Exploring Elements of Linear Algebra through Experiments with LOGO

Karl Josef Fuchs, *karl.fuchs@sbg.ac.at*
Department of Mathematics and Informatics Education, University of Salzburg

Hans-Stefan Siller, *hans-stefan.siller@sbg.ac.at*
Department of Mathematics and Informatics Education, University of Salzburg

Abstract

In terms of constructionism knowledge and skills are internalized activities. Teaching following this approach has turned out to be efficient at any stage of life. This paper discusses the design of a course for the introduction of Linear Algebra at secondary level combining Mathematics and Computer Science. The way to gain basic elements does not head to traditional definitions and subsequent exercises but turns the path almost upside down. Constructing, experimenting and exploring are at the beginning. The implementations of LOGO functions should open up and produce deeper insights into basic elements of Linear Algebra by the students.

Keywords

Mathematics, Computer Science, Linear Algebra, Constructing - Experimenting - Exploring, LOGO Functions, Transfer, Analogy, Inner Differentation

Prelude to the Constructionistic Approach

If you are skimming over the titles of our paper you may receive the impression of huge inconsistancy. Apparently we want something from Elements of Linear Algebra like **vectors** and **matrices** which are largely regarded as extremly abstract objects. Otherwise we address exploring and experimenting as appropriate strategies to discover these abstract basic ideas. With the following concept we want to demonstrate that constructionistic thinking (Papert 1993; Papert, Solomon 1971) overbears seemingly even such opposed gaps.

We do not conceal on the fact that through this approach with the support of computers in particular through the use of LOGO, LISP and CAS Mathematica we have taught many highly motivated students in computer classes at grammar school from age 16 to 18 and teacher students in Mathematics and Informatics at age 19 at the university for a couple of years with the result of a profound knowledge of elementary data – structures on one hand and of Elements of Linear Algebra on the other hand finally.

All implementations in this contribution are made in MSW LOGO in terms of uniformity, adaptations in LISP and CAS Mathematica® can be set up easily.

Wading through the Course's Design including students' monitorings

Atoms, Lists and Functions – Assembling and Disassembling

At the beginning all the students are informed of the data structures used in LOGO. Easy examples carried out by students in groups deal with manipulations on **atoms** and **lists** using the **functions** word and sentence for assembling and first, butfirst (bf), last, butlast (bl) for disassembling. Additionally the functions wordp and listp are used for checking the outputs. All the inputs are executed immediately after entering the code into the **Command line**.

The following section shows a work – sheet of average level of difficulty the students have to work with cooperately.

Given: [[1 2] 23 [24,25]]

- Combose the given list!
- Extract
  - the element 2 as a list,
  - the element 23 as a word and
  - the word 2425 from the given list!

Possible solutions are:

```
show [[1 2] 23 [24 25]]
[[1 2] 23 [24 25]]
```

```
show bf first [[1 2] 23 [24 25]]
[2]
```

```
show first bf [[1 2] 23 [24 25]]
23
```

```
show word first last [[1 2] 23 [24 25]] last last [[1 2] 23 [24 25]]
2425
```

Some of the students solved the last problem in two steps

```
show last [[1 2] 23 [24 25]]
[24 25] and then
```

```
show word first last [[1 2] 23 [24 25]] last last [[1 2] 23 [24 25]]
2425
```

which is absolutely reasonable as they want to ensure themselves of the object which they will be disassembling further.

Vectors and Matrices

Definitions and Basic Attributes

In the following it remains to be seen that the elements of Linear Algebra can be obtained from the fundamentals in the first work – sheet by the strategies of Transfer (Schubert, Schwill 2004) and Analogy (Herber, Vásárhelyi 2002) which are important for Constructionistic Thinking.

Hence the outcome of a consequent implementation of the concept for the methodology is as follows. In one respect the teacher must namely be the guider of the course as he owns the professional and educational competences. He develops work – sheets for each step of the course presented in the following with permanent regard to the learning process of the students. In short: During the construction process of the students the teacher withdraws from the classroom union and gives support individually in the other respect (Fuchs 2007, p. 183).

We will call elements like [1 2 3] or [-2.3 0] **vectors of dimension three** respectively of **dimension two**. Generally we will call $[x_1, x_2, \ldots, x_k]$ **vectors of dimension k**.

Immediately we have to cope with a new problem. What does the attribute **dimension** address? Very soon the students will find out that the dimension equals the number of elements of the vector.

The predefined LOGO function count satisfies our problem. The students can find out the dimension of a **vector** easily. For example typing in

```
show count [-2.3 4 5.75]
```

into the **Command Line** will yield 3 which is the correct answer.

