The place of computer programming in (undergraduate) mathematical practices

Laura Broley

Concordia University in Montréal, Department of Mathematics and Statistics, Canada, l_brole@live.concordia.ca

A recent survey of Canadian mathematicians found that while 43% of the participants use computer programming in their research, only 18% integrate this activity into their teaching. The first statistic highlights the significant place that programming may have in professional mathematics practice. The second suggests that such significance may not yet be acknowledged in undergraduate curricula across Canada. Our exploratory study involving 14 Canadian mathematicians sought to gain a deeper understanding of the place of programming in both contexts and therefore describe the gap from a more qualitative perspective. The views of our participants highlight some important issues that may require attention in order to bridge the identified gaps, should that be deemed the favourable direction to take.

Keywords: Mathematical practices, undergraduate teaching and learning, computer programming, institutional constraints.

INTRODUCTION

Several mathematicians and researchers in math education have reported on the disconnection between undergraduate curricula and professional practice. A recent quantitative survey of 302 Canadian mathematicians points to one possible aspect of the gap: while 43% of the participants reported using computer programming in their research, only 18% indicated that they rely on this activity in their teaching (Buteau, Jarvis, & Lavicza, 2014). Furthermore, when compared to the other technologies surveyed, programming was the only one for which such a gap was observed.

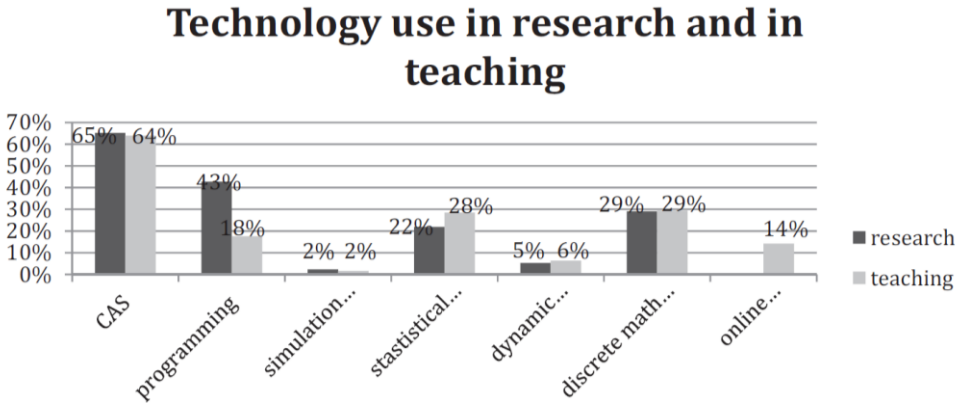

Figure 1: An intriguing gap for programming identified by Buteau et al., 2014.

The first statistic highlights the potential that programming may have for doing mathematics and the possible relevance of integrating it into university courses. The second inspires further research: why would such a gap exist? Buteau et al. (2014)

predict that the learning curve for programming is greater than for other tools such as computer algebra systems (e.g., *Mathematica*). They also mention the logistical obstacles of curriculum-wide integration. We were unaware of studies that verified these hypotheses or explored other possible explanations, while providing an in-depth look at how the 43% and 18% of mathematicians might be using programming in their research and teaching respectively. We hence wondered: what is the place of programming in mathematical research and undergraduate math education?

A FRAMEWORK FOR CAPTURING AND COMPARING PRACTICES

Our research question, posed as is, raises a subtle issue: since education contains two distinct perspectives, the teacher's and the student's, which should we take? After speaking with mathematicians, it appeared that many of them do not see fundamental similarities between the acts of teaching and researching. In contrast, we might assume that student experience should reflect mathematicians' work. We therefore decided that a comparison of research practices with *learning* practices (as opposed to *teaching* practices) could not only be more interesting, but also more important.

To be able to capture and compare such practices, we turned to Chevallard's (1998) Anthropological Theory of the Didactic (ATD), which provides a model for apprehending the various elements of mathematical activity that occur in any context (in this case, professional and educational). According to the ATD, an individual's "practices" are understood through the notion of "praxeology", which takes into account the know-how (the *praxis*) and the discourses (the *logos*) that describe, legitimise, explain, and produce the praxis. The praxis contains two components: *tasks* (things to accomplish) and *techniques* (methods used to accomplish the tasks). Similarly, the logos can be divided into several levels: Chevallard (1998) distinguishes between *technologies* (justifications for the techniques) and *theories* (foundations underlying the technologies). Following the example of Artigue (2002), we have chosen to avoid the ambiguity of the word "technology" and label any *logos* as a "justification". We will also classify different types of justifications, such as those that are *pragmatic* (concerning the productive potential of a technique) and those that are *epistemic* (concerning the potential of a technique to contribute to the understanding of the objects involved). As Artigue (2002) suggests, the place of certain techniques within an institution could depend on such justification types.

This leads us to another crucial aspect of the ATD: individuals' practices are framed by the social institutions where they live. Under the influence of institutional constraints, certain practices are normalized. Conversely, an increased acceptance of new practices can cause a reevaluation of constraints. To describe the variable status of practices within institutions, we use Morrissette's (2011) characterization of

1. *Shared Practices*, which are intimately tied to a profession and remain unquestioned;
2. *Admitted Practices*, which are not shared by everyone in a profession, but are accepted because they have been shown to be effective by innovative practitioners; and

3. *Contested Practices*, which are not accepted by everyone and are therefore situated at the boundaries of the professional culture.

