INT. J. MATH. EDUC. SCI. TECHNOL., 2003
VOL. 34, NO. 3, 385–402

Taylor & Francis
Taylor & Francis Group

Mimicry of proofs with computers: the case of Linear Algebra

ORIT HAZZAN*
Department of Education in Technology and Science, Technion - Israel Institute of Technology, Israel
e-mail: oritha@techunix.technion.ac.il

and RINA ZAZKIS
Faculty of Education, Simon Fraser University, 8888 University Drive, Burnaby, BC, Canada V5A 1S6.
e-mail: zazkis@sfu.ca

(*Received 25 June 2002*)

This article examines the idea of 'following the flow of a proof with an example' in order to assist the learner in the challenging task of understanding mathematical proofs. This strategy is termed 'mimicry of a proof'. However, such mimicry can be impractical or unreasonably demanding when the mathematical objects in the proof are difficult to manipulate without technological enhancement. This is the case with many proofs in Linear Algebra, in which the manipulated objects are vectors or matrices. Therefore, the article focuses on the idea of proof mimicry with a computer algebra system (CAS). As examples, this strategy is applied to the proofs of two theorems: the basis theorem and the orthogonalization theorem. In addition, pedagogical guidelines to be followed in constructing a set of computer activities for students are presented and examined.

1. Introduction

Teaching students to understand, write and appreciate mathematical proofs presents one of the major challenges in mathematics education. This challenge surfaces when students are introduced to formal proofs in secondary school and still exists when mathematics is taught at the undergraduate level. As it turns out, many of the proofs of mathematical theorems in the context of Linear Algebra involve additional complexity for the learners. These theorems deal with abstract objects, such as matrices or vector spaces, and some (intricate) manipulation of these objects is carried out as part of the proof. Specifically, the proofs often involve mathematical induction, a strategy that is not fully grasped by many learners. Moreover, and maybe most importantly, as Linear Algebra is a first or a second year course in most institutions, these proofs appear relatively early in learner's experiences with proofs.

In this article we present a pedagogical strategy for helping students unravel complex and condensed proofs. This strategy is 'mimicry of proofs'. We start with

* The author to whom correspondence should be addressed.

International Journal of Mathematical Education in Science and Technology
ISSN 0020–739X print/ISSN 1464–5211 online © 2003 Taylor & Francis Ltd
http://www.tandf.co.uk/journals
DOI: 10.1080/0020739031000108628

discussion of students' difficulties with mathematical proofs. We then introduce the idea of proof mimicry with a numerical example. We proceed with a discussion of students' difficulties in learning Linear Algebra, focusing on Linear Algebra proofs. We present examples of student activities based on the idea of mimicry of proofs in Linear Algebra. We conclude by sharing an experience of implementation of this strategy in a classroom. Our aim is to present a pedagogical approach. Thus, though we have data indicating that students do improve their understanding of the relevant Linear Algebra concepts, this article focuses on the pedagogical aspect rather than on the research aspect of our experience.

2. Proofs in learning mathematics

Abundant research that explores students' difficulties with mathematical proof has already been carried out. Some of these difficulties are discussed with respect to specific proofs, or specific kinds of proofs, such as proof by induction or indirect proof [1–4]. Other research undertakings indicate that some students do not appreciate the need for a formal mathematical proof as part of their 'proof schemes' [5]. Specifically, what is accepted by these students to be a convincing argument is often inconsistent with mathematical community conventions. Thus, 'proof by example' may serve as sufficient argument for a student possessing an empirical inductive proof scheme.

Though we do not accept particular examples as mathematical proofs, we do acknowledge their role in learning to prove and in understanding proofs. According to Edwards [6] expectation or belief which is based on a specific set of examples is an important element in the 'territory before proof' and a stepping stone towards conjecture and deductive reasoning. Rowland [7] focuses on the notion of generic proof; that is, 'proof by example' that does not capitalize on any specific features of the particular example. While 'proof by generic example' on its own does not constitute a formal mathematical proof, considering a generic example and extracting its non-particular features provide a bridge towards creating a general mathematical proof.

A large amount of research literature focuses on learners' writing proofs. Less attention is paid to topics related to reading proofs, such as reading proofs in a way that leads to student understanding of the proof and to student ability to analyse its structure, rather than the ability to recreate it from memory. Selden and Selden [8] have shown that undergraduate students have difficulty in interpreting the text of proofs and that often these students are unable to distinguish between a valid proof from a meaningless manipulation of symbols. It was suggested that undergraduates should be allowed ample opportunity to evaluate proofs.

Almost two decades ago, acknowledging the challenge in understanding a complex proof, Leron [9] presented the idea of structuring proofs, that is, arranging the presentation of a proof in a way that its structure is explicit to the reader. However, uncovering such a structure is not predominant in textbooks or lecture halls. As a result, most proofs are still presented to the learner in a linear fashion.

The question that arises is: how can one *truly* understand, and make sense of a proof? Specific examples are often used in order to demonstrate the correctness of a statement or theorem that has been or is about to be proved. Specific examples are also used as a bridge in building a general proof [7]. Another important role that can be attributed to specific examples is that of understanding a proof rather

than that of creating it. We suggest that following the flow of a proof (as it is presented in a conventional linear fashion), with a specific example helps in understanding the proof. We call this strategy 'proof mimicry'.

