CAS-supported Multiple Representations in Elementary Linear Algebra

The Case of the Gaussian Algorithm

Wolfgang Lindner, Duisburg (Germany)

Abstract: Usually the Gaussian algorithm (GA) is presented at school as a method of solving a given system of linear equations by reducing it to a "triangular form". In contrast to this technically oriented view, I will demonstrate a CAS-supported learning environment which includes a visual representation of GA and an activity-oriented „Gauss-game". This game stresses the concept of elementary matrices and leads directly to a partial implementation of GA in the form of a „semi-automatic" functional CAS-program. These multiple representation of GA tries to take into consideration the research results on mental representations, to design rich variations of student activities and thereby to lead leading to webbeb concepts around GA. The CAS MuPAD is used.

Kurzreferat: Üblicherweise wird der Gauß-Algorithmus (GA) in der Schule als eine Methode präsentiert, ein gegebenes lineares Gleichungssystem durch Reduktion auf Dreiecksgestalt zu lösen. Im Gegensatz zu dieser technisch orientierten Sicht wird hier ein CAS-gestütztes Lernarrangement skizziert, welches eine visuelle Repräsentation des GA und ein handlungsorientiertes „Gauss-Spiel" einschließt. Dieses Spiel basiert auf dem Konzept der Elementarmatrizen und führt unmittelbar zu einer partiellen Implementation des GA in Form eines „semi-automatischen" funktionalen CAS-Programms. Die multiplen Repräsentationen des GA versuchen Forschungsbefunde über mentale Repräsentationen aufzugreifen, um reichhaltige Aktivitäten der Lernenden zu ermöglichen und dadurch Vernetzungen im Umkreis des GA anzubahnen. Benutzt wird das CAS MuPAD.

ZDM-Classification: A30, A4

1. Introduction

Recent research in the theory of learning mathematics has stressed the importance of multiple representations for the learning process. Consequently, recommendations in the US NCTM-2000 Standards mention corresponding didactic rules in teaching mathematics, e.g. the *Rule of Four*: (re)present every mathematical topic *numerically*, *graphically*, *algebraically* (analytically) and *descriptively*.

This rule aims at the learner´s *cognitive flexibility,* with is an important requirement for mastering complex and open situations. Often this flexibility can be regarded as the capability to switch fluently between different mental representations of an object. The construction of flexible knowledge is supported by using multiple forms of representation of central mathematical concepts in the learning process.

In this paper I will use the Computer Algebra System (CAS) *MuPAD* as a technical medium for the construction of multiple representations in learning arrangements for elementary Linear Algebra, especially for the teaching of Gauss-Jordan algorithm. The reader who is familiar with the symbolic CAS *Maple* will immediately feel at home, because both syntax and semantics of MuPAD are comparable with Maple. There exists a fully functional free version MuPAD *Light*, so the interested reader can test the code and the system.

The CAS-environments itself are designed as moderate constructivistic learning situations; experimental problem contexts, confrontation with learning obstacles to invoke mistakes, misconceptions, conflicts or surprising outcomes are essential design components. Usually the problems and exercises were presented in the form of activity-based problem-oriented notebooks or worksheets, sometimes as small lectures with open-ended mini-projects presented as *hyperlinked* MuPAD specific DVI-files which were then included and invoked as part of the MuPAD help system.

2. Standard computational representation of GAUSS-algorithm

Looking at current German textbooks on elementary Linear Algebra at High School one will recognize that most books start with the solution of systems of linear equations using Gaussian elimination and are heavily involved in numerical calculations and algebraic operations (e.g. adding a multiple of one equation to another). As a result, the topic - despite some word problems -remains very formal and thus contradics the Rule of the Four. Here is a typical example formulated in MuPAD's symbolic language in the form of a notebook instead of on paper:

- `// ..solving LGS by hand in worksheet`

```
A := matrix([[3, 6,  6],  // 3x+6y =   6
             [2, 3,3.5]]);// 2x+3y = 3.5

A1:= matrix([[1, 2,  2],  // 1x+2y =   2
             [2, 3,3.5]]);// 2x+3y = 3.5

A2:= matrix([[1, 2,   2],
             [0,-1,-0.5]]);

A3:= matrix([[1, 2,   2],
             [0, 1, 0.5]]);

A4:= matrix([[1, 0,   1],
             [0, 1, 0.5]]);

linalg::gaussJordan(A); //verified by CAS
      +-          -+
      |  1, 0, 1.0  |
      |  0, 1, 0.5  |
      +-          -+
```

The solution process for the linear system is presented simultaneously in matrix form and – at the right side of the MuPAD commentary sign "//" - in equation form. The last line invokes the built-in procedure `gaussJordan` of the `linalg`−package, which immediately solves the given system. In view of this productivity of the CAS it is no longer reasonable that students learn the process of elimination with paper and pencil in the form of the usual "algorithmic" pattern.

Redesigning the curriculum of elementary Linear Algebra in schools therefore leads to the following question: Should we cancel the Gaussian elimination or are there still some benefits? And how can we redesign the learning sequence? In order to find an answer to this question I propose to locally extend the Rule of the Four to the *Rule of the Five* by enriching the instruction process "*cas*"ually with CAS-procedures (algorithmic view). The surplus using CAS could be:

- to get insight by CAS visualizations, instead of only calculating numbers
- to tutor the learning process by giving intermediate results via the use of CAS
- to stimulate concept formation (i.e. the concept of the inverse matrix)
- to use elementary programming activities as a constructive substitution of mathematical proof techniques – both are intellectually demanding thought processes, which can spend emotional satisfaction.

3. Visual representations of GAUSS-algorithm

In the following chapters I will sketch a CAS learning arrangement for the Gaussian algorithm (GA). I presuppose that the students have a thorough understanding of the concept of matrices and the accompanying operations, especially of matrix multiplication as discussed in Strang (1998). After two well-known but different graphical representation I will propose a new one: the *matrix image of the GA solution process*. All three representations of the solution process of a system of linear equations were given nearly simultaneously to the students, e.g. in the same cooperative CAS learning arrangement thus acting as a "big" advanced organizer in the sense of Ausubel and connecting the different approaches from the very beginning.

3.1 Representation using the Row Image

Taking for example the simple linear system of §2 given by the augmented system matrix A we let the students watch and discuss the graphical illustration of the GA-solution process A, A2, A3, A4:

```
• read("RowImage2d"): // user procedure
  RowImage2d(A4)      // final state:
```

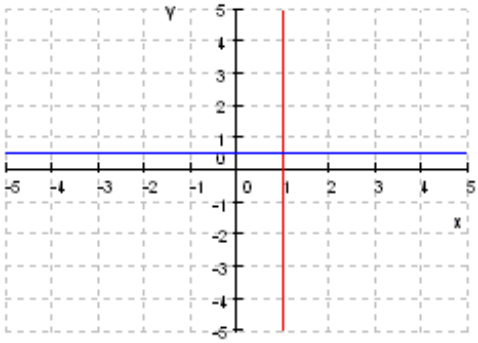

Fig. 1: *Row* image of the solution process for a 2 by 2 system.

In this *RowImage* each row of matrix A is represented visually as a line. The solution is the point (1;0.5) where the lines meet. The coordinates of the solution point remain invariant throughout the solution process. GA adjusts the lines into a standard position parallel to the axes where the solution is read off "best".