Gradually we define elements like [[1 2 3][4 5 6][7 8 9]] or [[1 0][0 1]] as **3 × 3** respectively **2 × 2 matrices**. Generally we will call $\begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$ **m × n matrices**.

The next steps are up to the students' responsibility in a broader extent.

The first task will soon be done by the students namely to show the relation between the 'list – of – lists' – representation such as [[1.5 3.43][2.756 3.1]] and the common symbolic 'mathematical representation' $\begin{bmatrix} 1.5 & 3.43 \\ 2.756 & 3.1 \end{bmatrix}$.

But when using the count function again the students soon become aware that for example

```
show count [[2 3][4 5]]
```

yields 2. A result which is absolutely justifiable by the students – two lists are the elements in the given list – but it is not adequate for **matrices**.

So the second task will be more difficult. Analysing the definition the students will answer the question according to the **dimension** of a **matrix** by bringing in the number of rows and the number of columns for a new definition of the attribute. We agree to this statement but for this reason the dimension of a matrix must be splitted into a **row – dimension** and a **column – dimension** consequently. Now the students' big challenges consist in the construction of a dim – **module** for **matrices**.

Additionally we want to introduce the **Programming – (Editing) Mode** of MSW LOGO and the concept of the **parameter** or **variable** in LOGO.

Empirically this implementation necessitates some time for experimenting to become familiar with the **Editing Mode**. Most of the students will approach by 'trial and error' when modifying the code in the edit – window. Only few of them focus on structuring the design before entering the code. Never astonishingly these students reach the desired implementation more quickly.

```
to dim :mat
op sentence count :mat count first :mat
end
```

Further Attributes

We sustain our course by going on the explorations of further properties of the elements **vector** and **matrix**.

Forces, velocity and acceleration are terms the students know from their Physics lessons. The Natural sientist uses **vectors** to describe such concepts. We take advantage of these interdisciplinary aspects in our concept.

As a matter of course the students argue that all these vectors are characterized by two informations, one is the **direction** the other the **absolute value** of the term. We will focus on the second one.

We will solve the problem by transferring it to Geometry and then by stepwise refinement.

The two dimensional problem:

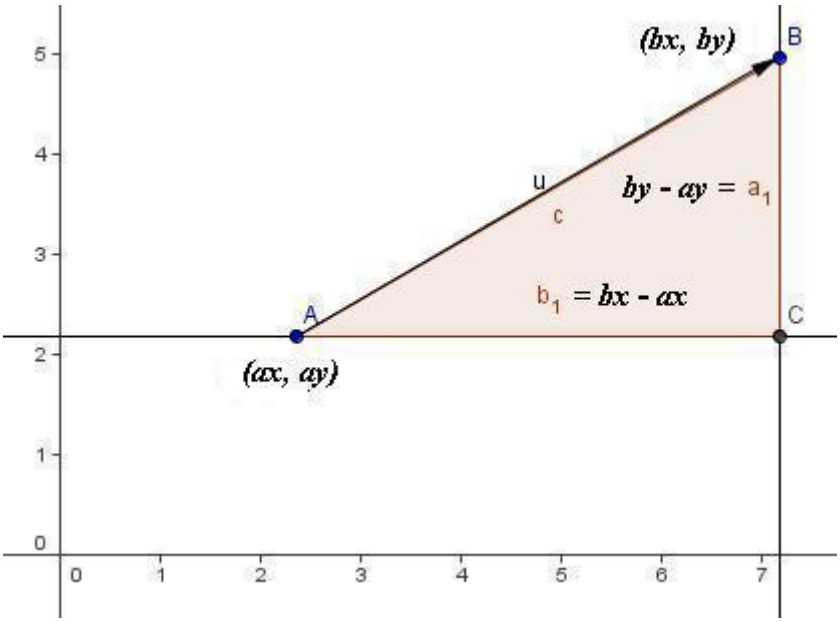

*Figure 1. Length of **vector** u*

Figure 1 suggests that the **absolute value** c of **vector** u can be found by the Pythagorean Theorem easily: $c = \sqrt{a_1^2 + b_1^2}$, where $a_1 = b_y - a_y$ and $b_1 = b_x - a_x$.

The solution for the three dimensional problem should be devolved by the students.

Although the generalization $c = \sqrt{u_1^2 + u_2^2 + \ldots + u_k^2}$ (with $u_1 = b_{x_1} - a_{x_1}, u_2 = b_{x_2} - a_{x_2}, \ldots, u_k = b_{x_k} - a_{x_k}$) is beyond graphical representation the expression is accepted by the students willingly.

But now our interest concentrates on the LOGO implementation of the attribute.

