Faculty of Sciences and Mathematics
University of Niš

Available at:
www.pmf.ni.ac.yu/sajt/publikacije/publikacije_pocetna.html

Filomat **21:1** (2007), 67–86

# GENERALIZED INVERSION BY INTERPOLATION

**Predrag S. Stanimirović[1], Milan B. Tasić and Predrag V. Krtolica**

**Abstract**

We investigate two algorithms for computing the Moore-Penrose and Drazin inverse of a given one-variable polynomial matrix by interpolation. These algorithms differ in the method used for constant matrices inverses computing. The first algorithm uses the Grevile's method, and the second one uses the Leverrier-Faddeev method and its generalization. These algorithms are especially useful for symbolic computation in procedural programming languages. We compare results by implementing the algorithms in the programming package MATHEMATICA and in the procedural programming languages DELPHI and C++.

## 1 Introduction

For any matrix $A \in \mathbf{C}^{m\times n}$ the Moore-Penrose inverse of $A$ is the unique matrix, denoted by $A^\dagger$, satisfying the following Penrose equations in $X$:

(1) $AXA=A$, (2) $XAX=X$, (3) $(AX)^*=AX$, (4) $(XA)^*=XA$

[1]Corresponding author

[2]*Keywords and phrases.* Pseudoinverse, MATHEMATICA, partitioning method, polynomial matrices, symbolic computation.

[3]2000 *Mathematics Subject Classification.* 15A09, 68Q40.

[4]Received: May 10, 2006

Let $\mathbb{C}$ be the set of complex numbers, $\mathbb{C}^{m\times n}$ be the set of $m\times n$ complex matrices, and $\mathbb{C}_r^{m\times n}=\{X\in\mathbb{C}^{m\times n} : \text{rank}(X)=r\}$. As usual, $\mathbb{C}[s]$ (resp. $\mathbb{C}(s)$) denotes the polynomials (resp. rational expressions) with complex coefficients in the indeterminate $s$. The $m\times n$ matrices with elements in $\mathbb{C}[s]$ (resp. $\mathbb{C}(s)$) are denoted by $\mathbb{C}[s]^{m\times n}$ (resp. $\mathbb{C}(s)^{m\times n}$). By $I_r$ we denote the identity matrix of the order $r$, and by $\mathbb{O}$ an appropriate null matrix is denoted. The Drazin and the Moore-Penrose inverse of $A(s)$ are denoted by $A(s)^D$ and $A(s)^\dagger$, respectively.

An algorithm for computation of the Moore-Penrose inverse of a constant complex matrix $A(s)\equiv A_0\in\mathbb{C}^{m\times n}$ by means of the Leverrier-Faddeev algorithm (also called Souriau-Frame algorithm) is introduced in [2]. A modification of the Leverrier-Faddeev algorithm for computation of the Drazin inverse is given in [5]. Hartwig in [6] continues investigation of this algorithm.

In [8] an implementation of the algorithm for computing the Moore-Penrose inverse of a singular rational matrix in the symbolic computational language MAPLE is described. In [9] and [11] a representation and two algorithms for computation of the Moore-Penrose inverse of a non-regular polynomial matrix of an arbitrary degree are presented. The corresponding algorithm for two-variable polynomial matrix is presented in [10].

An algorithm for computing the Drazin inverse of a rational matrix is introduced in [20, 21]. The corresponding finite algorithm for computation of the Drazin inverse of a given polynomial matrix is given in [7]. Moreover, in the literature a large number of applications of generalized inverses of polynomial matrices [8] is described.

Methods for computing generalized inverses of constant matrices are surveyed, classified, tested and compared in papers [15, 17, 18].

In the second section, we introduce an algorithm for computing the Moore-Penrose inverse of a given one-variable polynomial matrix by the Hermite interpolation. The similar idea is used in [19] to compute the ordinary inverse of the polynomial (non-singular) matrices. Starting from the idea of interpolation we made the algorithm for the generalized inverses calculation. Note that we use a matrix in the symbolic form as an input, evaluating it for some chosen values. Therefore, the introduced algorithm is a generalization of corresponding results from [19].

In the third section, we describe an implementation of the introduced algorithm in the package MATHEMATICA. For computing generalized inverses of constant matrices we use the Grevile's method.

In a recursive implementation of the *partitioning method*, a major problem emerges when dealing with repetitive computations of generalized in-

verses of some matrices as well as the repetitive computations of some vectors. We describe algorithms used for elimination of these difficulties [23].

Algorithm is especially useful for symbolic computation in procedural programming languages. In the fourth section, we describe corresponding implementation based on the application of the procedural programming language DELPHI.

In the next section, we describe the implementation in C++ of the algorithm which uses the Leverrier-Faddeev method and its generalization from [22].

In the last section, several illustrative examples are presented and compare results generated by applying two different implementations in MATHEMATICA, DELPHI and C++.

## 2 Inversion of polynomial matrices by interpolation

It is well known that there is one and only one polynomial of degree $q \leq n$ which assumes the values $f(s_0), f(s_1), \ldots, f(s_n)$ at distinct base points $s_0, s_1, \ldots, s_n$. The polynomial is called the $q$th degree interpolation polynomial. Three important interpolation methods are [19]:

i) the direct approach using Vandermonde's matrix

ii) Newton interpolation,

iii) Lagrange's interpolation.

Generalizing the various interpolation methods to the case of polynomial matrices $A(s)$, algorithms for the inversion of polynomial matrices can be developed.

Investigated Lagrange's interpolation seems to be especially suitable for the inversion of polynomial matrices.

The main idea is the symbolic computation of the Moore-Penrose inverse and the Drazin inverse of one-variable polynomial matrix using interpolation. We know which is the biggest degree of the adjoint matrix and the denominator polynomial. Then we compute the Moore-Penrose inverse and the Drazin inverse for several specific real values of the variable $s$, and find out the matrix interpolation polynomial which will give us the Moore-Penrose or the Drazin inverse of the matrix.