The corresponding MuPAD code for the user-written procedure `RowImage2d` may be written by the students as an exercise and reflects the underlying mathematical thoughts; the vertical line is produced as a polygon between two points. The last line writes the code into a file (frozen in machine readable form, which hides the coding as a *Black Box*) named `"RowImage2d"`:

```
• RowImage2d := proc(A:matrix,
                     x1=-5,x2=5,y1=-5,y2=5)
  begin
  plotfunc2d(
   if A[2,1]=0
      then A[2,3]/A[2,2]
       else (A[2,3]-A[2,1]*x)/A[2,2]
      end_if,
   if A[1,2]=0
      then
       [Mode=List,
        [polygon(point(A[1,3]/A[1,1],y1),
                 point(A[1,3]/A[1,1],y2))]]
      else (A[1,3]-A[1,1]*x)/A[1,2]
    end_if,
   x=x1..x2, ViewingBox=[x1,x2,y1,y2],
   Ticks=[abs(x1)+abs(x2)+1,
                  abs(x1)+abs(x2)+1],
   GridLines=Automatic)
  end_proc:
  write("RowImage2d", RowImage2d);
```

This MuPAD procedure `RowImage2d` was then used as an exploration aid in the problem sets.

3.2 Representation using the Column Image

We will now look at the same linear system A from another window by changing the representation.

In the *ColumnImage* of the solution process each column of system matrix A is represented visually as a (column) vector starting at the origin. The Gaussian algorithm transforms the column vectors of A into a standard position ("basis") where the solution of the linear system is immediately read:

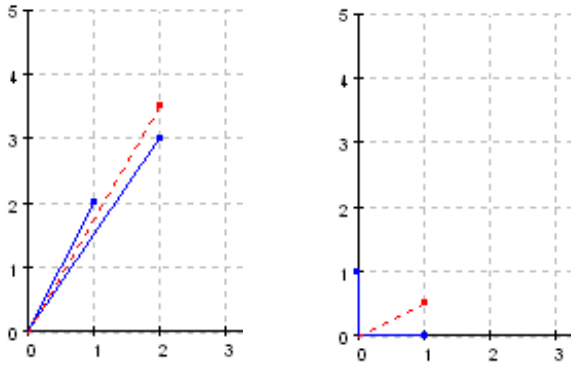

Fig. 2: *Column* Image of Gauss algorithm for the linear system.

The start configuration is illustrated in Fig. 2, left; Fig. 2 left, for example, is produced via the following call:

```
plot(ColumnImage(A4),
     ViewingBox=[0,7,-2,5],
     Ticks=[8,8],GridLines=Automatic)
```

Remark: The implementation of the procedure `ColumnImage` is rather technical and therefore used as black box by the students.

3.3 Representation using the Matrix Image

Whereas the row picture and the column picture as visual representation of Gaussian algorithm are well known (Strang, 1998, p.22 ff) I suggest another view on the solution process of the same linear system A, changing the representation a third time: the *MatrixImage* of A visualizes the columns of A as a polygonal line joining these columns ("points") from the first one to the last with the last connection dashed. Here is, for example, the chain of *MatrixImage*`s, which represent the snapshots of the GA solution process A,.., A4 in §2:

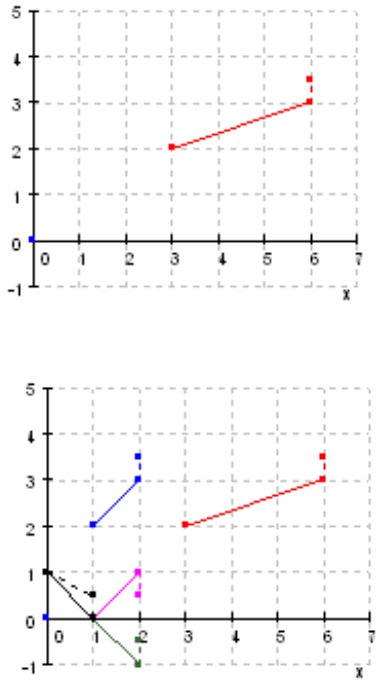

Fig. 4a,b: Chain of Matrix Images of Gauss algorithm for the 2 by 2 linear System A; starting position and whole scene

In the *MatrixImage* view of the Gaussian algorithm the linear system is again transformed from a start configuration (Fig. 4a) into a standard position ("canonical basis path": from [1;0] via [0;1] to solution [1;0.5]; see Fig. 4b) where the solution of the linear system is immediately read from the endpoint. A surplus is gained by regarding the transformation steps as linear mappings acting on the foregoing matrix ("polygonal figure"). The students can identify and explore shears, stretchings, reflections in the x-axis etc, see Banchoff (1983).

