure of an arbitrary vector $r_i(S)$).

29: **end if**

30: **if** $\ddagger = \{1, 4\}$ or $\ddagger = \{1, 2, 3, 4\}$ **then**

31: $\mathcal{E}_{V_i} = \mathcal{E}_{X_{i-1}}^T \cdot \mathcal{E}_{d_i}, \quad \mathcal{E}_{W_i} = \mathcal{E}_{Y_{i-1}}^T \cdot \mathcal{E}_{Y_{i-1}} + \mathcal{E}_{d_i}^T \cdot \mathcal{E}_{d_i},$

$\mathcal{E}_{dV_i} = \mathcal{E}_{dX_{i-1}}^T \cdot \mathcal{E}_{d_i} + \mathcal{E}_{X_{i-1}}^T \cdot \mathcal{E}_{dd_i}, \quad \mathcal{E}_{dW_i} = \mathcal{E}_{dY_{i-1}}^T \cdot \mathcal{E}_{Y_{i-1}} + \mathcal{E}_{Y_{i-1}}^T \cdot \mathcal{E}_{dY_{i-1}} + \mathcal{E}_{dd_i}^T \cdot \mathcal{E}_{d_i} + \mathcal{E}_{d_i}^T \cdot \mathcal{E}_{dd_i}.$

32: **end if**

33: **end if**

34: $\mathcal{E}_{\Theta_i} = \mathcal{E}_{X_{i-1}} \cdot \mathcal{E}_{W_i}^T - \mathcal{E}_{d_i} \cdot \mathcal{E}_{V_i}^T, \quad \mathcal{E}_{\Psi_i} = \mathcal{E}_{Y_{i-1}} \cdot \mathcal{E}_{V_i}^T$

$\mathcal{E}_{d\Theta_i} = \mathcal{E}_{dX_{i-1}} \cdot \mathcal{E}_{W_i}^T + \mathcal{E}_{X_{i-1}} \cdot \mathcal{E}_{dW_i}^T - \mathcal{E}_{dd_i} \cdot \mathcal{E}_{V_i}^T - \mathcal{E}_{d_i} \cdot \mathcal{E}_{dV_i}^T, \quad \mathcal{E}_{d\Psi_i} = \mathcal{E}_{dY_{i-1}} \cdot \mathcal{E}_{V_i}^T + \mathcal{E}_{Y_{i-1}} \cdot \mathcal{E}_{dV_i}^T.$

35: $X_i = \begin{bmatrix} \Theta_i \\ \Psi_i \end{bmatrix}, \quad dX_i = \begin{bmatrix} d\Theta_i \\ d\Psi_i \end{bmatrix}, \quad \mathcal{E}_{Y_i} = \mathcal{E}_{Y_{i-1}} \cdot \mathcal{E}_{W_i}^T, \quad \mathcal{E}_{dY_i} = \mathcal{E}_{dY_{i-1}} \cdot \mathcal{E}_{W_i}^T + \mathcal{E}_{Y_{i-1}} \cdot \mathcal{E}_{dW_i}^T.$

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

$$\begin{aligned} \mathbf{Ef}_{X_i} = \begin{bmatrix} \mathbf{Ef}_{\Theta_i} \\ \mathbf{Ef}_{\Psi_i} \end{bmatrix}, \quad \mathbf{Eff}_{X_i} = & \left\{ \left(j, \begin{bmatrix} (\Theta_i)_j \\ (\Psi_i)_j \end{bmatrix} \right) \mid (j, (\Theta_i)_j) \in \mathbf{Eff}_{\Theta_i}, (j, (\Psi_i)_j) \in \mathbf{Eff}_{\Psi_i} \right\} \\ & \cup \left\{ \left(j, \begin{bmatrix} (\Theta_i)_j \\ \mathbf{0} \end{bmatrix} \right) \mid (j, (\Theta_i)_j) \in \mathbf{Eff}_{\Theta_i}, (\Psi_i)_j = 0 \right\} \\ & \cup \left\{ \left(j, \begin{bmatrix} \mathbf{O} \\ (\Psi_i)_j \end{bmatrix} \right) \mid (\Theta_i)_j = \mathbf{O}, (j, (\Psi_i)_j) \in \mathbf{Eff}_{\Psi_i} \right\} \end{aligned} \tag{3.9}$$

Analogously we compute $\mathcal{E}_{dX_i}$ from $\mathcal{E}_{d\Theta_i}$ and $\mathcal{E}_{d\Psi_i}$.