**Algorithm 2.1** *Consider matrix $A(s)$ with respect to unknown $s$.*

- *Step 1. Select distinct base points: $s_0, s_1, \ldots, s_n$, $s_i \in \mathbf{R}$.*

- *Step 2. For $i = 0, 1, \ldots, n$ perform the following:*

  *Step 2.1. Find constants matrices $A_i = A(s_i)$, in each base point $s_i$;*

  *Step 2.2. Compute generalized inverses $A_i^g$ for all constants matrices where $A_i^g$ denotes $A_i^\dagger$ or $A_i^D$;*

- *Step 3. The generalized inverse $A(s)^g$ of a given one-variable polynomial matrix $A(s)$ is equal to*

$$A(s)^g = \sum_{i=0}^{n} A_i^g L_i(s), \tag{1}$$

*where*

$$L_i(s) = \frac{\prod\limits_{\substack{k=0 \\ k\neq i}}^{n} (s - s_k)}{\prod\limits_{\substack{k=0 \\ k\neq i}}^{n} (s_i - s_k)}.$$

In a procedural programming language without the possibility for symbolic computation, it is necessary to transform a given polynomial matrix $A(s) \in \mathbf{S}[s]^{m\times n}$ into the polynomial form with respect to unknown variable $s$:

$$A(s) = B_0 + B_1 s + \cdots + B_{q-1} s^{q-1} + B_q s^q = \sum_{i=0}^{q} B_i s^i \tag{2}$$

where $B_i$, $i = 0, \ldots, q$, $q \leq n$, are constant $m \times n$ matrices.
We also transform $L_i(s)$ in an analogous polynomial form.

$$L_i(s) = l_{i,0} + l_{i,1} s + l_{i,2} s^2 + \cdots + l_{i,q} s^q, \quad i = 0, 1, \ldots, n.$$

Then we have

$$A(s)^\dagger = \sum_{i=0}^{n} A_i^\dagger L_i(s) = \sum_{i=0}^{n} A_i^\dagger \sum_{j=0}^{q} l_{i,j} s^j = \sum_{j=0}^{q} \left( \sum_{i=0}^{n} A_i^\dagger l_{i,j} \right) s^j = \sum_{j=0}^{q} R_j^\dagger s^j \tag{3}$$

where

$$R_j = \sum_{i=0}^{n} A_i^\dagger l_{i,j}, \qquad j = 0, 1, \ldots, q. \tag{4}$$

Now, *Step 3.* of Algorithm 2.1 can be formulated as in the following:

- *Step 3.* Moore-Penrose inverse $A(s)^\dagger$ of given matrix $A(s)$ is

$$A(s)^\dagger = \sum_{j=0}^{q} R_j^\dagger s^j, \quad \text{where} \quad R_j = \sum_{i=0}^{n} A_i^\dagger l_{i,j}, j = 0, 1, \ldots, q. \tag{5}$$

**Proposition.** For this algorithm we supposed that the value of the interpolating polynomial passes trough a small number of points. We use two points $s_0$ and $s_1$ for the implementation in DELPHI. There is a unique line through any two points. In this case we have

$$\begin{aligned} L_0(s) &= \frac{s - s_1}{s_0 - s_1} = \frac{1}{s_0 - s_1} s + \frac{s_1}{s_1 - s_0} = l_{0,1} s + l_{0,0} \qquad (6) \\ L_1(s) &= \frac{s - s_0}{s_1 - s_0} = \frac{1}{s_1 - s_0} s + \frac{s_0}{s_0 - s_1} = l_{1,1} s + l_{1,0} \\ R_0 &= A_0^\dagger l_{0,0} + A_1^\dagger l_{1,0} = A_0^\dagger \frac{s_1}{s_1 - s_0} + A_1^\dagger \frac{s_0}{s_0 - s_1} \qquad (7) \\ R_1 &= A_0^\dagger l_{0,1} + A_1^\dagger l_{1,1} = A_0^\dagger \frac{1}{s_0 - s_1} + A_1^\dagger \frac{1}{s_1 - s_0} \end{aligned}$$

## 3 Implementation in package MATHEMATICA

Grevile in [4] proposed a recursive algorithm which relates the Moore-Penrose pseudoinverse of a matrix $R$ augmented by an appropriate vector $r$ with the pseudoinverse $R^\dagger$ of $R$.

In view of the Grevile's algorithm we present the following algorithm for computation of the Moore-Penrose inverse of a given constant matrix.

**Algorithm 3.1** *Consider any $m \times n$ constant matrix $A$. Let $\widehat{A}_i$ be the submatrix of $A$ consisting of first $i$ columns of $A$. If $i$-th column of $A$ is denoted by $a_i$, then $\widehat{A}_i$ is partitioned as $\widehat{A}_i = \left[\widehat{A}_{i-1} | a_i\right]$, $i = 2, \ldots, n$, $\widehat{A}_1 = a_1$.*

- *Step 1. Initial value:*

$$\widehat{A}_1^\dagger = a_1^\dagger = \begin{cases} \frac{1}{a_1^* a_1} a_1^*, & a_1 \neq 0, \\ a_1^*, & a_1 = 0. \end{cases}$$

- *Step 2. Recursive step: For each $i = 2, \ldots, n$ compute*

$$\widehat{A}_i^\dagger = \begin{bmatrix} \widehat{A}_{i-1}^\dagger - d_i b_i^* \\ \\ b_i^* \end{bmatrix}$$

*where the quantities $b_i$ and $d_i$ are defined in steps 2.1-2.3.*

*Step 2.1.* $d_i = \widehat{A}_{i-1}^{\dagger} a_i$,

*Step 2.2.* $c_i = a_i - \widehat{A}_{i-1} d_i = \left(I - \widehat{A}_{i-1}\widehat{A}_{i-1}^{\dagger}\right) a_i$,

*Step 2.3.* $b_i = \begin{cases} \frac{1}{c_i^* c_i} c_i, & c_i \neq 0 \\ \frac{1}{1+d_i^* d_i}(\widehat{A}_{i-1}^{\dagger})^* d_i, & c_i = 0. \end{cases}$

- *Step 3. The stopping criterion:* $A^{\dagger} = \widehat{A}_n^{\dagger}$.