Remark: The implementation of the procedure `MatrixImage` is again technical and therefore used as black box by the students.

4. Intermezzo: Playing with Elementary Matrices

In describing the solution process of a linear system of equations we are naturally led to the consideration of elementary matrices, which en passant allows for the construction of the inverse of a regular matrix and at the same time evokes an idea of implementing a functional version of the Gauss algorithm.

4.1 Playing with Elementary Matrices

Using the following *Elementary-Matrix-Game* the students get a *CAS-activity based informal concept* of elementary matrices and their properties.

Elementary-Matrix-Game:

Play ElementaryMatrix with your partner 10 times in turn. Each player writes down an elementary matrix e.g. `EM(2,1,-2,3)` as input at the MuPAD prompt, both then write their prediction for the MuPAD output matrix on paper.

- The player with the most correct predictions wins.
- OK, you can also play ElementaryMatrix EM as solitaire at home.

```
EM := (i,j,k,d,M) ->
        (M:=matrix(d,d,1,Diagonal);
         M[i,j]:=k;
         return(M)  ):
```

```
matrix(2,3, -1,Diagonal)
```

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}$$

```
EM(2,1,-2,3);  // <- change input here
```

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The meaning of the input parameters of EM should become clear without any problems.

4.2 Playing with actions of Elementary Matrices from the left

To let the students explore the effects of EM-matrices acting from the left on a given matrix (linear system) A, I designed the following cooperative

Effect-on-Matrix-Game: Play with your partner 10 times Effect-on-Matrix. Each player alters in turn one of the first three variables in `EM(1,1,1,2)*A`, e.g. in `EM(2,1,-2,3)` and both write their prediction for the MuPAD output on paper.

- The player with the most correct predictions wins.
- You can play on level 2: here you must alter 2 of the first three inputs in `EM(1,1,1,2)`.
- You can use `MatrixImage` or another visualization to consolidate your insight.
- Can you think of a level 3?

```
A := matrix([[3,6,6], [2,3,3.5]]);
EM(1,1,2,2)*A;
```

```
+-           -+      +-           -+
|  3, 6,  6   |      |  6, 12,  12  |
|             |      |              |
|  2, 3, 3.5  |      |  2,  3, 3.5  |
+-           -+      +-           -+
```

4.3 Playing GAUSS-JORDAN

Now I will show a CAS based learning arrangement in the form of a game, which leads to a constructive concept of the inverse of a matrix. At the same time the GA algorithm becomes clear and an idea of using elementary matrices for explicitly implementing GA arises.

GAUSS-JORDAN-Game: Beginning from the right, the players take turns altering exactly one EM-factor in the following equation #3. At the same time the corresponding matrices in line #1 and #2 are altered. The MuPAD output is watched. The game is over if one player reaches the final output

```
+-           -+
|  1, 0, 1.0  |
|             |
|  0, 1, 0.5  |
+-           -+
```

and the solution of the well-known linear system A emerges.

Before starting the game, we fix on 2-dimensional EM-matrices for easy playing:

```
• EM := (i,j,k,d,M) ->
       (d:=2;  // fixing on 2x2-Mat's
        M:= matrix(d,d,1,Diagonal);
        M[i,j]:=k;     return(M)  ):

  A := matrix([[3,6,6], [2,3,3.5]]);
```

We now demonstrate a first possible move; #1# gives the sequence, #2# the product and #3# the changing A:

```
• EM(1,1,1),EM(1,1,1),EM(1,1,1/3);      #1#
  EM(1,1,1)*EM(1,1,1)*EM(1,1,1/3);      #2#
  EM(1,1,1)*EM(1,1,1)*EM(1,1,1/3)*A;    #3#
  //     (3)        (2)          (1)
```

- Repeat the game and alter the matrix A of the given linear system.
- You can play on level 2: delete rows #1 and #3 playing only on line #3
- You can use `MatrixImage` or another visualization to consolidate your insight.
- This is the demanding level 3: can you go back from the solution position to the start matrix A?