We focus here on construction proofs. Not to be confused with proofs of correctness of geometrical constructions, construction proofs are proofs that entail the construction of a new mathematical object (be it a function, a set or even just a number) as one of their central features. Thus, the renowned Cantor's proof, often referred to as the 'diagonal method', establishes the equality of the cardinality of rational numbers and the cardinality of natural numbers by constructing a function, a one-to-one correspondence. Euclid's proof of the infinity of primes introduces a construction of a new number. We believe that following this construction process with a specific example may help in understanding the core argument. In the next section we consider this example in further detail.

3. Example of proof mimicry: infinity of primes

As short proofs are sometimes the most difficult to unravel, several researches suggest pedagogical approaches to help students understand such proofs. For example, Leron [1] and Rowland [7] discuss the pedagogical approach for turning a proof-by-contradiction into a constructive proof.

Our purpose here is less ambitious. In this section we illustrate how specific examples may be helpful. Table 1 presents the proof of the theorem that states that there are infinitely many prime numbers. To demonstrate the pedagogical power of specific examples we simply follow the proof with numerical examples.

Suppose that the finite set of primes is $S = \{2,3\}$. Then the new constructed number $M = (2 . 3) + 1 = 7$. In this case 7 is a 'new prime', prime that has not been included in the presumably finite list of primes.

Gradually enlarging the set of primes, we get:

$(2 . 3 . 5) + 1 = 31$, 'new prime'
$(2 . 3 . 5 . 7) + 1 = 211$, 'new prime'
$(2 . 3 . 5 . 7 . 11) + 1 = 2311$, 'new prime'

At this stage, and maybe even earlier, an expectation is created that adding 1 to any multiple of primes will generate a new prime number. Therefore the next step is crucial:

$(2 . 3 . 5 . 7 . 11 . 13) + 1 = 30031$
but, $30031 = 59 * 509$

Indeed, 30031 is a composite number, and its prime factors, 59 and 509, are not elements in the presumably finite set of primes. In our experience this particular example has a greater explanatory power for many learners than the argument 'M has a prime factor p different from $p_1,\ldots,p_n$, since dividing M by any of the primes in S gives a reminder of 1'.

Theorem.	There are infinitely many prime numbers.
Proof.	Suppose, on the contrary, that the number of primes in finite, $p_1,\ldots,p_n$. Consider $M = (p_1 . p_2 \ldots . p_n) + 1$. Then M is a prime number or M has a prime factor p different from $p_1,\ldots,p_n$ Contradiction.

Table 1. Proof of infinite number of primes.

The human mind is often not satisfied with the knowledge that some objects exist. There is a desire to point out *exactly* what these objects are. Similarly, unraveling a construction process with a specific example helps understand *exactly* how the construction works. In the above illustration we could unravel the proof by following it along with numerical examples. As the calculations involved in the numerical examples are simple, they could easily be carried out mentally or with the help of a hand-held calculator. There are cases, though, where the mathematical objects constructed in proofs are too complex to be manipulated with such ease. Many of the proofs in Linear Algebra present such complexity. In what follows we discuss general difficulties that students face in the Linear Algebra course and suggest a way to assist them with the complexity of Linear Algebra proofs.

4. Learning Linear Algebra

The experience of teaching and learning Linear Algebra is often referred to as difficult and frustrating [10, 11]. One obvious reason for this is that this course appears early in a sequence of courses for specializing in Mathematics, before students are fully accustomed to the demands of the discipline for rigor and abstraction. Dorier *et al*. [12] suggest that for many students 'Linear Algebra is no more than a catalogue of very abstract notions' (p. 95). These researchers further describe students' perception of Linear Algebra as 'an avalanche of new words, new symbols, new definitions and new theorems' [12]. However, the abundance of novelty with which students are faced presents only part of the difficulties. According to [13], Linear Algebra is the first course in which students encounter 'full fledged mathematical theory, built systematically from the ground up, with all its fuss about making assumptions explicit, justifying statements by reference to definitions and already proved facts' (p. 65). One difficulty that students experience with this 'full fledged theory' is its generality, that is, the difficulty is in shifting attention from particular objects to general structures.

A further difficulty is with the objects themselves—vector spaces, bases, kernels, etc.—which are found by learners to be difficult, probably because such concepts require a higher level of abstraction than students are used to. Furthermore, the existence of multiple levels of description and representation (general theory of vector spaces, more specific theory of R_n and geometric representation of R_2 or R_3) and the constant shifting among these levels provide an additional obstacle.

Instructors and researchers acknowledge these difficulties and, as a result, are seeking pedagogical approaches that make Linear Algebra more accessible. Utilizing computer technology could be a possible means to this end. In the next section we focus on the use of computer technology for the learning of Linear Algebra.

