FACTA UNIVERSITATIS (NIŠ)
SER. MATH. INFORM. 25 (2010), 25–33

IMPLEMENTATION OF THE PARTITIONING METHOD*

Milan B. Tasić and Ivan P. Stanimirović

Abstract. We investigate various implementations of Greville's partitioning method for computing generalized inverses from [1]. This method is implemented in the completely different software environments: in symbolic package MATHEMATICA and procedural programming language DELPHI. In this way some files are constructed during the running time to memorize all computed values and results. Our main goal is avoiding the problems with memory allocation and the insufficient number of variables.

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:

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

Greville in [3] proposed an ingenious recursive algorithm which relates the Moore-Penrose pseudo-inverse of a matrix R augmented by an appropriate vector r with the pseudo-inverse $R^{\dagger}$ of R. The algorithm is estimated as relatively complicated and numerically stable. We investigate implementation of the Greville's *partitioning method* for numerical computation of generalized inverses [3]. In a recursive implementation of the *partitioning method*, repetitive computations of some generalized inverse matrices is the major problem. This problem is investigated and resolved in [5]. We state an analogous statement applicable to the set of rational matrices.

Algorithm 1.1. *Consider $m \times n$ rational matrix $A(s)$ with parameter s. Let $\widehat{A}_i(s)$ be the submatrix of $A(s)$ consisting of the first i columns of $A(s)$. If the i-th column of $A(s)$ is denoted by $a_i(s)$, then $\widehat{A}_i(s)$ is partitioned as $\widehat{A}_i(s) = \left[\widehat{A}_{i-1}(s)|a_i(s)\right]$, $i = 2,\dots,n$, $\widehat{A}_1(s) = a_1(s)$.*

Received May 19, 2010.
2010 *Mathematics Subject Classification.* Primary 68N15; Secondary 97N60, 65F50
*Supported by Ministry of Education and Science, Republic of Serbia, Grant no. 174013

- *Step 1. Initial value:*

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

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

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

where the quantities $b_i(s)$ and $d_i(s)$ are defined in steps 2.1-2.3.

  - *Step 2.1.* $d_i(s) = \widehat{A}_{i-1}(s)^\dagger a_i(s)$,
  - *Step 2.2.* $c_i(s) = a_i(s) - \widehat{A}_{i-1}(s) d_i(s) = \left(I - \widehat{A}_{i-1}(s)\widehat{A}_{i-1}(s)^\dagger\right) a_i(s)$,
  - *Step 2.3.* $b_i(s) = \begin{cases} \frac{1}{c_i(s)^* c_i(s)} c_i(s), & c_i(s) \neq 0 \\ \frac{1}{1 + d_i(s)^* d_i(s)} (\widehat{A}_{i-1}(s)^\dagger)^* d_i(s), & c_i(s) = 0. \end{cases}$

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

Our main idea is to compare the implementation of the Algorithm 1.1 on different software environments. The paper is organized as follows. In the second section we describe implementation of the Algorithm 1.1 in symbolic package MATHEMATICA. We consider two implementations of the partitioning method. The first approach uses a possibility of the package MATHEMATICA to define functions which remember all found values. The second implementation is adjusted for sparse matrices. Also, a matrix algebra applicable to sparse matrices introduced in [7] is used for this implementations. In the third section our implementation is based on procedures and functions in the procedural programming language DELPHI [4]. Few illustrative examples are presented in the last section.

2. Implementation of partitioning method in MATHEMATICA

Beside the simplification, significant problem in a recursive implementation of Algorithm 1.1 is the increase of the number 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 pseudo-inverse $\widehat{A}_{i-1}(s)^\dagger$ is needful during the computation of the values $d_i(s)$ and $b_i(s)$. Consequently, Algorithm 1.1 requires $3(n - i + 1)$ recomputations of the Moore-Penrose inverse $\widehat{A}_i(s)^\dagger$, for

each $i \in \{2,\dots,n\}$. 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 [8].

At the beginning of our implementation in programming package MATHEMATICA we describe two auxiliary procedures.