4.4 Playing and exploring further – getting an idea of Invertibility

The CAS-learning arrangement in 4.3 demonstrated the equivalence of the GA-transforming steps by using the invertability of the elementary matrices and thus transforming the "solution matrix" back into the given linear system A. All concepts were stabilized by means of visual multiple representations and action-based cooperative CAS-games.

Compacting all EM-factors of the solution process in a single resulting matrix G (resp. G_ for reversing the process) the students construct the mathematical concept of the inverse matrix in an informal way. The following MuPAD notebook provides an outline:

```
• // Solution of last GAUSS-JORDAN-game:
  EM(1,2,-2,2)*EM(2,2,-1,2)*EM(2,1,-2,2)
                          *EM(1,1,1/3,2)*A;
```

```
+-           -+
|  1, 0, 1.0  |
|             |
|  0, 1, 0.5  |
+-           -+
```

```
• G:=EM(1,2,-2,2)*EM(2,2,-1,2)*
     EM(2,1,-2,2)*EM(1,1,1/3,2)
```

```
+-          -+
|   -1,   2  |
|            |
|  2/3,  -1  |
+-          -+
```

```
• G*A   // Test
```

```
+-           -+
|  1, 0, 1.0  |
|             |
|  0, 1, 0.5  |
+-           -+
```

Defining the coefficient Matrix A_ of the system matrix A, the students can see that G equals the built-in inverse of A_:

```
• A_ := matrix([[3,6], [2,3]]);
• A_^-1
```

```
+-          -+
|   -1,   2  |
|            |
|  2/3,  -1  |
+-          -+
```

```
• A_^-1*A;
• linalg::gaussJordan(A); // same results
```

```
+-           -+     +-           -+
|  1, 0, 1.0  |     |  1, 0, 1.0  |
|             |     |             |
|  0, 1, 0.5  |     |  0, 1, 0.5  |
+-           -+     +-           -+
```

To sum up, up to this point the students have explored the properties and effects of EM-matrices for the solution process of systems of linear equations. The constructive concept of the inverse A^{-1} of a given matrix A as product of elementary matrices has evolved and is consolidated. Now we are reaching a point where we can generate hypotheses about the process of inverting, arguing rules and proving theorems about $(..)^{-1}$. After an experimental CAS supported phase in the learning process we are now entering the *exactification phase* in the sense of Buchberger's creativity helix (Heugl 1996, p. 82 ff). In reconstructing elementary Linear Algebra from a CAS point of view it is now possible to forget about the technical solution process for solving systems of linear equations and simply use the built-in Swiss army knifes like $(..)^{-1}$ or `gaussJordan` to further study the theory of linear equations.

Alternatively it is possible to use the insight in the construction process of A^{-1} resp. the solution process for an linear system to program a simple tutorial version of

Gaussian algorithm. This will be demonstrated next.

4.5 Motivating a program for GAUSS-JORDAN algorithm in MuPAD

In the next paragraph we will consider the following 3 by 3 system of linear equations given in form of the augmented system matrix B. To motivate the students for the algorithm we let them solve the system B using EM-matrices:

```
• B:=matrix([[ 2, 4,-2, 2], //2x+4y-2z = 2
             [ 4, 9,-3, 8], //4x+9y-3z = 8
             [-2,-3, 7,10]])//-2x-3y+7z=10
• EM(2,1,-2,3)*B
• EM(3,1,1,3)*EM(2,1,-2,3)*B
• EM(3,2,-1,3)*EM(3,1,1,3)*EM(2,1,-2,3)*B
• EM(3,3,1/4,3)*EM(3,2,-1,3)
                *EM(3,1,1,3)*EM(2,1,-2,3)*B
• EM(1,1,1/2,3)*EM(3,3,1/4,3)*EM(3,2,-1,3)
                *EM(3,1,1,3)*EM(2,1,-2,3)*B
```