5. Linear Algebra and technology

Computer technology is slowly making its way into learning and teaching undergraduate mathematics. About two decades ago the emerging discussion surrounded possible benefits of utilizing computers and the feasibility of computer-supported approaches in the existing structure of undergraduate mathematics education. Today, though the 'chalk and tack' prevails in many Linear Algebra classes, the use of technology is acknowledged and often mentioned as recommendation [14]. The discussion among the researchers has

shifted from 'for and against technology' to 'how to utilize technology?', 'what software to use?', 'what activities to develop?'.

One possible approach to integrating technology in Linear Algebra builds on work with computer algebra systems (such as MATLAB and Maple) with which students can manipulate n-tuples and matrices (cf. [15, 16]). A different perspective is suggested by Sierpinska *et al*. [17], who designed an approach that provides students with a 'soft entry' to Linear Algebra through the two-dimensional vector space of dynamic geometry environments (in that case it is Carbi). This approach builds on the geometric mode of description of Linear Algebra. However, despite the work with low dimensions and the use of visualizations (as the implementation of this method advocates) students' difficulties persisted when the students were in the process of moving to the structural abstract mode [18].

Hillel [10] proposes four levels of activity with the Computer Algebra System (in this case Maple) that support the learning of Linear Algebra:

(1) computing (moving beyond 2- and 3-dimensional examples);
(2) becoming acquainted with objects and operations;
(3) exploring and getting inductive evidence of proofs;
(4) creating a milieu which supports mathematical discourse.

We consider the types of activities described by Hillel as dimensions rather than levels, to avoid the presumption of hierarchical structure. In this article we demonstrate a new fifth dimension—mimicry of proofs—that encompasses and builds on the four previously mentioned dimensions.

6. Mimicry in Linear Algebra

In the previous sections we laid out the rationale behind the mimicry of mathematical proofs in general and the importance of such implementation in the context of Linear Algebra in particular. In this section we present two examples for mimicry of proofs with the aid of a computer algebra system (Maple). The two examples are taken from the traditional introductory Linear Algebra course. The first example is the proof of the theorem that states that 'in a finite-dimensional vector space, all bases have the same number of elements' (referred to in this article as the Basis Theorem). It is illustrated how the proof of the theorem can be represented by Maple functions[1]. In addition to the proof mimicry itself we present a set of activities designed for teaching students the proof. We found this theorem suitable for the illustration as its proof is based on a lemma the proof of which relies on an inductive process. As it turns out, Linear Algebra instructors consider this proof to be the most sophisticated proof presented to students in an introductory Linear Algebra course. Our second example—the Gram-Schmidt theorem—is presented very briefly. In this case we illustrate only the mimicry idea. Specific activities for students can be designed similarly to the first example and according to the instructor's preference.

[1]The illustration is based on an experiment that Orit Hazzan (one of the authors) together with David Chillag and Uri Leron all from the Technion - Israel Institute of Technology, carried out in 1997–1999 as part of a campus wide project that aimed to integrate information technologies into Technion teaching.

Theorem. In a finite-dimensional space, all bases have the same number of elements.
Proof. Follows from the following *lemma*.

Lemma.
Let V be a finite-dimensional vector space and $\{x_1, x_2,\ldots,x_n\}$ be a basis for V with n elements. If $\{y_1, y_2,\ldots,y_m\}$ is a set of m linearly independent vectors in V, then $m \leqslant n$.

Proof of Theorem.
Let $\{x_1, x_2,\ldots,x_n\}$ and $\{y_1, y_2,\ldots,y_m\}$ be bases for V, a finite-dimensional space. Since $\{y_1, y_2,\ldots,y_m\}$ is a linearly independent set and $\{x_1, x_2,\ldots,x_n\}$ is a basis, by the above lemma $m \leqslant n$. If we consider $\{x_1, x_2,\ldots,x_n\}$ to be a linearly independent set and $\{y_1, y_2,\ldots,y_m\}$ to be a basis, by the above lemma $n \leqslant m$. Thus, $n = m$.

The above *lemma* is based on the *Replacement lemma*.
Let B be a set of linearly independent vectors in the spanning space of a set of vectors A. For all subset B_1, $B_1 \subseteq B$, exists A_1, $A_1 \subseteq A$ and $|A_1| = |B_1|$, that B_1 replaces. In short, $(A - A_1) \cup B_1$ spans the same span as A.

Table 2. Proof of the theorem that states that in a finite-dimensional space, all bases have the same number of elements.

Example 11: The Basis Theorem
Table 2 presents the proof of the theorem discussed in this section.

In what follows we present the proof of the Replacement Lemma (Cf. table 2). This is the proof that is, in fact, mimicked in the set of activities for students presented in Appendix B. As mentioned above and described in what follows, the proof consists of an inductive process in which the elements of B_1 are added one by one at the beginning of the spanning indexed set we got so far. At each step the first element (from the left) in the spanning indexed set, that is linearly dependent by its previous ones, is thrown away. Since the eliminated vector cannot be one of the vectors that we just added to the spanning indexed set, the elements of B_1 are exhausted before the elements of the spanning set. In the case of equality, the elements of B_1 are exhausted at the same step of the inductive process with all the elements of the spanning set. The above inductive process is based on the fact that if an ordered set of vectors is linearly dependent, then there is one vector that is a linear combination of its previous ones.