In [17] the Grevile's algorithm is estimated as relatively complicated and numerically stable.

In the beginning we describe implementation in MATHEMATICA.

*A.* The function $Col[a, j]$ extracts $j$-th column of the matrix $a = A(s)$:

```
Col[a_List,j_]:=Transpose[{Transpose[a][[j]]}]
```

*B.* The submatrix $A_j(s) = [a_1(s), \cdots a_j(s)]$ which contains first $j \leq n$ columns of the matrix $A(s) = A_n(s) = [a_1(s), \cdots a_n(s)]$ is generated as follows:

```
ACompl[a_List,j_]:=Module[{m,n},
  {m,n}=Dimensions[a];
   Return[Transpose[Drop[Transpose[a],-(n-j)]]]]
```

Beside the simplification, a significant problem in a recursive implementation of *Algorithm 3.1*, is the magnification of arithmetic operations. This problem arises from multiplicative recomputations. In view of *Step 2* in *Algorithm 3.1*, for each $i \in \{2, \ldots, n\}$, the Moore-Penrose inverse $\widehat{A}_i(s)^{\dagger}$ must be computed $n - i + 1$ times. Moreover, in view of *Step 2.1* and *Step 2.3*, the pseudoinverse $\widehat{A}_{i-1}(s)^{\dagger}$ is needed during the computation of the values $d_i(s)$ and $b_i(s)$. Consequently, *Algorithm 3.1* requires $3(n - i + 1)$ recomputations of the Moore-Penrose inverse $\widehat{A}_i(s)^{\dagger}$, for each $i \in \{2, \ldots, n\}$. The total number of different values that will be produced is comparatively small, but these values must be recomputed many times. In order to obviate this problem, we use possibility of the programming package MATHEMATICA to define functions which remember values that they generate [24, 25].

*Step 2* of the *Algorithm 3.1* is implemented in the following functions which remember already computed values.

Implementation of *Step 2.1*.

```
DD[a_List,i_]:=DD[a,i]=
 Module[{s={}},
  s=Simplify[A[a,i-1].Col[a,i]];
  If[Length(s)==1, Return[s[[1]]], Return(s)]
```

Implementation of *Step 2.2*.

```
CC[a_List,i_]:=CC[a,i]=
 Module[{s={},vr},
   vr=Variables[DD[a,i]];
   If[vr=!={},
       If[Length[Dimensions[DD[a,i]]]==1,
          s=Col[a,i]-ACompl[a,i-1]DD[a,i],
          s=Col[a,i]-ACompl[a,i-1].DD[a,i]],
       If[Length[DD[a,i]]==0,
          s=Col[a,i]-ACompl[a,i-1]DD[a,i],
          s=Col[a,i]-ACompl[a,i-1].DD[a,i]]];
   Return[Simplify(s)]]
```

Implementation of *Step 2.3*.

```
B[a_List,i_]:=B[a,i]=
 Module[{nul,m1,j,k,n1,s={}},
  {m1,n1}=Dimensions[CC[a,i]];
  nul=Table[0,{j,1,m1},{k,1,n1}];
  If[CC[a,i]=!=nul,
    s=(1/(Transpose[CC[a,i]].CC[a,i])[[1,1]])CC[a,i],
    If[Length[Dimensions[DD[a,i]]]==1,
     s=(1/(1+DD[a,i]DD[a,i]))Transpose[{A[a,i-1]}]DD[a,i],
     s=(1/(1+Transpose[DD[a,i]].DD[a,i])[[1,1]])
        Transpose[A[a,i-1]].DD[a,i]]];
  Return[Simplify(s)]]
```

Implementation of *Step 1*, *Step 2* and *Step 3*.

```
A[a_List,i_]:=A[a,i]=
 Module[{b=a,vr},
  If[i==1,
     If[Col[a,1]==Col[a,1]*0, b=Transpose[a][[1]],
        b=(1/(Transpose[a][[i]].Col[a,i])[[1]])
           Transpose[a][[1]]],
     vr=Variables[DD[a,i]];
     If[vr=!={},
        If[Length[Dimensions[DD[a,i]]]==1,
           b={A[a,i-1]}-DD[a,i]Transpose[B[a,i]],
           b=A[a,i-1]-DD[a,i].Transpose[B[a,i]]],
        If[Length[DD[a,i]]==0,
           b={A[a,i-1]}-DD[a,i]Transpose[B[a,i]],
           b=A[a,i-1]-DD[a,i].Transpose[B[a,i]]]];
     b=Append[b,Transpose[B[a,i]][[1]]]];
  Return[Simplify[b]]]
```

The following function starts recursive computations:

```
Partitioning[a_List]:= Block[{m,n},{m,n}=Dimensions[a];A[a,n]]
```

Lagrange's interpolation described in *Step 3* of *Algorithm 2.1* can be implemented as follows.

```
Lagr[s_List, var_List, k_]:=
    Module[{n=Length[s],p},
      p=Product[(var[[1]]-s[[i]])/(s[[k]]-s[[i]]),{i,1,k-1}]*
        Product[(var[[1]]-s[[i]])/(s[[k]]-s[[i]]),{i,k+1,n}];
    Return[Simplify[p]]]
```

Finally, the Moore-Penrose inverse of a given matrix $A(s)$ can be found by using the following:

```
Interpol[A_List, s_List] :=
    Module[{n=Length[s],var=Variables[A]},
        Do[A1[i]=A/.var[[1]]->s[[i]],{i,1,n}];
        Do[A2[i]=Partitioning[A1[i]],{i,1,n}];
        A3=Sum[A2[i]*Lagr[s,var,i],{i,1,n}];
    Return[Expand[A3]//N]]
```

Instead of described function $Partitioning[]$ it is possible to use the standard MATHEMATICA function $PseudoInverse[]$. But this function with matrices of larger dimensions requires much more time for computation with respect to our function $Partitioning[]$. Also, in procedural programming language such as DELPHI there is no standard function for computing generalized inverses. Therefore, Grevile's recursive algorithm is needed.