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

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

B. The submatrix $A_j(s) = \left[a_1(s), \cdots a_j(s)\right]$ which contains the first $j \le n$ 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)]]]]
```

Step 2 of the Algorithm 1.1 is implemented in the following functions which remember before 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]:= Module[{m,n},{m,n}=Dimensions[a];A[a,n]//MatrixForm]
```

2.1. Implementation for sparse matrices

Possibly the most intuitive sparse matrix storage format is the Coordinate Format or *COO*. Instead of storing the matrix densely, a list of the coordinates containing row and column numbers is stored, along with the list of associated nonzero values. *COO* requires no specific matrix structure, so it is a very flexible format. It requires three (unordered) arrays and a single scalar recording the total *number of nonzero elements*, denoted with *nnz*. The combination of these three arrays provides a row i and column j coordinate pair of the entry on the matrix along with its value a_{ij} . In general, for a matrix with *nnz* non-zero entries, *COO* requires three 1-dimensional arrays of length *nnz* plus a scalar. By default, some implementation methods create matrices in *COO*. Using internal utilities subroutines, users can easily transform from COO to the other storage formats.

When we deal with *sparse matrices*, computer algebra comes close to the methods of numerical calculation [2]. For this purpose in paper [7] are written functions for implementation of basic operations in matrix algebra, which are applicable to sparse matrix representation.

The function *Sparse[mat_]* uses the formal parameter *mat_*, representing a matrix in the usual ("dense") form, required in MATHEMATICA [8].

The local variable l denotes an adequate *sparse representation* of the matrix *mat*:

$$l = \{\{i_1, j_1, mat_{i_1,j_1}\}, \{i_2, j_2, mat_{i_2,j_2}\}, \ldots, \{i_u, j_u, mat_{i_u,j_u}\}\}.$$

Assuming the above described representation of sparse matrices and described implementation of the partitioning method, implementation of the partitioning method applicable to sparse matrices can be written as follows.

```
DDS[a_List,i_]:=DDS[a,i]= Block[{s={}},
 s=MultMat[AS[a,i-1],ColS[a,i]]; If[Length[s]==0,Return[0],Return[s]]];

CCS[a_List,i_]:=CCS[a,i]= If[Length[DDS[a,i]]==0,
```

```
  Simplify[SubMat[ColS[a,i],MultSk[TakeFirstS[a,i-1],DDS[a,i]]]],
  Simplify[SubMat[ColS[a,i],MultMat[TakeFirstS[a,i-1],DDS[a,i]]]]];

BS[a_List,i_]:=BS[a,i]= Block[{b=a},
 If[Length[CCS[a,i]]==0,  If[Length[DDS[a,i]]==0,
    MultSk[AS[a,i-1],DDS[a,i]/(1+DDS[a,i])],
    MultSk[MultMat[HermitS[AS[a,2]],DDS[a,3]],
    1/(1+MultMat[HermitS[DDS[a,3]],DDS[a,3]][[1,1]])]  ],
  b=Rcpv[MultMat[HermitS[CCS[a,i]],CCS[a,i]]];
  If[Length[b]==0,MultSk[CCS[a,i],b[[1,3]]],MultMat[CCS[a,i],b]] ]];

AS[a_List,i_]:=AS[a,i]= Block[{b=a,h={},h1={},j,mx},
 If[i==1,MultMat[Rcpv[MultMat[HermitS[ColS[a,1]],ColS[a,1]]],
    HermitS[ColS[a,1]]],  If[Length[DDS[a,i]]==0,
       b=SubMat[AS[a,i-1],MultSk[HermitS[BS[a,i]],DDS[a,i]]],
       b=SubMat[AS[a,i-1],MultMat[DDS[a,i],HermitS[BS[a,i]]]] ];
    For[j=1,j<=Length[b],j++,If[b[[j,3]]=!=0,h=Append[h,b[[j]]]]];
    b=h; mx=MaxDim[b,1]; h1=HermitS[BS[a,i]];
    For[j=1,j<=Length[h1],j++,h=Append[h,{mx+1,h1[[j,2]],h1[[j,3]]}]];
    b=Union[b,h]] ]

PartitioningS[a_List]:= Module[{m,n,X}, X=Sparse[a][[3]];n=MaxDim[X,2];AS[X,n]]
```

3. Implementation of partitioning method in programming package 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 the partitioning method, a major problem is repetitive computation of generalized inverses of some matrices. We describe corresponding algorithms to eliminate these difficulties [5, 6].