37: Find the polynomials $X_i(S)$, $Y_i(S)$, $dX_i(S)$ and $dY_i(S)$ from its effective structures and compute:

$$\begin{aligned} A_i^{\ddagger}(S) &= \frac{X_i(S)}{Y_i(S)}, \\ \frac{\partial A^{\ddagger}(S)_i}{\partial s_k} &= \frac{dX_i(S)Y_i(S) - dY_i(S)X_i(S)}{Y_i(S)^2} \end{aligned} \tag{3.10}$$

38: **end for**

39: The stopping criterion is $i = n$. In this case is $A^{\ddagger}(S) = A_n^{\ddagger}(S)$ and $\frac{\partial A^{\ddagger}(S)}{\partial s_k} = \frac{\partial A_n^{\ddagger}(S)}{\partial s_k}$.

4 Examples

Algorithm 3.2 is implemented in the programming package MATHEMATICA (version 7.0) and tested on several test matrices. Functions `KalabaEf` and `KalabaEff` provide the implementation of Algorithm 3.2 for **Ef** and **Eff** sparse structures effectively. Tests are run on Intel Core i5 CPU at 2.6 GHz, without multicore optimization.

Example 4.1. The following table contains the CPU times of functions `KalabaEf` and `KalabaEff`, while computing $\{1,3\}$ inverses, on test matrices from [17]. All presented times are in seconds.

Table 1. CPU times corresponding to test matrices from Zielke [17]

Test matrix	Alg 3.2 with **Ef**	Alg. 3.2 with **Eff**
$V_2(s,t)$	0.094	7.363
$V_3(s,t)$	20.131	78.329
$V_3(s,s)$	0.031	3.127
$S_2(s)$	0.234	0.843
$S_3(s)$	17.847	61.312
$H_3(s)$	0.016	0.046
$H_4(s)$	0.032	0.218
$H_5(s)$	0.203	3.073
$H_6(s)$	2.668	50.217

Comparing the numerical data arranged in Table 1, it is clear that the sparse structure **Ef** requires significantly smaller computational time with respect to **Eff** structure.

We also tested our implementations on randomly generated test matrices. Results are provided for different levels of sparsity of the input matrices.

Example 4.2. In the following tables, we provide running times of the functions `KalabaEf` and `KalabaEff` for randomly generated test matrices of various dimensions and degrees. We have chosen six groups of tests, as the combination of three different levels of sparsity of an input matrix (0.4, 0.6 and 0.9) and two levels of its rank (2 and 4). That choose is made in order to demonstrate the dependence of running time of Algorithm 3.2 of both sparsity and rank.

Table 2. Numerical experiments for various dimensions m, n and degree d

m	n	d	Alg 3.2 with **Ef**	Alg. 3.2 with **Eff**
4	4	2	0.002	0.054
4	4	4	0.013	0.059
5	4	2	0.005	0.058
5	4	4	0.012	0.103
5	5	2	0.013	0.081
5	5	4	0.017	0.103
6	4	2	0.011	0.061
6	4	4	0.025	0.136
6	5	2	0.013	0.088
6	5	4	0.033	0.201
6	6	2	0.016	0.107
6	6	4	0.058	0.359

$sp_1(A(S)) = 0.4,\ sp_2(A(S)) = 0.4,\ rank(A(S)) = 2$

m	n	d	Alg 3.2 with **Ef**	Alg. 3.2 with **Eff**
4	4	2	0.009	0.087
4	4	4	0.061	0.433
5	4	2	0.025	0.130
5	4	4	0.047	0.302
5	5	2	0.035	0.230
5	5	4	0.271	1.305
6	4	2	0.025	0.125
6	4	4	0.458	1.628
6	5	2	0.096	0.402
6	5	4	0.946	3.750
6	6	2	0.118	0.530
6	6	4	1.051	4.524