As was previously mentioned, according to Linear Algebra instructors, this proof causes students a number of difficulties. This observation is followed from at least two characteristics of the proof. First, students should mentally hang on to algebraic structures (spanning set, linear dependency and linear combination). At the stage in which the proof is presented, not all the students have as yet constructed these objects mentally. Second, the inductive process is usually described in a condensed way, and students cannot unravel it, understand what it contains and follow it. The following computer activities are intended for the purpose of helping students overcome these difficulties. Before the set of activities for students is presented, the guidelines that were adopted in the design of the activities are outlined.

The main guideline that was adopted in the development of the Maple activity, in which the proof of the Basis Theorem is mimicked, focuses on the construction of (computational and mental) mathematical objects. This was done in the spirit of constructionism [19]. Thus, in order to implement the idea of 'out-of-the-mind construction leads to mental construction', mathematical concepts are represented

by Maple functions. The idea of representing mathematical objects by means of programming languages is not new. For example, Dubinsky and Leron [20] present an activity in which students *discover* Lagrange Theorem through a set of ISetL activities (pp. 108–111). A similar approach is adopted here for mimicking mathematical proofs.

Further, the set of student activities consists of two main kinds of tasks. The first kind of tasks is based on the *exploration of a given function behavior* (e.g., basis). Concepts were introduced to students by way of Maple activities prior to the formal presentation of the concepts in the lecture. Thus, when students were asked to reveal the meaning of a given function, even a meaningful name (such as *basis*) was unknown to them and therefore, of no use to them. As a result, students had to reveal the meaning of a given function based on their knowledge at the time. In that process they had to manipulate mentally what they already knew, and to construct a new mental object from those building blocks. The second kind of tasks is based on *object construction* (e.g. LD, which determines whether a given set of vectors is linearly dependent). In this case students were asked to construct a new function which mimics a mathematical object (and its properties). The *mental construction* which is associated with such *function construction* can be explained by way of the constructionist perspective mentioned above. Furthermore, as the properties of the mathematical notion are mimicked, the students have to approach the concept as an object. Hence, we believe that this kind of activities may lead students to an object conception of the concept under (mental and computational) construction (cf. [21, 22]).

Appendix A presents the functions on which the mimicry of the proof of the Basis Theorem is based. Appendix B presents the entire set of activities developed for student engagement with the theorem. In what follows several illustrative sections from the set of student activities are presented. The instructions presented in these activities should be executed in Maple. However, in order to make the presentation here self-contained for the reader, the instructions were carried out and the computer responses were copied (indented). Our comments on each activity accompany the text presented to students.

The first set of instructions defines a set of vectors to be manipulated in later stages for checking algebraic connections and relationships. This set can be extended in order to allow for additional exploration.

```
>v1 := vector( [1,0,0]); v2 := vector( [0,1,0] );
>v3 := vector( [0,0,1]); v4 := vector( [1,1,1] );
                  v1 := [1, 0, 0]
                  v2 := [0, 1, 0]
                  v3 := [0, 0, 1]
                  v4 := [1, 1, 1]
```

The following exercise calls upon the students to explore the properties of an algebraic concept (basis), prior to introducing the concept in class. The idea is to let the students (mentally and computationally) construct a new mathematical concept based on their current knowledge. Guidance to explore the properties of basis by manipulating specific examples is provided with the aim of leading students from concrete thinking to abstract thinking.

Exercise. Try to understand what the function basis does.

```
>basis( [v1, v2, v3, v4] );
                [v1, v2, v3]
>basis( [v1, v2, v3] );
                [v1, v2, v3]
>basis( [v3, v2, v1] );
                [v3, v2, v1]
>basis([vector([1,1,1]),vector([2,2,2]),vector([1,-1,1]),
vector([2,-2,2]), vector([1,0,1]),vector([0,1,1])]);
                [[1, 1, 1] , [1, -1, 1], [0, 1, 1]]
```

In the next step students are given the Maple function which implements the concept of linear combination. In addition to the presentation of a mathematical concept, this function illustrates the idea of programming/constructing mathematical *concepts*. That is, students realize that they can build new mathematical *concepts*, and not only *processes*. This awareness is crucial for the consequent activities, in which more advanced mathematical concepts are constructed.

After the introduction of the Maple definition of linear combination, the students are invited to explore the properties of this notion. Then, they are asked to construct a Maple function LD (which determines whether a given set of vectors is linearly dependent) that, in fact, uses the function that has just been defined for them. The idea is to show the students that these constructions, that have been added to the programming language by the programmer, can also be used in the construction of more complicated mathematical notions. This idea is also illustrated by the next function, sub.space, that is given to the students and is used in later stages.

The following function LC (for Linear Combination) inputs a vector and a set of vectors and checks whether the vector is a liner combination of the vectors in the set:

```
>LC := proc(u,X)
>  RETURN ((basis(X)) = (basis([op(X),u])));
>end;
>B := [v1,v2,v3];  LC(v4,B);  evalb(LC(v4,B));
                B := [v1, v2, v3]
                [v1, v2, v3] = [v1, v2, v3]
                true

>evalb(LC(v4,[v1,v2]));
                false
```

Exercise. Write a Maple function LD which determines whether a given set of vectors is linearly dependent.