## 4 Implementation in DELPHI

A common problem arising in the implementation of various methods for numerical or symbolic computation of generalized inverses is a dramatic magnification of floating point operations.

In a recursive implementation of the *partitioning method* [4], the repetitive computation of generalized inverses of some matrices is a major problem.

We describe corresponding algorithms to eliminate these difficulties [23].

A major problem which occurs in a recursive implementation of *Algorithm 3.1* arises from multiplicative recomputations of several matrices. In view of *Step 2* of *Algorithm 3.1*, for each $i \in \{2, \ldots, n\}$, the Moore-Penrose inverse $A_i^\dagger$ must be computed $n - i + 1$ times. Moreover, in view of *Step 2.1* and *Step 2.3*, it is also necessary to generate the inverse $A_{i-1}^\dagger$ in the computation of the values $d_i$ and $b_i$.

The total number of different values that will be produced is comparatively small, but these values must be recomputed many times.

We define a type of date as follows:

```
type  Matrix = record
                m,n:integer;
                Element:array[1..80,1..80] of real;
               end;
      ArrayMatrix=array[1..10] of Matrix;

public A,R,Z:ArrayMatrix;
```

Firstly, we describe a few auxiliary procedures for elementary matrix transformations.

Generate the $i$-th column $a_i$ of $A$:

```
function TForm1.ithCol(i:integer; A:Matrix):Matrix;
var R:Matrix;k:integer; begin
 R.n:=1; R.m:=A.m;
 for k:=1 to A.m do R.Element[k,1]:=A.Element[k,i];
 ithCol:=R;
end;
```

The submatrix $A_j = [a_1, \cdots a_j]$ which contains first $j \leq n$ columns of the matrix $A = A_n = [a_1, \cdots a_n]$ is generated as follows:

```
function TForm1.FirstiCol(i:integer; A:Matrix):Matrix;
var R:Matrix;b,k:integer;
begin
 R.n:=i; R.m:=A.m;
 for b:=1 to i do
   for k:=1 to A.m do R.Element[k,b]:=A.Element[k,b];
 FirstiCol:=R;
end;
```

The function `isZero(A:Matrix):boolean` returns $True$ if $A$ is the zero matrix, and $False$ othervise.

The function `MatNum(A:Matrix)` returns $A[1,1]$, in the case when $A$ is $1 \times 1$ matrix.

In the function $AppRow(X, Y : Matrix)$ it is assumed that $X$ is $m \times n$ matrix and $Y$ is a row $1 \times n$ matrix. The result is $(m+1) \times n$ matrix, obtained by appending $Y$ after the last row in $X$.

```
function TForm1.AppRow(X,Y:Matrix):Matrix;
var j:integer;
begin
 for j:=1 to X.n do X.Element[X.m+1,j]:=Y.Element[1,j];
 X.m:=X.m+1;
 AppRow:=X;
end;
```

The function `ZeroMat(m,n:integer)` generates $m \times n$ zero matrix;

The function $SubsMat(X, Y : Matrix)$ gives the matrix $X - Y$;

The function $MultSk(k : Real; A : Matrix)$ gives the product of the matrix $A$ with the scalar $k$.

The result of function `Trans(A:Matrica)` is the transpose of $A$;

The result of the function `MultMat(X,Y:Matrix)` is the product of two matrices $X$ and $Y$.

Grevile's algorithm for calculating the Moore-Penrose inverse is implemented in the following function `Grevile()`.

```
function TForm1.Grevile(A:Matrix):Matrix;
var P,AR,AA,BB,CC,DD:Matrix; pom:Real;i:integer;
begin
 {Step 1}AA:=ithCol(1,A);
  if isZero(AA) then AR:=Trans(AA)
   else begin
             AR:=MultMat(Trans(AA),AA); pom:=MatNum(AR);
             AR:=MultSk(1/pom,Trans(AA));
           end;
{Step 2} for i:=2 to A.n do begin
   {Step 2.1} DD:=MultMat(AR,ithCol(i,A));
   {Step 2.2} CC:=SubsMat(ithCol(i,A),MultMat(FirstiCol(i-1,A),DD));
   {Step 2.3} if isZero(CC) then  begin
     P:=MultMat(Trans(DD),DD); pom:=1+MatNum(P);
     BB:=MultSk(1/pom,MultMat(Trans(AR),DD));
  end else begin
            P:=MultMat(Trans(CC),CC);pom:=MatNum(P);BB:=MultSk(1/pom,CC);
            end;
   {Step 3}AR:=SubsMat(AR,MultMat(DD,Trans(BB)));AR:=AppRow(AR,Trans(BB));
 end;
 Grevile:=AR;
end;
```

Finally, inversion of polynomial matrices by interpolation using the modified *Algorithm 2.1* in *Step 3.* with two starting points $s_0$ and $s_1$, (see (6) and (7)) is given as

```
procedure TForm1.InterpolClick(Sender: TObject);
var t1,t2:real;
begin
    t1:=strtofloat(first.Text); {t1 is the first interpolation point}
    t2:=strtofloat(seconds.Text); {t2 is the second interpolation point}
    Z[1]:=Grevile(SubsMat(MultSk(t1,A[1]),MultSk(-1,A[2])));
    Z[2]:=Grevile(SubsMat(MultSk(t2,A[1]),MultSk(-1,A[2])));
    R[1]:=SubsMat(MultSk(1/(t1-t2),Z[1]),MultSk(1/(t1-t2),Z[2]));
    R[2]:=SubsMat(MultSk(t2/(t2-t1),Z[1]),MultSk(t1/(t2-t1),Z[2]));
end;
```

## 5 Implementation in C++

In this case, computation of the values $A(s_1), \cdots, A(s_n)$ is based on the reverse Polish notation of matrix $A(s)$ elements. Let us recall that the single-variable polynomials are elements of the given matrix. After the conversion of these polynomials into the postfix form we are able to find out their values in the points $s_1, \ldots, s_n$, using two stacks: one is stack of strings and the other is stack of numerical values.