Already from this stage on we use the widely unfamiliar LOGO functions MAP and APPLY for the manipulations of our **lists** (**vectors**). Once again we can avoid recurrent value assigments (Fuchs, Siller, Vasarhelyi 2008).

```
to abs_val :v
op sqrt apply "sum map [? * ?] :v
end
```

We will test our function to err on the side of caution:

```
pr abs_val [2 3 4]
```

yields

```
5.3851648071345
```

which can easily be checked as the right answer.

But we can also discover further attributes for **matrices**. We want to pick out the **symmetry**. On one hand we choose this property as it is of notably importance when teaching algorithms. More efficient strategies can be gained when adjacency matrices which represent the implementations of graphs are **symmetric**. On the other hand we settle on this attribute as we will come across with another interesting **module** namely *x* on the way to the final implementation of *symmetry*.

So let's go back to the problem. A **matrix** is **symmetric** when it fits in with its **transposed matrix** which evolves by mirroring the elements on the main diagonal $(x_{11}, x_{22}, \ldots, x_{kk})$.

Although this attribute sounds very repellent the students will have no problem with it as some of the students' comments like **symmetric matrices** must be quadratic give evidence to our statement.

Hence we will implement a LOGO transpose function first. Investigating a reduced problem – a **symmetric 3 × 3 matrix**

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$

Figure 2. symmetric 3 x 3 matrix

we will find out that the strategy will be to generate a new **matrix** where $x_{ij} = x_{ji}$ for *i, j = 1, 2 3*. We are satisfied with the **algorithm** and generalize it for i, j = 1, 2, .. k but the implemetation of this strategy makes a most interesting new module necessary. We will call it simply *x*.

x will be a selector function in the two parameters *:i* and *:j* using the predefined LOGO item function. In doing so *x : i :j* addresses the element x_{ij} in the **matrix** *:mat*.

```
to x :i :j :mat
op item :j item :i :mat
end
```

The implementation of the **module** to check the **symmetry** brings in some very challenging new ideas which must be explored in a dialogue between teacher und students. Thereby a main

focus rests on the **control structure** of **conditional branching** that comes along with the **recursion**.

```
to symmetry :mat :row :col
if equalp dim :mat sentence :row :col [op "true]
if not equalp x :row :col :mat x :col :row :mat [op "false]
if not greaterp :row first dim :mat [if lessp :col last dim :mat [op symmetry :mat :row :col+1]]
op symmetry :mat :row+1 1
end
```

When discussing the LOGO source code the main focus will be on the two **conditions** *if equalp dim :mat sentence :row :col [op "true]* and *if not equalp x :row :col :mat x :col :row :mat [op "false]*. Such conditions are essential in programming recursive functions to avoid infinity in executing. Empirically they are often ignored by the students.

We are satisfied with the **module** as

```
show symmetry [[1 -2 3][-2 4 0][3 0 5]] 1 1
```

yields

```
true
```

whereas

```
show symmetry [[1 -2 3][-2 4 1][3 0 5]] 1 1
```

outputs

```
false.
```

We expect that our severe philosophy in implementing the code did not escape the attention of the reader as we strictly avoid value assignments using make. Our intention is to show the students that these commands which they know from courses in imperative programming very well are not necessary for a consequent functional programming style.

Discovering Operations

Exemplarily we will discuss some operations with **vectors** and **matrices**. Constructionistic Thinking in this case means that we will not define the operations traditionally but gain them by experimenting and playing.

Our first example is about the similarity of documents. Although the solution of the following problem is well – known in Information Theory this fact is not communicated to the students.

The modelling process will start with the question how to indicate the similarity of a document. After some discussions we agree upon the strategy to bring the absolute frequency of some relevant terms in the document into account for similarity.

We decide to describe each document by a **vector** with the absolute frequencies as its components. [1 5 0 0 1 2 8 1 0 1] will be an implementation for a document where the number of relevant terms *n* equals 10.

After some further continuative discussions with the students we decide to multiply the according elements of two documents (= **vectors**) and sum up these products finally. Hence the output sum is a rate for the similarity of the two documents.

Main arguments for this solution are:

- If the absolute frequency of a relevant item equals zero then the result will be zero regardless of the value of the other factor.

- If the absolute frequency of a relevant item equals one then the value of the other factor will be the value of the product.
- If both values of the absolute frequencies of the relevant items are bigger than one then the value of the product will be bigger than the value of each factor.

Now we are prepared for the LOGO implementation which should be done mainly by the students.

```
to similarity :v1 :v2 :pv
if equalp :v1 [][op apply "sum :pv]
op similarity bf :v1 bf :v2 fput product item 1 :v1 item 1 :v2 :pv
end
```

Finally we nominate this operation with **vectors inner product** or **scalar product** and test the **module**. The results are very satisfying.

show similarity [1 5 0 0 1 2 8 1 0 1] [0 4 0 1 1 3 5 0 4 1] [] yields 68 and
show similarity [1 5 0 0 1 2 8 1 0 1] [5 0 4 3 1 2 1 0 5 0] [] outputs 18.