The following function sub.space inputs two sets of vectors and checks whether the subspace spanned by the first set is a subspace of the space spanned by the second set. This is done by checking whether each vector in the first set is a linear combination of the vectors of the second set.

```
> sub.space := proc(T,S);
>  if nops(T) = 1 then RETURN(LC(T[1],S)); fi;
>  if LC(T[1],S) then RETURN(sub.space(T[2..nops(T)],S)); fi;
>  RETURN (LC(T[1],S));
>end;
```

The following vectors are defined for checking the properties of subspaces with a larger set of vectors. After the exploration (that is skipped here but is presented in Appendix 2), the function sub.space is used for constructing the function eq.subspace. As with sub.space the students are asked to explore the properties of the notion that is expressed by the function eq.subspace. This function is used in later stages to check that the replacement lemma does, in fact, work.

```
>r := vector([1,1,0]); s := vector([1,2,3]); t := vector([2,2,2]);
                r := [1, 1, 0]
                s := [1, 2, 3]
                t := [2, 2, 2]
>eq.subspace := proc(T,S);
>  RETURN (sub.space(T,S) and sub.space(S,T));
>end;
```

So far the basic components for the proof of the Basis Theorem have been constructed. At this stage, the functions LCP and LCP.index that implement the proof of the Basis Theorem are constructed one by one. The first one finds, from a given set of vectors, the first vector that is a linear combination of its previous ones. The second one outputs the index of that vector. For each of these functions, the students are asked to explore the function behaviour. In what follows only one example of such exploration is presented.

The following function inputs an ordered set S of vectors and, if the set is linearly dependent, returns the first vector which is a linear combination of its previous ones.

```
>LCP := proc(S);
>     for i from 1 to nops(S)-1 do
>      if LC(S[i+1],S[1..i])
>      then RETURN(S[i+1]); fi; od;
>     end;

>B1 := [v1,v2,v3,v4];  LCP(B1);
                B1 := [v1, v2, v3, v4]
                v4

>LCP.index := proc(S);
>     for i from 1 to nops(S)-1 do
>      if LC(S[i+1],S[1..i])
>      then RETURN(i+1) fi; od;
>     end;
```

```
>LCP.index([v1,v2,r,t,s]);
                3
```

The next two functions that were presented to the students implement one step in the inductive process on which the proof of the Basis Theorem is based. The idea behind the functions is explained to the students. After these two functions are presented and explored, the students are asked to implement the full process of the proof.

The following function inputs an ordered set of vectors S and, if the set is linearly dependent, 'kicks out' the first vector which is a linear combination of its previous ones.

```
>kick.LCP := proc(S);
>    i := LCP.index(S);
>     RETURN([op(S[1..i-1]),op(S[i+1..nops(S)])]);
>    end;

>r := vector([1,1,0]);  kick.LCP([r,v1,v2,v3,v4]);
               r := [1, 1, 0]
               [r, v1, v3, v4]
```

The following function performs the main step in the proof that any two bases of the same vector space have the same number of elements. The function inputs a vector q and an ordered set S of vectors, adds the vector at the beginning of the set, and (if possible) replaces one of the vectors in the set with the given vector, so that the span is not changed.

```
>replace := proc(q,S);
>     RETURN (kick.LCP([q,op(S)]));
>    end;
```

The following commands confirm the claim of the theorem; that is, that the spanned vector space does not change after each of the replacement steps.

```
>SP := [v2,s,v1,v3];
               SP := [v2, s, v1, v3]
>SP1 := replace(r,SP);
               SP1 := [r, v2, s, v3]
>SP1;
               [r, v2, s, v3]
>evalb(eq.subspace(SP,SP1));
               true
>SP2 := replace(v4,SP1);
               SP2 := [v4, r, v2, v3]
>evalb(eq.subspace(SP,SP2));
               true
```

Exercise. The previous lines have carried out the replacement of one vector at a time. Write a function set.replace which replaces any independent set of vectors with the same number of vectors in any spanning set.

Here, the written instructions guiding the students' activity end. Students are encouraged to define different sets of vectors and run through the procedures several times with different vectors as inputs. In such, the process underlying the theorem is unraveled and the burden of computation is left to the computer. From a cognitive perspective, the main goal of the activity is the construction of mathematical notions, whether the building blocks are Maple commands or functions that are programmed by the learners or the instructor. It is our belief that these computational constructions lead to the mental construction of the relevant mathematical notions.

Example 2: The Gram-Schmidt Theorem

This example illustrates how the Gram-Schmidt orthogonalization process is mimicked by Maple functions. Additional activities similar to those illustrated in our first example, can be added according to student level, the amount of time that the instructor wishes to dedicate to this topic, and the instructor's preference.