In *Step 2*, using Leverrier-Faddeev algorithm and generalization from [22], we calculate the "interpolation points", i.e. real matrices $A_k$.

Finally, in *Step 3*, we use algorithm for polynomial interpolation (e.g. form [16]), restated for matrix polynomials. We get matrices $C_k \in \mathbb{R}^{n\times m}$ satisfying

$$\sum_{k=1}^{n} A_k l_k(s) = \sum_{k=0}^{n-1} C_k s^k.$$

Obtained matrices $C_k$ contain polynomial coefficients of the needed inverse matrix elements. After that, the inverse matrix could be easily expressed in the polynomial form.

## 6 Examples

In the beginning of this section we give several examples related to packages MATHEMATICA and DELPHI.

**Example 6.1** *The following test example is known as* $S_{11}$ *from* [26]*:*

$$S11 = \begin{pmatrix} a+1 & a & a & a & a & a & a & a & a & a & a+1 \\ a & a-1 & a & a & a & a & a & a & a & a & a \\ a & a & a+1 & a & a & a & a & a & a & a & a \\ a & a & a & a-1 & a & a & a & a & a & a & a \\ a & a & a & a & a+1 & a & a & a & a & a & a \\ a & a & a & a & a & a-1 & a & a & a & a & a \\ a & a & a & a & a & a & a+1 & a & a & a & a \\ a & a & a & a & a & a & a & a-1 & a & a & a \\ a & a & a & a & a & a & a & a & a+1 & a & a \\ a & a & a & a & a & a & a & a & a & a-1 & a \\ a+1 & a & a & a & a & a & a & a & a & a & a+1 \end{pmatrix}.$$

MATHEMATICA *function* $Partitioning[S11]$ *produces well known inverse* $S_{11}^{\dagger}$*:*

$$\mathbf{Out}[1] := \begin{pmatrix} \frac{1-a}{4} & \frac{a}{2} & -\frac{a}{2} & \frac{a}{2} & -\frac{a}{2} & \frac{a}{2} & -\frac{a}{2} & \frac{a}{2} & -\frac{a}{2} & \frac{a}{2} & \frac{1-a}{4} \\ \frac{a}{2} & -1-a & a & -a & a & -a & a & -a & a & -a & a \\ -\frac{a}{2} & a & 1-a & a & -a & a & -a & a & -a & a & -a \\ \frac{a}{2} & -a & a & -1-a & a & -a & a & -a & a & -a & a \\ -\frac{a}{2} & a & -a & a & 1-a & a & -a & a & -a & a & -a \\ \frac{a}{2} & -a & a & -a & a & -1-a & a & -a & a & -a & a \\ -\frac{a}{2} & a & -a & a & -a & a & 1-a & a & -a & a & -a \\ \frac{a}{2} & -a & a & -a & a & -a & a & -1-a & a & -a & a \\ -\frac{a}{2} & a & -a & a & -a & a & -a & a & 1-a & a & -a \\ \frac{a}{2} & -a & a & -a & a & -a & a & -a & a & -1-a & a \\ \frac{1-a}{4} & \frac{a}{2} & -\frac{a}{2} & \frac{a}{2} & -\frac{a}{2} & \frac{a}{2} & -\frac{a}{2} & \frac{a}{2} & -\frac{a}{2} & \frac{a}{2} & \frac{1-a}{4} \end{pmatrix}$$

Applying *Algorithm 2.1* and function $Interpol[[S11, \{1, 2\}]]$, developed in DELPHI, with the starting point $\{1, 2\}$, we get the result:

$$\begin{aligned}
&\{\{0.25 - 0.25\,a, 0.5\,a, -0.5\,a, 0.5\,a, -0.5\,a, 0.5\,a, -0.5\,a, 0.5\,a, -0.5\,a, 0.5\,a, 0.25 - 0.25\,a\},\\
&\{0.5\,a, -1. - 1.\,a, a, -1.\,a, a, -1.\,a, a, -1.\,a, a, -1.\,a, 0.5\,a\},\\
&\{-0.5\,a, a, 1. - 1.\,a, a, -1.\,a, a, -1.\,a, a, -1.\,a, a, -0.5\,a\},\\
&\{0.5\,a, -1.\,a, a, -1. - 1.\,a, a, -1.\,a, a, -1.\,a, a, -1.\,a, 0.5\,a\},\\
&\{-0.5\,a, a, -1.\,a, a, 1. - 1.\,a, a, -1.\,a, a, -1.\,a, a, -0.5\,a\},\\
&\{0.5\,a, -1.\,a, a, -1.\,a, a, -1. - 1.\,a, a, -1.\,a, a, -1.\,a, 0.5\,a\},\\
&\{-0.5\,a, a, -1.\,a, a, -1.\,a, a, 1. - 1.\,a, a, -1.\,a, a, -0.5\,a\},\\
&\{0.5\,a, -1.\,a, a, -1.\,a, a, -1.\,a, a, -1. - 1.\,a, a, -1.\,a, 0.5\,a\},\\
&\{-0.5\,a, a, -1.\,a, a, -1.\,a, a, -1.\,a, a, 1. - 1.\,a, a, -0.5\,a\},\\
&\{0.5\,a, -1.\,a, a, -1.\,a, a, -1.\,a, a, -1.\,a, a, -1. - 1.\,a, 0.5\,a\},\\
&\{0.25 - 0.25\,a, 0.5\,a, -0.5\,a, 0.5\,a, -0.5\,a, 0.5\,a, -0.5\,a, 0.5\,a, -0.5\,a, 0.5\,a, 0.25 - 0.25\,a\}\}
\end{aligned}$$

In the following table we show effectiveness of Algorithm 2.1 with respect to MATHEMATICA and DELPHI. Consider the matrices known as $S_n$, $A_n$ and $F_n$ from [26], for the interpolating point $\{1, 2\}$ of the parameter $s$. We show the effectiveness with respect to CPU time. These results (in seconds) are obtained using version 4.0 for MATHEMATICA and DELPHI 6.0 at Pentium II 433 MHz.