In view of *Step 2* of Algorithm 1.1, for each $i \in \{2, \dots, n\}$, 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.

Let us define a type of date as follow:

```
type Matrix = record
                m,n:integer;
                Element:array[1..100,1..100] of real;
              end;
```

It is not difficult to generate a few auxiliary procedures for elementary matrix transformation:

- the function $ithCol(i : integer; A : Matrix) : Matrix$ returns the i-th column a_i of A;

- the function $FirstiCol(i : integer; A : Matrix) : Matrix$ generates the submatrix $A_j = \left[a_1, \cdots a_j\right]$ consisting of the first $j \le n$ columns of the matrix $A = A_n = [a_1, \cdots a_n]$;

- depending of if the matrix A is the zero matrix or not, the function $isZero(A : Matrix) : boolean$, having the formal parameter A, returns True or False value;

- the function $MatNum(A : Matrix) : Real$ returns the value of a 1×1 matrix' single element; otherwise reports the error;

- the function $AppRow(X, Y : Matrix) : Matrix$ appends the row-matrix Y to the rows of the matrix X, and returns the resulting matrix with $m + 1$ rows;

- the function $ZeroMat(m, n : integer) : Matrix$ returns $m \times n$ zero matrix;

- the function $SubsMat(X, Y : Matrix) : Matrix$ gives the matrix which is equal to the substraction of the matrices X and Y;

- the function $MultSk(k : Real; A : Matrix) : Matrix$ gives the matrix which is equal to the product of the matrix A with the scalar k;

- transpose of a given matrix A is implemented by function $Trans(A : Matrix) : Matrix$;

- the result of the function $MultMat(X : Matrix; Y : Matrix) : Matrix$ is the product of two matrices X and Y;

- Procedure $ReadFile(varA : Matrix; ime : String)$ read elements from file and puts them into the matrix A. Denote that this procedure can be very useful for test matrix. Therefore, we introduce the whole code.

```
procedure TForm1.ReadFile(var A:Matrix;ime:String); var
fajl:TextFile;i,j:integer; begin
 AssignFile(fajl,ime); Reset(fajl); Read(fajl,A.m); ReadLn(fajl,A.n);
 for i:=1 to A.m do begin for j:=1 to A.n do Read(fajl,A.Element[i,j]); ReadLn(fajl); end;
 CloseFile(fajl);
end;
```

Finally, the Algorithm 1.1 is implemented with function *Grevile*() as follows:

```
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 begin AR:=Trans(AA) end
    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)); WriteTable(i+1,DD);
     {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 2}
     AR:=SubsMat(AR,MultMat(DD,Trans(BB)));AR:=AppRow(AR,Trans(BB)); end;
 Grevile:=AR;
end;
```

4. Examples

Example 4.1. One of applications of the function *Partitioning[]* is given in the following example.

In[1]:= Partitioning[1+w,w,1+w,w,-1+w,w,1+w,w,1+w]

Out[1]= $\left\{\left\{\frac{1-w}{4}, \frac{w}{2}, \frac{1-w}{4}\right\}, \left\{\frac{w}{2}, -1-w, \frac{w}{2}\right\}, \left\{\frac{1-w}{4}, \frac{w}{2}, \frac{1-w}{4}\right\}\right\}$

By means of the function *PartitioningS[ ]* we generate the following sparse representation:

In[2]:=PartitioningS[{{1,1,1+w},{1,2,w},{1,3,1+w},{2,1,w},{2,2,w-1},{2,3,w},{3,1,1+w},{3,2,w},{3,3,1+w}]

Out[2]=$\left\{\left\{1,1,\frac{1-w}{4}\right\}, \left\{1,2,\frac{w}{2}\right\}, \left\{1,3,\frac{1-w}{4}\right\}, \left\{2,1,\frac{w}{2}\right\}, \{2,2,-1-w\}, \left\{2,3,\frac{w}{2}\right\}, \left\{3,1,\frac{1-w}{4}\right\}, \left\{3,2,\frac{w}{2}\right\}, \left\{3,3,\frac{1-w}{4}\right\}\right\}$.