Appendix C presents the three functions that can be used for mimicking the Gram-Schmidt orthogonalization process. The function create.new.vector that is presented below is the core of the Gram-Schmidt orthogonalization process. It mimics the creation of the vector that is added to the orthogonal basis at the kth stage of the process.

```
>create.new.vector := proc (Ak, ortho_basis);
>     RETURN (Ak - SumVec (Ak, ortho_basis));
>end;
```

If the function is given to the students, they may be asked to check the function's behaviour with various inputs. Specifically, students may be guided to check the properties of the created vector for different cases and to explore why these properties ensure a creation of an orthogonal basis. If the construction of the function is assigned to students as a programming task, they would mimic the creation of the orthogonal basis.

After the creation of the new vector is completed, what remains to be done is to normalize the vectors and to gather them into one orthogonal basis. The first operation is simple when implemented in Maple; the second one turns out to be quite messy in Maple. However, as the heart of the process consists of the way the vectors are created one by one and the analysis of vector properties, it is reasonable to expect that constructing the above function create.new.vector will assist students in making mental constructions for the ideas that comprise the essence of the proof.

7. Conclusion

The core of this article is dedicated to the idea of mimicking mathematical proofs with computational tools. We focused on two examples in Linear Algebra and elaborated on some pedagogical and cognitive ideas and considerations behind

the choice of various tasks. The activities presented above do not exhaust all the possible options. For example, students may also be asked to give examples of different objects and to check whether the objects hold certain properties. For further discussion about the contribution of this kind of activity to students' learning see [23].

In conclusion we present responses of two students, in which they reflect on their learning at the end of the course. These two responses were selected since they express ideas of constructivism and constructionism in the students' own words. The two responses were given as answers to the following question: *In your opinion, how did the Maple activities influence your understanding of the Linear Algebra course?* These students, like others, present an indication that the CAS enhancement of Linear Algebra course aided their understanding.

Student A: Interaction with Maple helped me understand the material in depth. I think that each student should 'get involved' in the software and check his/her understanding. There is a difference between theoretical knowledge which is presented by the lecturer during the lesson and knowledge gained from personal experience, and this is exactly the point.

Student B: In my opinion, the activity improved my understanding of the course, because it enabled me to imagine the definitions and the proofs the first time I heard the lecturer present them. In addition, the option 'to play' with vectors and matrices helped me understand clearly what linear dependency is and what a basis is.

Proofs are the heart of mathematical activity and understanding of proofs is crucial for students' mathematical maturity. However, mathematical convention in the presentation of proofs often emphasizes compactness and elegance over explanatory power. Acknowledging this convention, Hanna [24] makes a profound distinction between proofs that 'prove' and proofs that 'explain'. The proofs of the theorems discussed above definitely prove, rather than explain. We believe that the mimicry of proofs helps explain proofs that do not explain.

Appendix A: Functions that mimic the proof of the basis theorem

This appendix presents the functions on which the mimicry of the proof of the Basis Theorem relies.

- The Maple function basis finds a basis for a vector space. Its input is a set (or list) of vectors and it returns the maximal set of linearly independent vectors from that set.
- The following function LC (for Linear Combination) inputs a vector and a set of vectors and checks whether the vector is a linear combination of the vectors in the set:

```
>LC:=proc(u,X)
    >RETURN ((basis(X)) = (basis([op(X),u])));
>end;
```

▶ The function *op* removes the brackets.

- The following function inputs an ordered set of vectors S and, if the set is linearly dependent, 'kicks out' the first vector which is a linear combination of the previous ones.

```
>kick.LCP:=proc(S);
>    i:=LCP.index(S);
>    RETURN([op(S[1..i-1]),op(S[i+1..nops(S)])]);
>  end;
>LCP.index:=proc(S);
>  for i from 1 to nops(S)-1 do
>   if LC(S[i+1],S[1..i])
>   then RETURN(i+1) fi; od;
>  end;
```

  ▶ The function *nops* returns the number of elements in a list.

- The following function replaces one vector at a time: the function inputs a vector **q** and an indexed set of vectors **S**, adds the vector at the beginning of the set, and (if possible) replaces one of the vectors in the set with the given vector, so that the span is not changed.

```
>replace:= proc(q,S);
    >RETURN (kick.LCP([q,op(S)]));
>end;
```

Appendix B: Worksheet implementing proof of the basis theorem

Notes

1. The instructions in the following worksheet should be executed in Maple. However, in order to make this Appendix self-contained, the instructions were executed and the computer responses were copied into the Appendix. In order to distinguish the Maple instructions from the computer responses, the computer responses are indented.
2. The worksheet presented here is a shorter version of the original activity that was given to the students. The original activity included additional exercises and mathematical explorations.

```
>with(linalg); {Calling the Linear Algebra package}
>v1:=vector( [1,0,0] ); v2:=vector( [0,1,0] );
>v3:=vector( [0,0,1] ); v4:=vector( [1,1,1] );
                                v1 := [1, 0, 0]
                                v2 := [0, 1, 0]
                                v3 := [0, 0, 1]
                                v4 := [1, 1, 1]
```