| Dimension | Test matrix | MATHEMATICA | DELPHI |
|---|---|---|---|
| 4 | S | 0.22 | 0.51 |
| 4 | A | 0.22 | 0.51 |
| 4 | F | 0.22 | 0.51 |
| 11 | S | 0.27 | 1.01 |
| 11 | A | 0.27 | 1.01 |
| 11 | F | 0.27 | 1.01 |
| 35 | S | 4.50 | 2.01 |
| 35 | A | 2.52 | 2.01 |
| 35 | F | 2.47 | 2.01 |
| 50 | S | 12.64 | 3.01 |
| 50 | A | 6.09 | 3.01 |
| 50 | F | 5.54 | 3.01 |
| 79 | S | 48.17 | 5.01 |
| 79 | A | 17.52 | 5.01 |
| 79 | F | 16.31 | 5.01 |

Table 1.

Standard MATHEMATICA function $PseudoInverse[]$ for the test matrix dimension greater than 9, could not be evaluated for a very long time. Thus, we conclude that Algorithm 3.1 is faster with less memory cost.

We now arrange examples related to MATHEMATICA and C++.

**Example 6.2** *Let us consider the matrix from* [26]

$$H_5(s) = \begin{bmatrix} a+1 & a+2 & a+2 & a+3 & a+4 \\ a+2 & a+2 & a+3 & a+4 & a+5 \\ a+2 & a+3 & a+4 & a+5 & a+6 \\ a+3 & a+4 & a+5 & a+5 & a+6 \\ a+4 & a+5 & a+6 & a+6 & a+7 \\ a & a+1 & a+1 & a+2 & a+3 \\ a-1 & a-1 & a-1 & a+1 & a+2 \end{bmatrix}.$$

*Using values $s_i = i, i = 1, 2$, we get the following array of matrices representing polynomial coefficients in the Moore-Penrose inverse $H_5^{\dagger}(s)$. Note that $\approx 0$ means that corresponding value is approximate to zero (order $10^{-14}$ or less):*

$$\begin{bmatrix} 0.333333 & 0.666667 & -1 & -0.333333 & 0.666667 & -0.666667 & 0.333333 \\ 1.33333 & -0.833333 & \approx 0 & -0.833333 & 0.666667 & -0.166667 & -0.166667 \\ -1.83333 & 0.333333 & \approx 0 & 1.83333 & -1.16667 & 1.16667 & -0.333333 \\ 1.33333 & -0.833333 & 3 & -2.83333 & 0.666667 & -2.16667 & -0.166667 \\ -0.666667 & 0.666667 & -2 & 1.66667 & -0.333333 & 1.33333 & 0.333333 \end{bmatrix},$$

$$\begin{bmatrix} \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 & 0 \\ \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 \\ \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 \\ 0.5 & \approx 0 & \approx 0 & -0.5 & 0.5 & -0.5 & \approx 0 \\ -0.5 & \approx 0 & 0 & 0.5 & -0.5 & 0.5 & \approx 0 \end{bmatrix}.$$

*In* [26] *the exact value of pseudoinverse matrix is:*

$$H_5^{\dagger}(s) = \frac{1}{12}\begin{bmatrix} 4 & 8 & -12 & -4 & 8 & -8 & 4 \\ 16 & -10 & 0 & -10 & 8 & -2 & -2 \\ -22 & 4 & 0 & 22 & -14 & 14 & -4 \\ 6a+16 & -10 & 36 & -6a-34 & 6a+8 & -6a-26 & -2 \\ -6a-8 & 8 & -24 & 6a+20 & -6a-4 & 6a+16 & 4 \end{bmatrix}.$$

*If we consider extremely small values (order* $10^{-14}$ *and less) as zeroes, then we can conclude that result is correct (within certain rounding error).*

**Example 6.3** *Let us consider another matrix from* [26]

$$F_4(s) = \begin{bmatrix} s+4 & s+3 & s+2 & s+1 \\ s+3 & s+3 & s+2 & s+1 \\ s+2 & s+2 & s+1 & s \\ s+1 & s+1 & s & s-1 \end{bmatrix}.$$

*Using values* $s_i = i, i = 1, 2$, *we get the following array of matrices representing polynomial coefficients in the Moore-Penrose inverse* $F_4^{\dagger}(s)$*:*

$$\begin{bmatrix} 1 & -0.833333 & -0.333333 & 0.166667 \\ -0.833333 & 0.777778 & 0.444444 & 0.111111 \\ -0.333333 & 0.444444 & 0.111111 & -0.222222 \\ 0.166667 & 0.111111 & -0.222222 & -0.555556 \end{bmatrix},$$

$$\begin{bmatrix} \approx 0 & \approx 0 & \approx 0 & \approx 0 \\ \approx 0 & -0.25 & \approx 0 & 0.25 \\ \approx 0 & \approx 0 & \approx 0 & 0 \\ \approx 0 & 0.25 & \approx 0 & -0.25 \end{bmatrix}.$$

*The exact pseudoinverse matrix is (*[26]*):*

$$F_4^{\dagger}(s) = \begin{bmatrix} 1 & -\frac{5}{6} & -\frac{1}{3} & \frac{1}{6} \\ -\frac{5}{6} & \frac{7}{9} - \frac{s}{4} & \frac{4}{9} & \frac{1}{9} + \frac{s}{4} \\ -\frac{1}{3} & \frac{4}{9} & \frac{1}{9} & -\frac{2}{9} \\ \frac{1}{6} & \frac{1}{9} + \frac{s}{4} & -\frac{2}{9} & -\frac{5}{9} - \frac{s}{4} \end{bmatrix}.$$