Example 4.2. Consider the following matrix from [9], page 156:

$$X = \begin{bmatrix} 7 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 0 & 6 & -1 & -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 5 & -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 4 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 3 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 2 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Its sparse representation is given by

X={{1,1,7},{1,2,-1},{1,3,-1},{1,4,-1},{1,5,-1}, {1,6,-1}, {1,7,-1},{1,8,-1},

{2,2,6},{2,3,-1}, {2,4,-1},{2,5,-1},{2,6,-1},{2,7,-1},{2,8,-1},

{3,3,5},{3,4,-1},{3,5,-1},{3,6,-1},{3,7,-1}, {3,8,-1}, {4,4,4},{4,5,-1},{4,6,-1},{4,7,-1}, {4,8,-1},

{5,5,3},{5,6,-1},{5,7,-1},{5,8,-1}, {6,6,2},{6,7,-1},{6,8,-1}, {7,7,1},{7,8,-1}}

An application of the functions *PartitioningS*[] gives us the following result:

In[3]:=PartitioningS[X]

Out[3]=

$\left\{\left\{1,1,\frac{1}{8}\right\}, \left\{2,1,-\frac{1}{56}\right\}, \left\{2,2,\frac{1}{7}\right\}, \left\{3,1,-\frac{1}{56}\right\}, \left\{3,2,-\frac{1}{42}\right\}, \left\{3,3,\frac{1}{6}\right\},\right.$

$\left\{4,1,-\frac{1}{56}\right\}, \left\{4,2,-\frac{1}{42}\right\}, \left\{4,3,-\frac{1}{30}\right\}, \left\{4,4,\frac{1}{5}\right\}, \left\{5,1,-\frac{1}{56}\right\}, \left\{5,2,-\frac{1}{42}\right\}, \left\{5,3,-\frac{1}{30}\right\}, \left\{5,4,-\frac{1}{20}\right\}, \left\{5,5,\frac{1}{4}\right\},$

$\left\{6,1,-\frac{1}{56}\right\}, \left\{6,2,-\frac{1}{42}\right\}, \left\{6,3,-\frac{1}{30}\right\}, \left\{6,4,-\frac{1}{20}\right\}, \left\{6,5,-\frac{1}{12}\right\}, \left\{6,6,\frac{1}{3}\right\},$

$\left\{7,1,-\frac{1}{56}\right\}, \left\{7,2,-\frac{1}{42}\right\}, \left\{7,3,-\frac{1}{30}\right\}, \left\{7,4,-\frac{1}{20}\right\}, \left\{7,5,-\frac{1}{12}\right\}, \left\{7,6,-\frac{1}{6}\right\}, \left\{7,7,\frac{1}{2}\right\},$

$\left.\left\{8,1,-\frac{1}{56}\right\}, \left\{8,2,-\frac{1}{42}\right\}, \left\{8,3,-\frac{1}{30}\right\}, \left\{8,4,-\frac{1}{20}\right\}, \left\{8,5,-\frac{1}{12}\right\}, \left\{8,6,-\frac{1}{6}\right\}, \left\{8,7,-\frac{1}{2}\right\}\right\}$

Example 4.3. Consider the following data from a text file of the form

```
10  11

 1   2   3   4   1   1   3   4   6   2
 1   3   4   6   2   2   3   4   5   3
 2   3   4   5   3   3   4   5   6   4
 3   4   5   6   4   4   5   6   7   6
 4   5   6   7   6   6   6   7   7   8
 6   6   7   7   8   1   2   3   4   1
 3   4   1   1   3   4   6   2   1   2
 4   6   2   2   3   4   5   3   1   3
 4   5   3   3   4   5   6   4   2   3
 5   6   4   4   5   6   7   6   3   4
 6   7   6   6   6   7   7   8   4   5
```