We let them control the result using B^{-1} or `linalg::gaussJordan`. Then they had to read in the following unknown procedure `rref`, which also gives the result but shows *all relevant intermediate steps* of *the solution process* as a surprise:

```
• read("rref"):
  rref(B);
```

```
+-              -+  --  +-                  -+
|  1, 0, 0, -1   |  |   |   2,  4, -2,  2   |
|                |  |   |                   |
|  0, 1, 0,  2   |, |   |   4,  9, -3,  8   |,
|                |  |   |                   |
|  0, 0, 1,  2   |  |   |  -2, -3,  7, 10   |
+-              -+  --  +-                  -+

   +-              -+   +-              -+
   |  2, 4, -2,  2  |   |  2, 4, -2, 2  |
   |                |   |               |
   |  0, 1,  1,  4  |,  |  0, 1,  1, 4  |,
   |                |   |               |
   |  0, 1,  5, 12  |   |  0, 0,  4, 8  |
   +-              -+   +-              -+  ..
```

This rises the question for the curious ones: how is `rref` constructed? Can we remake it? The following topic was therefore presented to some of my students in the form of a bonus mini-project. It demonstrates another didactic surplus in a productive CAS learning arrangement.

5. A functional teaching-code version of the Gauss-Jordan algorithm in MuPAD

In their preface Kernighan (1999, S.) list some important programming principles: "These include *simplicity*, which keeps programs short and manageable; *clarity*, which makes sure they are easy to understand[..]; *generality*, which means they work well in a broad range of situations[..]; and *automation*, which lets the machine do the work for us, freeing us from mundane tasks." In the following, to realize some of these principles, I will use the concept of semi-automatic algorithms, which was mentioned independently by Sarvari (2001) and Lindner (2001).

The next CAS mini-project shows a functional program for the Gauss-Jordan algorithm (GJ). The code is teaching-code and uses insights concerning elementary matrices. The procedures will extend the professionally implemented built-in procedure with tutorial elements by showing intermediate results.

The plan is to program each phase of GJ as a stand-alone MuPAD function. This way we will try to maintain simplicity and clarity, being able to test each part of the whole algorithm alone. At the end we will paste the working parts together *using the mathematical concept of composition*. This strategy is called *functional programming*.

5.2.1 Going down for Elimination

Knowing about the usefulness of elementary matrices for the elimination process (4.2), let us go down for elimination building products of EM's:

```
• export(linalg, nrows, ncols, col):

• Gdown:=(v,k)->
        _mult(EM(i,k,-v[i]/v[k], nrows(v))
                        $i=k+1..nrows(v)):

  Elimination := A ->
     ((A:=Gdown(col(A,i),i)*A)
          $ i=1.. ncols(A)-1;  return(A)):

  Elimination_:= A ->
     (A:=Gdown(col(A,i),i)*A)
          $ i=1.. ncols(A)-1:
```

Note the use of explicit `return(A)` in function `Elimination`, which therefore *only* gives back *the last value* of matrix A, whereas *omitting* `return(A)` *shows all intermediate results*:

```
• Elimination(B);
  +-              -+
  |  2, 4, -2,  2  |
  |                |
  |  0, 1,  1,  4  |
  |                |
  |  0, 1,  5, 12  |
  +-              -+
• Elimination(B);
   +-              -+   +-              -+
   |  2, 4, -2,  2  |   |  2, 4, -2, 2  |
   |                |   |               |
   |  0, 1,  1,  4  |,  |  0, 1,  1, 4  |,
   |                |   |               |
   |  0, 1,  5, 12  |   |  0, 0,  4, 8  |
  +-              -+   +-              -+  ...
```