*When we use the generalization from* [22]*, Theorem 2.1, (i) and (ii), we get the same result as above (in this and some other cases the Moore-Penrose and Drazin inverses are equal). When we use (iv) for our method we get*

$$A(s)A(s)^D = \begin{bmatrix} 1 & \approx 0 & \approx 0 & \approx 0 \\ \approx 0 & 0.833333 & 0.333333 & -0.166667 \\ \approx 0 & 0.333333 & 0.333333 & 0.333333 \\ \approx 0 & -0.166667 & 0.333333 & 0.833333 \end{bmatrix},$$

**Example 6.4** *Consider the singular square matrix from* [26] *for* $n = 7$*.*

$$S_7(a) = \begin{bmatrix} a+1 & a & a & a & a & a & a+1 \\ a & a-1 & a & a & a & a & a \\ a & a & a+1 & a & a & a & a \\ a & a & a & a-1 & a & a & a \\ a & a & a & a & a+1 & a & a \\ a & a & a & a & a & a-1 & a \\ a+1 & a & a & a & a & a & a+1 \end{bmatrix}.$$

*Using values* $s_i = i, i = 1, 2$*, we get the following array of matrices representing polynomial coefficients in the Moore-Penrose inverse* $S_7^{\dagger}(a)$*:*

$$\begin{bmatrix} 0.25 & 0 & 0 & 0 & 0 & 0 & 0.25 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0.25 & 0 & 0 & 0 & 0 & 0 & 0.25 \end{bmatrix},$$

$$\begin{bmatrix} -0.25 & 0.5 & -0.5 & 0.5 & -0.5 & 0.5 & -0.25 \\ 0.5 & -1 & 1 & -1 & 1 & -1 & 0.5 \\ -0.5 & 1 & -1 & 1 & -1 & 1 & -0.5 \\ 0.5 & -1 & 1 & -1 & 1 & -1 & 0.5 \\ -0.5 & 1 & -1 & 1 & -1 & 1 & -0.5 \\ 0.5 & -1 & 1 & -1 & 1 & -1 & 0.5 \\ -0.25 & 0.5 & -0.5 & 0.5 & -0.5 & 0.5 & -0.25 \end{bmatrix}.$$

*In* [26] *the exact pseudoinverse matrix is:*

$$S_7^{\dagger}(a) = \frac{1}{4}\begin{bmatrix} -a+1 & 2a & -2a & 2a & -2a & 2a & -a+1 \\ 2a & -4a-4 & 4a & -4a & 4a & -4a & 2a \\ -2a & 4a & -4a+4 & 4a & -4a & 4a & -2a \\ 2a & -4a & 4a & -4a-4 & 4a & -4a & 2a \\ -2a & 4a & -4a & 4a & -4a+4 & 4a & -2a \\ 2a & -4a & 4a & -4a & 4a & -4a-4 & 2a \\ -a+1 & 2a & -2a & 2a & -2a & 2a & -a+1 \end{bmatrix}.$$

**Example 6.5** *Finally, for matrix* $A_2$ *of rank two and dimensions* $7\times 6$ *from* [26]

$$A_2 = \begin{bmatrix} a+10 & a+9 & a+8 & a+7 & a+6 & a+5 \\ a+9 & a+8 & a+7 & a+6 & a+5 & a+4 \\ a+8 & a+7 & a+6 & a+5 & a+4 & a+3 \\ a+7 & a+6 & a+5 & a+4 & a+3 & a+2 \\ a+6 & a+5 & a+4 & a+3 & a+2 & a+1 \\ a+5 & a+4 & a+3 & a+2 & a+1 & a \\ a+4 & a+3 & a+2 & a+1 & a & a-1 \end{bmatrix}.$$

*we get*

$$\begin{bmatrix} -0.0306122 & -0.0136054 & 0.00340136 & 0.0204082 & 0.037415 & 0.0544218 & 0.0714286 \\ -0.0112245 & -0.00340136 & 0.00442177 & 0.0122449 & 0.020068 & 0.0278912 & 0.0357143 \\ 0.00816327 & 0.00680272 & 0.00544218 & 0.00408163 & 0.00272109 & 0.00136054 & \approx 0 \\ 0.027551 & 0.0170068 & 0.00646259 & -0.00408163 & -0.0146259 & -0.0251701 & -0.0357143 \\ 0.0469388 & 0.0272109 & 0.00748299 & -0.0122449 & -0.0319728 & -0.0517007 & -0.0714286 \\ 0.0663265 & 0.037415 & 0.0085034 & -0.0204082 & -0.0493197 & -0.0782313 & -0.107143 \end{bmatrix},$$

$$\begin{bmatrix} -0.0153061 & -0.0102041 & -0.00510204 & \approx 0 & 0.00510204 & 0.0102041 & 0.0153061 \\ -0.00918367 & -0.00612245 & -0.00306122 & \approx 0 & 0.00306122 & 0.00612245 & 0.00918367 \\ -0.00306122 & -0.00204082 & -0.00102041 & \approx 0 & 0.00102041 & 0.00204082 & 0.00306122 \\ 0.00306122 & 0.00204082 & 0.00102041 & \approx 0 & -0.00102041 & -0.00204082 & -0.00306122 \\ 0.00918367 & 0.00612245 & 0.00306122 & \approx 0 & -0.00306122 & -0.00612245 & -0.00918367 \\ 0.0153061 & 0.0102041 & 0.00510204 & \approx 0 & -0.00510204 & -0.0102041 & -0.0153061 \end{bmatrix}.$$

*In* [26] *the exact pseudoinverse matrix is:*

$$A_2^{\dagger} = \frac{1}{20580}\times$$

$$\begin{bmatrix} -315a-639 & -210a-280 & -105a+70 & 420 & 105a+770 & 210a+1120 & 315a+1470 \\ -189a-231 & -126a-70 & -63a+91 & 252 & 63a+413 & 126a+574 & 189a+735 \\ -63a-168 & -42a+140 & -21a+112 & 84 & 21a+56 & 42a+28 & 63a \\ 63a+567 & 42a+350 & 21a+133 & -84 & -21a-301 & -42a-518 & -63a-735 \\ 189a+966 & 126a+560 & 63a+154 & -252 & -63a-658 & -126a-1064 & -189a-1470 \\ 315a+1365 & 210a+770 & 105a+175 & -420 & -105a-1015 & -210a-1610 & -315a-2205 \end{bmatrix}.$$

Programming package MATHEMATICA has the capability of the generalized inverse calculation. In the following table we exhibit the behavior of our algorithm in comparison with MATHEMATICA function expressed in execution times (in seconds) calculating the inverse of $S_n$ matrix. Corresponding values are generated using version 4.0 for MATHEMATICA and Pentium II 1.2 GHz.

| $n$ | Our algorithm | MATHEMATICA function |
|---|---|---|
| 70 | 1 | 10.391 |
| 75 | 2 | 12.859 |
| 80 | 3 | 15.188 |
| 85 | 3 | 18.640 |
| 90 | 3 | 21.625 |

Table 2.