$sp_1(A(S)) = 0.4,\ sp_2(A(S)) = 0.4,\ rank(A(S)) = 4$

m	n	d	Alg 3.2 with **Ef**	Alg. 3.2 with **Eff**
4	4	2	0.027	0.122
4	4	4	0.072	0.302
5	4	2	0.029	0.138
5	4	4	0.074	0.333
5	5	2	0.057	0.243
5	5	4	0.136	0.586
6	4	2	0.055	0.202
6	4	4	0.076	0.327
6	5	2	0.052	0.222
6	5	4	0.208	0.688
6	6	2	0.142	0.507
6	6	4	0.382	1.181

$sp_1(A(S)) = 0.6,\ sp_2(A(S)) = 0.6,\ rank(A(S)) = 2$

m	n	d	Alg 3.2 with **Ef**	Alg. 3.2 with **Eff**
4	4	2	0.393	1.548
4	4	4	2.621	9.479
5	4	2	0.590	1.981
5	4	4	4.028	13.367
5	5	2	1.271	4.223
5	5	4	4.914	16.146
6	4	2	0.811	2.496
6	4	4	5.005	15.282
6	5	2	1.532	4.481
6	5	4	7.117	20.068
6	6	2	2.453	7.471
6	6	4	13.191	38.625

$sp_1(A(S)) = 0.6,\ sp_2(A(S)) = 0.6,\ rank(A(S)) = 4$

m	n	d	Alg 3.2 with **Ef**	Alg. 3.2 with **Eff**
4	4	2	0.079	0.248
4	4	4	0.179	0.509
5	4	2	0.117	0.296
5	4	4	0.288	0.634
5	5	2	0.177	0.473
5	5	4	0.542	1.117
6	4	2	0.132	0.318
6	4	4	0.384	0.757
6	5	2	0.247	0.538
6	5	4	0.733	1.330
6	6	2	0.347	0.731
6	6	4	1.072	1.895

$sp_1(A(S)) = 0.9,\ sp_2(A(S)) = 0.9,\ rank(A(S)) = 2$

m	n	d	Alg 3.2 with **Ef**	Alg. 3.2 with **Eff**
4	4	2	0.998	3.145
4	4	4	4.859	15.853
5	4	2	1.763	5.006
5	4	4	6.382	18.622
5	5	2	3.058	7.839
5	5	4	11.191	30.075
6	4	2	1.886	5.019
6	4	4	8.208	21.998
6	5	2	3.482	8.562
6	5	4	15.175	36.473
6	6	2	5.569	12.648
6	6	4	21.530	50.610

$sp_1(A(S)) = 0.9,\ sp_2(A(S)) = 0.9,\ rank(A(S)) = 4$

It can be noticed, from Table 2, that the running time of Algorithm 3.2 increases when the sparsity of matrix $A(S)$ increases. The fact that this dependence is significant, shows that sparse structures considerably improve the performances of Algorithm 3.2. Moreover, it shows that intermediate matrices in Algorithm 3.2 are also sparse.

By comparing fourth and fifth column from each table in Table 2, we observe that the effective structure **Ef** posses considerably better performances than the effective structure **Eff**, for all test matrices.

Finally, by comparing values in tables from left and right side of Table 2, we can see that running time of Algorithm 3.2 depends on the rank of an input matrix $A(s)$. That dependence is notable both for sparse and dense matrices.

Example 4.3. We also compared Algorithm 3.2 (with both effective structures) without derivative computation with Algorithm 3.1 from [10]. Results are shown in the following tables.

Table 3. Comparison of both effective structures and usual representation

m	n	d	Alg 3.2 with **Ef**	Alg. 3.2 with **Eff**	Alg. 3.1 from [10]
4	4	2	0.012	0.044	0.621
4	4	4	0.025	0.105	1.217
5	4	2	0.025	0.049	0.643
5	4	4	0.027	0.118	1.617
5	5	2	0.026	0.104	0.899
5	5	4	0.052	0.208	3.931
6	5	2	0.027	0.098	1.111
6	5	4	0.062	0.213	4.150
6	6	2	0.047	0.166	2.532
6	6	4	0.135	0.348	5.528