Matrix A of the size 10×11 is derived. By applying Algorithm 1.1 and function *Grevile*() we get the following Moore-Penrose inverse

$$A^{\dagger} = \begin{bmatrix}
0.294371 & -0.168563 & -1.51113 & 1.41468 & -0.390817 \\
-0.0665814 & -0.0736365 & 1.22658 & -1.06364 & 0.229515 \\
0.226663 & -0.178769 & -0.692492 & 0.705403 & -0.213752 \\
-0.164739 & 0.547244 & -0.656536 & 0.375969 & -0.115828 \\
-0.296725 & -0.0104986 & 1.03479 & -0.972437 & 0.359073 \\
-0.00818833 & -0.473753 & 1.66304 & -1.31891 & 0.352318 \\
0.0619243 & 0.368475 & -1.34903 & 1.08137 & -0.32958 \\
-0.0806039 & 0.157918 & 0.0289882 & -0.143698 & -0.0341059 \\
0.195087 & -0.484252 & 1.19782 & -0.791344 & 0.211391 \\
-0.0806039 & 0.157918 & -0.971012 & 0.856302 & -0.0341059
\end{bmatrix}$$

$$\begin{bmatrix}
0.0414486 & -0.0402081 & -0.259697 & 0.454236 & -0.264066 & 0.230377 \\
0.000526393 & -0.192068 & 0.649101 & -0.103239 & -0.190912 & -0.102084 \\
-0.00878296 & -0.253651 & -0.237774 & 1.51422 & -1.31922 & 0.448655 \\
0.0147536 & 0.179273 & -0.196037 & -0.456054 & 0.531094 & -0.104233 \\
0.107833 & 0.237939 & 0.0435639 & -1.06532 & 0.937535 & -0.365726 \\
-0.102915 & -0.428182 & 0.224424 & 1.82997 & -1.84384 & 0.414315 \\
0.00597066 & 0.425623 & -0.433811 & -0.441829 & 0.711873 & -0.155578 \\
-0.0212507 & 0.0871716 & -0.0192523 & -1.49888 & 1.44795 & -0.138105 \\
0.00491787 & -0.190242 & 0.267988 & 0.764649 & -0.906302 & 0.048589 \\
-0.0212507 & 0.0871716 & -0.0192523 & -0.498879 & 0.447946 & -0.138105
\end{bmatrix}.$$

5. Conclusions

We investigate the implementation of Greville's recursive algorithm in different programming packages. The problem caused by recomputations of identical quantities is also examined. This problem is solved using the possibility of the programming package MATHEMATICA and the language DELPHI to define functions which remember values that they find. These methods can be used as usual tools in all similar problems no matter what programming language is used for implementation.

Acknowledgement

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* Wiley-Interscience, New York, 1974.
2. J.H. Davenport, Y. Siret and E. Tournier *Computer Algebra*, Academic Press, (1988)
3. T.N.E. Grevile, *Some applications of the pseudo-inverse of matrix* SIAM Rev. **3** (1960), 15–22.
4. K. Henderson, DELPHI 3 *Client-Server Developer's Guide* Borland PRESS, (1997)
5. P.S. Stanimirović and M.B. Tasić, *Partitioning method for rational and polynomial matrices*, Appl. Math. Comput., **155** (2004) 137–163.
6. P.S. Stanimirović and M.B. Tasić, *Computing determinal representation of generalized inverses*, Korean J. Comput & Appl. Math. **9** (2002), 349–359.
7. I.P. Stanimirović and M.B. Tasić, *Performance comparison of storage formats for sparse matrices*, Facta Universitatis, Ser. Math. Inform. **24** (2009), 39-51.
8. S. Wolfram, *The Mathematica Book, 5th ed.*, Champaign: Wolfram Media, Inc., 2004.
9. G. Zielke, *Report on test matrices for generalized inverses* Computing **36** (1986), 105–162

Milan B. Tasić
Faculty of Sciences and Mathematics,
Department of Mathematics and Informatics,
P. O. Box 224, Višegradska 33,
18000 Niš, Serbia
milan12t@ptt.rs

Ivan P. Stanimirović
Faculty of Sciences and Mathematics,
Department of Mathematics and Informatics,
P. O. Box 224, Višegradska 33,
18000 Niš, Serbia
ivan.stanimirovic@gmail.com