Exercise: Try to understand what the function basis does.

```
>basis( [v1, v2, v3, v4] );
                                  [v1, v2, v3]
>basis( [v1, v2, v3] );
                                  [v1, v2, v3]
>basis( [v3, v2, v1] );
```

```
                                        [v3, v2, v1]
>basis([vector([1,1,1]),vector([2,2,2]),vector([1, -1,1]),vec-
tor([2,-2,2]), vector([1,0,1]), vector([0,1,1])]);
[[1, 1, 1] , [1, -1, 1], [0, 1, 1]]
```

The following function LC (for Linear Combination) inputs a vector and a set of vectors and checks whether the vector is a liner combination of the vectors in the set:

```
>LC:=proc(u,X)
>    RETURN ((basis(X)) = (basis([op(X),u])));
>end;
>B:=[v1,v2,v3]; LC(v4,B); evalb(LC(v4,B))
                                    B := [v1, v2, v3]
                                [v1, v2, v3] = [v1, v2, v3]
                                          true
>evalb(LC(v4,[v1,v2]));
                                         false
```

Exercise. Write a Maple function LD which determines whether a given set of vectors is linearly dependent.

The following function inputs two sets of vectors and checks whether the subspace spanned by the first set is a subspace of the space spanned by the second set. It does this by checking whether each vector in the first set is a linear combination of the vectors of the second set.

```
>sub.space := proc(T,S);
>    if nops(T) = 1 then RETURN(LC(T[1],S)); fi;
>    if LC(T[1],S) then
       RETURN(sub.space(T[2..nops(T)],S));
     fi;
>RETURN (LC(T[1],S));
>end;
```

Note: The function sub.space can be written in a more elegant programming style. However, since the students were learning their first programming course in parallel to the introductory Linear Algebra course, we gave up the elegancy of programming to make the functions more readable.

```
>v1:=vector([1,0,0]); v2:=vector([0,1,0]);
v3:=vector([0,0,1]);
                                    v1 := [1, 0, 0]
                                    v2 := [0, 1, 0]
                                    v3 := [0, 0, 1]
>B:=[v1,v2,v3];
                                     B := [v1, v2, v3]
>r:=vector([1,1,0]); s:=vector([1,2,3]);
```

```
t:=vector([2,2,2]);
                                    r := [1, 1, 0]
                                    s := [1, 2, 3]
                                    t := [2, 2, 2]
>evalb(sub.space([r,s],B));
                                    true
>evalb(sub.space(B,[r,s]));
                                    false
>eq.subspace:= proc(T,S);
>    RETURN (sub.space(T,S) and sub.space(S,T));
>end;
>evalb(eq.subspace([r,s,t],B));
                                    true
>evalb(eq.subspace([r,v1],[v1,v2]));
                                    true
>evalb(eq.subspace([r,v1],[v1,v3]));
                                    false
```

The following function inputs an ordered set S of vectors and, if the set is linearly dependent, returns the first vector that is a linear combination of its previous ones.

```
>LCP:=proc(S);
>   for i from 1 to nops(S)-1 do
>    if LC(S[i+1],S[1..i])
>    then RETURN(S[i+1]); fi; od;
>   end;

>B1:=[v1,v2,v3,v4]; LCP(B1);
                                 B1 := [v1, v2, v3, v4]
                                          v4

>LCP([v4,v1,v2,v3]);
                                          v3

>t:=vector([2,2,2]); B2:=[op(B1),t];
                                    t := [2, 2, 2]
                                 B2 := [v1, v2, v3, v4, t]

>LCP(B2);
                                          v4

>LCP([v1,v2,v3,t,v4]);
                                          t

>LCP.index:=proc(S);
>   for i from 1 to nops(S)-1 do
>    if LC(S[i+1],S[1..i])
>    then RETURN(i+1) fi; od;
>   end;

>r:=vector([1,1,0]); s:=vector([1,2,3]);
t:=vector([2,2,2]);
```

```
                                    r := [1, 1, 0]
                                    s := [1, 2, 3]
                                    t := [2, 2, 2]

>LCP.index([v1,v2,r,t,s]);
                                              3
>LCP.index([v1,v2,t,s]);
                                              4
```

The following function inputs an ordered set of vectors S and, if the set is linearly dependent, 'kicks out' the first vector which is a linear combination of its previous ones.

```
>kick.LCP:=proc(S);
>    i:=LCP.index(S);
>    RETURN([op(S[1..i-1]),op(S[i+1..nops(S)])]);
>  end;
>r:=vector([1,1,0]); kick.LCP([r,v1,v2,v3,v4]);
                                    r := [1, 1, 0]
                                    [r, v1, v3, v4]
```

The following function performs the main step in the proof that any two bases of the same vector space have the same number of elements. The function inputs a vector q and an ordered set S of vectors, adds the vector at the beginning of the set, and (if possible) replaces one of the vectors in the set with the given vector, so that the span is not changed.

```
>replace := proc(q,S);
>    RETURN (kick.LCP([q,op(S)]));
>  end;

>eval(r); eval(B);
                                    [0, 1, 1]
                                    [v1, v2, v3]
>replace(r,B);
                                    [r, v1, v3]
>eval(s); SP:=[v2,s,v1,v3];
                                    [3, 2, 1]
                              SP := [v2, s, v1, v3]
>SP1:=replace(r,SP);
                            SP1 := [r, v2, s, v3]
>SP1;
                                    [r, v2, s, v3]
>evalb(eq.subspace(SP,SP1));
                                         true
>SP2:=replace(v4,SP1);
                            SP2 := [v4, r, v2, v3]
>evalb(eq.subspace(SP,SP2));
                                         true
```

Exercise. The previous lines have done a replacement of one vector at a time. Write a function set.replace which replaces any independent set of vectors with the same number of vectors in any spanning set.