Even if we only fly over the inputs we will consider the first two vectors as more similar than the second ones.

The second example will bring us to the **matrices** and it may be called metamorphoses.

We start with a square. It is a magic one as the sums of all its rows, columns and diagonals equals fifteen.

2	7	6
9	5	1
4	3	8

Figure 3. Magic Squares

The square will be implemented as **matrix**.

Now the students' problem is to create a new **3 × 3 matrix** by multiplying all the rows' permutations of the given square. For the further process it is indicated to name the **matrices** with A which is the original magic square and B, C, D, E and F for its rows' permutations.

A·A	A·B	A·C	A·D	A·E	A·F
B·A	B·B	B·C	B·D	B·E	B·F
C·A	C·B	C·C	C·D	C·E	C·F
D·A	D·B	D·C	D·D	D·E	D·F
E·A	E·B	E·C	E·D	E·E	E·F
F·A	F·B	F·C	F·D	F·E	F·F

Table 1. Magic Square Permutations

Back order the question 'What do we mean by **multiplying two matrices**?' is still open.

After some discussion we stick back to the knowledge that each row or column is a **vector**. In the example before we became acquainted with the **scalar product** expressed in the similarity **module**.

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} \cdot \begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$$

*Figure 4. Identifying the **scalar product***

So let us implement the already existing **module** as a new function for the product of two (quadratic) **matrices**.

```
to metamorph :mat1 :mat2 :mat3 :i :v
if :mat1=[][op :mat3]
if greaterp :i 0 [op metamorph :mat1 :mat2 :mat3 :i-1 fput similarity last :mat1 pick_out :mat2 [] :i [] :v]
op metamorph bl :mat1 :mat2 fput :v :mat3 3 []
end
```

with

```
to pick_out :mat :v :i
if :mat=[][op :v]
op pick_out bl :mat fput item :i last :mat :v :i
end
```

Only the students with outstanding abilities in functional programming are able to implement this LOGO **module**. Nevertheless there is enough room for all the students to participate when recapitulating the given problem.

Now the additional task is not only to generate the permutations of the given magic square but to find out that some new generated **matrices** will be **symmetric** such as C · B with $B = \begin{pmatrix} 4 & 3 & 8 \\ 9 & 5 & 1 \\ 2 & 7 & 6 \end{pmatrix}$ and $C = \begin{pmatrix} 9 & 5 & 1 \\ 4 & 3 & 8 \\ 2 & 7 & 6 \end{pmatrix}$.

```
show metamorph [[9 5 1][4 3 8][2 7 6]] [[4 3 8][9 5 1][2 7 6]] [] 3 []
```

yields

```
[[83 59 83] [59 83 83] [83 83 59]]
```

Finally we use our already discovered symmetry **module** for automatic testing.

```
show symmetry [[83 59 83] [59 83 83] [83 83 59]] 1 1
true
```

Final Short Perspectives on the Constructivist Approach

The main intention of our paper was to put the design of a different course which is partly going beyond Mathematics and Computer Science in school up for discussion. LOGO looms large in the concept which is rigid in no case but very open. It leaves multitudinous possibilities of constructionistic acting to committed teachers namely be it the discussion of additional attributes and operations as well as the introduction of further basic elements of Linear Algebra such as the determinant.

Furthermore the course offers numerous opportunities for inner differentiation wherewith teachers are able to make the grade to different levels of achievements through activities adapted to the students' capacities.

References

Fuchs, K. J. (2007). *Fachdidaktische Studien*. Shaker Verlag, Aachen.

Fuchs, K. J.; Siller, H. – S.; Vásárhelyi, É. (2008). *Informatics With Casio CP 300+*. CASIO Europe GmbH, Norderstedt.

Herber, H. – J.; Vásárhelyi, É. (2002). *Das Unterrichtsmodell „Innere Differenzierung einschließlich Analogiebildung" – Aspekte einer empirisch veranlassten Modellentwicklung*. In Salzburger Beiträge zur Erziehungswissenschaft 6, Nr. 2, pp. 5−19.

Papert, S. (1993). *Mindstorms – Children, Computers and Powerful Ideas*. 2nd Edition Basic Books, New York.

Papert, S.; Solomon, C. (1971). *Twenty Things To Do With A Computer*. MIT, A. I. Laboratory, AI MemoNo. 248, LOGO Memo No. 3.

Schubert, S.; Schwill, A. (2004). *Didaktik der Informatik*. Spektrum Akademischer Verlag, Heidelberg.