$sp_1(A(S)) = 0.7,\ sp_2(A(S)) = 0.5,\ rank(A(S)) = 2$

m	n	d	Alg 3.2 with **Ef**	Alg. 3.2 with **Eff**	Alg. 3.1 from [10]
4	4	2	0.088	0.312	5.538
4	4	4	0.575	1.603	18.606
5	4	2	0.193	0.614	12.199
5	4	4	0.655	1.880	42.549
5	5	2	0.212	0.684	32.841
5	5	4	1.205	3.851	111.954
6	5	2	0.343	0.986	79.631
6	5	4	1.329	4.088	331.623
6	6	2	0.427	1.383	129.777
6	6	4	2.027	6.664	487.912

$sp_1(A(S)) = 0.7,\ sp_2(A(S)) = 0.5,\ rank(A(S)) = 4$

Besides the before observed advantages of the sparse representation **Ef** with respect to the representation **Eff**, from Table 3 we also observe significant acceleration of Algorithm 3.1 after the usage of both sparse representations. We also see that the difference is larger for right table (higher value of matrix rank).

5 Conclusion

We extended the algorithm for computing $\{1\}, \{1,3\}, \{1,4\}$, the Moore-Penrose inverse and the weighted Moore-Penrose inverse from [10] to the set of multiple-variable rational matrices with complex coefficients. An adaptation of the algorithm from [11], applicable to sparse polynomial matrices is developed. Two effective structures from [6], which make use of only nonzero addends in polynomial matrices, are used. Partial derivatives on these sparse structures are defined. In the last section we presented illustrative examples and compared various algorithms. Significant improvements with respect to previous results on the set of sparse matrices are observed.

Acknowledgement

Authors wish to thank to anonymous referee for valuable comments improving the quality of the paper. Also, authors gratefully acknowledge the support from the research project 174013 of the Serbian Ministry of Science.

References

[1] A. Ben-Israel and T.N.E. Grevile, *Generalized inverses, Theory and applications, Second edition*, Canadian Mathematical Society, Springer, New York, 2003.

[2] T.N.E. Grevile, *Some applications of the pseudo-inverse of matrix*, SIAM Rev. **3** (1960), 15–22.

[3] D.A. Harville, *Matrix algebra From a Statistician's Perspective*, Springer Science+Business Media, New York, 1997.

[4] R.E. Kalaba, F.E. Udwadia, *Analytical Dynamics: A New Approach*, Cambridge University Press, Cambridge 1996.

[5] B.J. Layton, *Efficient direct computation of the pseudo-inverse and its gradient*, Internat. J. Numer. Methods Engrg. **40** (1997), 4211–4223.

[6] M.D. Petković, P.S. Stanimirović and M.B. Tasić, *Effective partitioning method for computing weighted MoorePenrose inverse*, Comput. Math. Appl. **55** (2008), 1720-1734.

[7] M.D. Petković and P.S. Stanimirović, *Symbolic computation of the Moore-Penrose inverse using partitioning method*, International Journal of Computer Mathematics **82** (2005), 355–367.

[8] P.S. Stanimirović and M.B. Tasić, *Partitioning method for rational and polynomial matrices*, Appl. Math. Comput. **155** (2004), 137–163.

[9] 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.

[10] M.B. Tasić, P.S. Stanimirović, *Symbolic and recursive computation of different types of generalized inverses*, Appl. Math. Comput. **199** (2008), 349–367.

[11] M.B. Tasić, P.S. Stanimirović, *Differentiation of generalized inverses for rational and polynomial matrices*, Appl. Math. Comput. **216** (2010), 2092-2106.

[12] F.E. Udwadia and R.E. Kalaba, *An Alternative Proof for Greville's Formula*, J. Optim. Theory Appl. **94** (1997), 23–28.

[13] F.E. Udwadia and R.E. Kalaba, *A Unified Approach for the Recursive Determination of Generalized Inverses*, Comput.Math. Appl. **37** (1999), 125–130.

[14] G.R. Wang and Y.L.Chen, *A recursive algorithm for computing the weighted Moore-Penrose inverse* $A^{\dagger}_{MN}$, J. Comput. Math. **4** (1986), 74–85.

[15] G. Wang, Y. Wei, S. Qiao, *Generalized inverses: theory and computations*, Science Press, 2003.

[16] S. Wolfram, *The* MATHEMATICA *Book, 5th ed.*, Wolfram Media/Cambridge University Press, Champaign, IL 61820, USA, 2003.

[17] G. Zielke, *Report on test matrices for generalized inverses*, Computing, **36** (1986), 105–162.