## 7 Conclusions

We investigate the problem of symbolic computation of the Moore-Penrose and Drazin inverses of one-variable polynomial matrix. We combine the interpolation with usage of the Grevile's recursive algorithm, the Leverrier-Faddeev algorithm with and without the generalization form [22]. Generalizing the interpolation methods to the case of polynomial matrices $A(s)$, we develop the algorithms for the inversion of polynomial matrices.

Both methods start from the idea of interpolation similar to the one from [19]. Note that in the second method we use a matrix in symbolic form as input, evaluating it for some chosen values.

## References

[1] S. Barnett, *Leverrier's algorithm: a new proof and extensions*, SIAM J. Matrix Anal. Appl. **10** (1989) 551–556.

[2] H. P. Decell, *An application of the Cayley-Hamilton theorem to generalized matrix inversion*, SIAM Review **7** No 4 (1965) 526–528.

[3] G. Fragulis, B. G. Mertzios, and A. I. G. Vardulakis, *Computation of the inverse of a polynomial matrix and evaluation of its Laurent expansion*, Int. J. Control **53** (1991) 431–443.

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

[5] T. N. E. Grevile, *The Souriau-Frame algorithm and the Drazin pseudoinverse*, Linear Algebra Appl. **6** (1973) 205–208.

[6] R. E. Hartwig, *More on the Souriau-Frame algorithm and the Drazin inverse*, SIAM J. Appl. Math. **31** No 1 (1976) 42–46.

[7] J. Ji, *A finite algorithm for the Drazin inverse of a polynomial matrix*, Appl. Math. Comput. **130** (2002) 243–251.

[8] J. Jones, N. P. Karampetakis, and A. C. Pugh, *The computation and application of the generalized inverse via Maple*, J. Symbolic Computation **25** (1998) 99–124.

[9] N. P. Karampetakis, *Computation of the generalized inverse of a polynomial matrix and applications*, Linear Algebra Appl. **252** (1997) 35–60.

[10] N. P. Karampetakis, *Generalized inverses of two-variable polynomial matrices and applications*, Circuits Systems Signal Processing **16** (1997) 439–453.

[11] N. P. Karampetakis and P. Tzekis, *On the computation of the generalized inverse of a polynomial matrix*, Ima Journal of Mathematical Control and Information **18** (2001) 83–97.

[12] V. Lovass, R. Miller and D. Powers, *Transfer function matrix synthesis by matrix generalized inverses*, Int. J. Control **27** (1978) 387–391.

[13] V. Lovass, R. Miller and D. Powers, *Further results on output control in the servomechanism sence*, Int. J. Control **27** (1978) 133–138.

[14] V. Lovass, R. Miller and D. Powers, *An introduction to the application of the simplest matrix-generalized inverse in system science*, IEEE Trans. Auto. Control **25** (1978) 766–771.

[15] B. Noble, *Methods for computing the Moore-Penrose generalized inverse, and related matters*, Generalized Inverses and Applications edited by M.Z. Nashed, Academic Press, New York (1976), 245–301.

[16] W. H. Press, S. A. Teukolsky, W. T. Wetterling and B. P. Flannery, *Numerical receipts in C*, Cambridge University Press, Cambridge (MA), 1992.

[17] N. Shinozaki, M. Sibuya and K. Tanabe, *Numerical algorithms for the Moore-Penrose inverse of a matrix: direct methods*, Annals of the Institute of Statistical Mathematics **24**, No. 1 (1972) 193–203.

[18] N. Shinozaki, M. Sibuya and K. Tanabe, *Numerical algorithms for the Moore-Penrose inverse of a matrix: iterative methods*, Annals of the Institute of Statistical Mathematics **24**, No. 1 (1972) 621–629.

[19] A. Schuster and P. Hippe, *Inversion of Polynomial Matrices by Interpolation*, IEEE Transactions on Automatic control **37**, No. 3 (March 1992) 363–365.

[20] P. S. Stanimirović and N. P. Karampetakis, *Symbolic implementation of Leverrier-Faddeev algorithm and applications*, 8th IEEE Medit. Conference on Control and Automation, Patra, Greece, 2000.

[21] P. S. Stanimirović and N. P. Karampetakis, *On the computation of Drazin inverse of a rational matrix and applications*, Technical Report, Department of Mathematics, Aristotle University of Thessaloniki, Thessaloniki 54006, Greece, 2000.

[22] P. S. Stanimirović, *A finite algorithm for generalized inverses of polyinomial and rational matrices*, Appl. Math. Comput. **144** (2003) 199–214.

[23] P. S. Stanimirović and M. B. Tasić, *A problem in computation of pseudoinverses*, Appl. Math. Comput. **135** (2-3) (2003) 443–469.

[24] S. Wolfram, *Mathematica: a system for doing mathematics by computer*, Addison-Wesley Publishing Co., Redwood City, California, 1991.

[25] S. Wolfram, *Mathematica Book, Version 3.0*, Wolfram Media and Cambridge University Press, 1996.

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

Address

Predrag Stanimirović, Milan Tasić and Predrag Krtolica:

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

*E-mail*: pecko@pmf.pmf.ni.ac.yu, milan12t@ptt.yu, krca@pmf.ni.ac.yu