Every part of the code is testable alone so the student can see the parts working. This is possible because MuPAD works as interpreter:

```
• v:=col(B,1);   Gdown(col(B,1),1);
```

```
   +-    -+              +-          -+
   |   2  |              |   1, 0, 0  |
   |   4  |              |  -2, 1, 0  |
   |  -2  |              |   1, 0, 1  |
   +-    -+              +-          -+
```

```
• EM(i,1,-v[i]/v[1],nrows(v))
                    $i=1+1..nrows(v);
```

$$\begin{pmatrix} 1, 0, 0 \\ -2, 1, 0 \\ 0, 0, 1 \end{pmatrix} \begin{pmatrix} 1, 0, 0 \\ 0, 1, 0 \\ 1, 0, 1 \end{pmatrix}$$

G(auss)*down* gets a column v of the coefficient matrix A and a position (e.g. k) in v and builds the product of all eliminating elementary matrices EM below this position k. Such a product with many factors is realized in MuPAD using `_mult(..)`. The function `Elimination` is now performing the process of repeated eliminations in all columns of A using this helper function `Gdown` for the elimination of a single column. In each step we get a changed A (e.g. A1, A2, ..), so that the next elimination step works on this changed intermediate result: therefore this result must be memorized with `A:=`..

It is important to mention that the two lines above for the procedures `Gdown` und `Elimination` are functionally equivalent to the code of `GaussElimin` in 5.1.

5.2.2 Going down for Normalization
This is a simple task:

```
Gnorm := A ->
         _mult(EM(i,i, 1/A[i,i],nrows(A))
                          $i=1..nrows(A)):
                  // o.k.- it's critical ..
Normalization := (A)-> Gnorm(A) * A:
```

Test suite:

```
Gnorm(Elimination(B));
```

$$\begin{pmatrix} 1/2, 0, 0 \\ 0, 1, 0 \\ 0, 0, 1/4 \end{pmatrix}$$

```
Normalization(Elimination(B));
```

$$\begin{pmatrix} 1, 2, -1, 1 \\ 0, 1, 1, 4 \\ 0, 0, 1, 2 \end{pmatrix}$$

5.2.3 Going up for (Back-) Substitution
We have learned to eliminate (going down), so we can also do back-substitution (going up):

```
Gup := (v,k) ->
       _mult(EM(i,k,-v[i]/v[k],nrows(v))
                                $i=1..k-1):
Substitution := A ->
  ((A:=Gup(col(A,i),i)*A) $i=1..nrows(A):
    return(A)):
Substitution_:= A ->
   (A:=Gup(col(A,i),i)*A) $i=1..nrows(A):
```

Test suite:

```
Substitution_( Normalization
                      (Elimination(B)) );
```

$$\begin{pmatrix} 1, 2, -1, 1 \\ 0, 1, 1, 4 \\ 0, 0, 1, 2 \end{pmatrix} \begin{pmatrix} 1, 0, -3, -7 \\ 0, 1, 1, 4 \\ 0, 0, 1, 2 \end{pmatrix} \begin{pmatrix} 1, 0, 0, -1 \\ 0, 1, 0, 2 \\ 0, 0, 1, 2 \end{pmatrix}$$

5.2.4 Composing one after another
This again is simple: we use MuPAD's functional composition operator `@` just as in mathematics and finally abstract the whole solution process in the function `rref1`:

```
rref1 := B -> (Substitution  @
               Normalization @
               Elimination     )(B);
```

Test suite:

```
rref1(B);
linalg::gaussJordan(B);
```

$$\begin{pmatrix} 1, 0, 0, -1 \\ 0, 1, 0, 2 \\ 0, 0, 1, 2 \end{pmatrix}$$

As a result, the black box `gaussJordan` alias `rref1` (=*r*ow *r*educed *e*chelon *f*orm) is brightened to a white box. This is not the whole story but only a first step to a deeper understanding. For school the above insight should be sufficient because afterwards we can use the professional built-in version `linalg::gaussJordan`.