Appendix C: The Gram-Schmidt orthogonalization process mimicked by Maple functions

```
> create.new.vector:= proc (Ak, ortho_basis);
>    RETURN (Ak - SumVec (Ak, ortho_basis));
> end;

> SumVec := proc (v, orth_bas);
>   if nops(orth_bas) = 1 then RETURN
     ((inner.product (v ,orth_bas[1])) * orth_bas [1]); fi;
>   RETURN (((inner.product (v ,orth_bas[1])) * orth_bas[1]) +

     SumVec (v, orth_bas [2.. nops(orth_bas)]));
> end;
>   inner.product := proc(u,v);
>   if nops (u) = 1 then RETURN (u [1]* v[1]); fi;
>   RETURN ((u[1] * v[1]) +
     inner.product(u[2..nops(u)],v[2..nops (v)]));
> end;
```

References

[1] Leron, U., 1985, *Educ. Studies Math.*, **16**, 321.
[2] Dubinsky, E., 1986, *J. Math. Behav.*, **5**, 305.
[3] Dubinsky, E., 1989, *J. Math. Behav.*, **8**, 285.
[4] Harel, G., 2002, in *Learning and Teaching Number Theory: Research in Cognition and Instruction*, edited by S. Campbell and R. Zazkis (Westport, CT: Ablex Publishing), pp. 185–212.
[5] Harel, G., and Sowder, L., 1998, in *Research on Collegiate Mathematics Education*, edited by E. Dubinsky, A. Schoenfeld, and J. Kaput (American Mathematical Society).
[6] Edwards, L., 1997, *Int. J. Comput. Math. Learning.*, **2**. 187.
[7] Rowland, T., 2002, in *Learning and Teaching Number Theory: Research in Cognition and Instruction*, edited by S. Campbell, and R. Zazkis (Westport, CT: Ablex Publishing), pp. 157–184.
[8] Selden, A., and Selden, J., 2003, *J. Res. Math. Educ.*, **34**(1), 4–36.
[9] Leron, U., 1983, Structuring mathematical proofs. *American Mathematical Monthly*, **90**(3), pp. 174–185.
[10] Hillel, J., 1995, in *5th International Symposium on Mathematics Education*, National University of Mexico, Mexico City.
[11] Hillel, J., and Sierpinska, A., 1994, in *Proceedings of the 18th International Conference on the Psychology of Mathematics Education*, Lisbon, August 1994.
[12] Dorier, J.-L., Robert, A., Robinet, J., and Rogalski, M., 2000, in *On the Teaching of Linear Algebra*, edited by J.-L. Dorier (Kluwer Academic), pp. 85–124.
[13] Hillel, J., and Sierpinska, A., 1997, in *Proceedings of the 18th International Conference for the Psychology of Mathematics Education*, edited by J. P. da Ponte and J. F. Matos, Lisbon, Portugal, Vol. 3, pp. 65–72.
[14] Harel, G., 2000, in *On the Teaching of Linear Algebra*, edited by J.-L. Dorier (Kluwer Academic), pp. 177–189.

[15] BAULDRY, W. C., EVANS, B., and JOHNSON, J., 1995, *Learning Linear Algebra with Maple* (John Wiley and Sons, Inc).
[16] DEEBA, E. and GUNAWARDENA, A., 1998, *Interactive Linear Algebra with Maple V* (New York: Springer-Verlag).
[17] SIERPINSKA, A., TRGALOVA, J., HILLEL, J., and DREYFUS, T., 1999, in *Proceedings of the 23rd Conference of the International Group for the Psychology of Mathematics Education*, edited by O. Zaslavsky, Haifa, Israel, Vol. I, pp. 119–134.
[18] SIERPINSKA, A., 2000, in *On the Teaching of Linear Algebra*, edited by J.-L. Dorier (Kluwer Academic), pp. 209–246.
[19] HAREL, I., and PAPERT, S. (eds), 1991, *Constructionism* (Norwood, NJ: Ablex).
[20] DUBINSKY, E., and LERON, U., 1994, *Learning Abstract Algebra with Isetl* (Springer-Verlag).
[21] DUBINSKY, E., 1991, in *Advanced Mathematical Thinking*, edited by D. Tall (Kluger Academic Press), pp. 95–123.
[22] SFARD, A., 1991, *Educ. Studies Math.*, **22**, 1–36.
[23] HAZZAN, O., and ZAZKIS, R., 1999, *FOCUS on Learning Problems in Mathematics*, **21**(4), 1.
[24] HANNA, G., 1989, in *Proceedings of the Thirteenth International Conference for the Psychology of Mathematics Education*, Vol. 2, pp. 45–51, Paris, France.