6. Conclusions

This article has demonstrated a productive, cooperative, activity-based CAS-learning arrangement for the Gauss-Jordan algorithm for the solution of systems of linear equations. It focuses on the use of invertible elementary matrices in both mathematical and programming aspects. Important mathematical concepts (matrix, operations) and informatical concepts (data and control structure; user-defined vs. built-in functions) evolved in an informal manner thus avoiding well-known problems of conceptualization in elementary Linear Algebra. With simple (semi)automatic MuPAD-functions we focused on the crucial steps of the algorithm but did not get lost in administrative program-specific technical details. This way we tried to reduce the formal aspect of mathematics res. informatics to the necessary extend and to keep the considerations for our students simple and clear. Also we tried to substitute purely mathematical existence proofs (e.g. the construction of the inverse matrix) by algorithmic constructions and to support procedural thought processes.

It should be mentioned that the procedure `MatrixImage` could also be used as means for multiple representations in 3D:

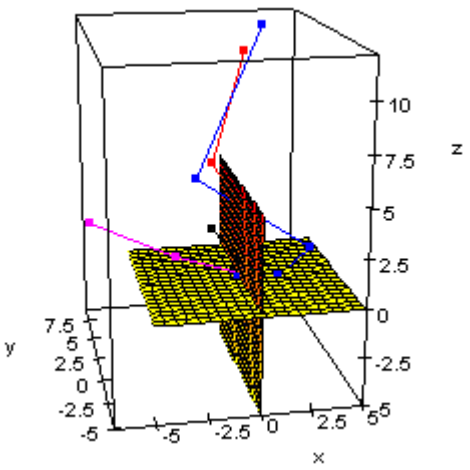

Fig. 5: 3D `MatrixImage` of 3 by 3 linear system B

Remark: It should be mentioned that the normal matrix output of MuPAD is pretty-printed. Because of printing reasons this was changed to ASCII output in the text above.

References

Banchoff, T.; Wermer, J. (1983): Linear Algebra Through Geometry. New York: Springer

Heugl, H.; Klinger, W.; Lechner, J. (1996): Mathematikunterricht mit Computeralgebrasystemen. Bonn: Addison-Wesley

Kernighan, B.W.; Pike, R. (1999): The Practice of Programming. Bonn: Addison-Wesley

Lindner, W. , (2001): Misconceptions around Matrix multiplication and their Correction in Dialogue with CAS. – In: R. Soro (Ed.), Current State of Research on Mathematical Beliefs X, Proceedings of the MAVI-10 European Workshop 2001. Turku: Pre-Print Series of University of Turku, No. 1, 41-46

Lindner, W. (2002): The Digraph-CAS-Environment and corresponding Elementary Programming Concepts. – In: M. Borovcnik, H. Kautschitsch (Eds), Technology in Mathematics Teaching, Proceedings of the ICTMT 5 in Klagenfurt 2001. – Wien: öbv&hpt (Schriftenreihe Didaktik der Mathematik v. 26), S. 199-202

Sarvari, C. (2001): Rolle der CAS in der Entwicklung des mathematischen Denkens. – In: G. Kaiser (Ed), Beiträge zum Mathematikunterricht 2001, Proceedings of the 35. GDM in Ludwigsburg. - Bad Salzdetfurth: Franzbecker, S. 528-531

Strang, G. (1998): Introduction to Linear Algebra. – Wellesley: Wellesley-Cambride Press

---

Autor

Lindner, Wolfgang, Fakultät 4 – Naturwissenschaften, Institut für Mathematik LE 424, Gerhard-Mercator-Universität Duisburg, Lotharstr. 65, D-47048 Duisburg.
E-mail: Lindner@math.uni-duisburg.de