METHODOLOGICAL CONSIDERATIONS

To gain a deeper understanding of how programming is integrated into the practices of mathematicians and their students, we carried out an exploratory study involving individual semi-directed interviews with 14 mathematicians: 3 women and 11 men of various ages, languages (French or English), and research domains, working within 10 universities in 3 Canadian provinces (British Columbia, Ontario, and Québec).

The structure of our interviews, captured in a written guide, was largely inspired by Vermersch's (2006) *entretien d'explicitation*. This model considers the actions of an individual as the main source of reliable information regarding the reasoning involved in those actions (different from the reasoning adopted outside of the actions). Hence, the interviewer is mainly a guide who tries to lead the interviewee into a state of descriptive verbalisation, where they "relive" specific experiences. Our participants were encouraged to relive moments throughout their research and teaching, in relation to computer programming. Some chose to share resources they had developed (e.g., computer programs and activity outlines), which enhanced their descriptions. Nevertheless, some general reflections on mathematics, programming, and institutional constraints were also solicited from the participants.

Interviews were recorded and transcribed. We then performed a categorical analysis (Van der Maren, 1996), using a mixed coding approach to identify, classify, and compare the main ideas. Examples of programming use underwent a supplementary characterisation using Chevallard's (1998) framework in order to extract the types of tasks, techniques, and justifications that define research and learning practices.

CLARIFICATION OF A DEFINITION

At the beginning of our project, we were surprised at how difficult it was to develop a definition of "computer programming". First of all, there was no clear agreement on what constitutes "programming" within the literature we had read. Secondly, we could not decide which kinds of activities to consider in our definition. It was clear that numerically solving a system of differential equations by developing a new method, writing some code, and ensuring the correct functioning of the resulting program should be seen as involving "programming"; but which steps exactly? Additionally, how should we classify activities such as the use of a computer algebra system to calculate a particularly difficult integral? Previous studies (e.g., Buteau et al., 2014) have distinguished between computer algebra systems and programming languages; but we can just as easily write a program in *Mathematica* as in *C++*.

In hopes that our participants could help us circumscribe "programming" in the context of mathematics, we left the interpretation of this word up to them. From the perceptions that emerged, we identify programming as an activity that aims to

construct a computer tool (a program) by way of three nonlinear tasks of varying importance: the development of an algorithm, the coding of the algorithm, and the verification and validation of the program. Still, there was no unanimity on the sorts of activities that correspond to the completion of these tasks. While some questioned whether or not using library routines in *Mathematica* really was "programming", others proudly described examples of this type that allowed them to make major advances in their research. At times interviewees would take an even larger perspective, wondering, for example, whether or not geometric constructions in *Geometer's Sketchpad* could be classified as "programming". Other times, they were more restrictive, reducing programming to the task of writing code. While most participants recognized a mathematical character in programming, comparing it to solving a problem or constructing a proof, they also tended to restrict the whole activity to the status of a technique for accomplishing more important research tasks.

THE PLACE OF PROGRAMMING IN MATHEMATICAL PRACTICES

Based on the experiences described by our participants, we characterized several research and learning practices that may involve programming. What was particularly interesting was the varying degree to which programming (as we have just defined it) could be involved. Our analysis led us to identify six levels on which mathematicians or students may interact with programming: they may

L0: Strictly observe the results of a computer program;

L1: Manipulate the interface of a program;

L2: Observe the code of a program;

L3: Modify the code of a program;

L4: Construct the code of a program, with elements (e.g., the algorithm) specified; or

L5: Develop a program, including algorithm development, coding, and verification.

The higher the level, the more the programming activity becomes visible and the more the mathematician or student becomes active in it. The identification of these levels allowed us to make important comparisons between the practices of a given mathematician and those intended to be developed by his/her students. In what follows, I restrict myself to an insightful selection of these practices, organized into two naturally-arising categories: "pure" and "applied" problem solving.

"Pure" problem solving and the cases of Omar and Paul

This category refers to when a mathematician or mathematics student seeks to develop abstract mathematical knowledge, the former contributing directly to the discipline and the latter supporting the personal education of the student. In both contexts, a main type of task where programming may be involved is the discovery of concepts, properties, or theories, typically realized through an *Exploration Cycle* including the observation of mathematical objects and the formulation/verification of

conjectures. Our study shows that the place of programming within the associated praxeologies may differ significantly for mathematicians and their students.

	Task Type: Discover mathematical concepts, properties, or theories	
	Technique	**Justification**
Omar	L5 + *Exploration Cycle* until "sufficient" evidence collected	(1) *Epistemic*: Gain insight to formulate conjectures and confidence to proceed to proof (2) *Pragmatic*: Organization, precision, speed, adaptability, and reusability of programs
Omar's Students	L0 + *Guided/ Limited Exploration Cycle* in class	*Epistemic*: Have intuition challenged and abstract theory rendered concrete, exciting, and memorable *Pragmatic* as above, but from professor's perspective

Table 1: Omar's research praxeology vs. the praxeology proposed to his students.

Like several of the pure mathematicians I interviewed, Omar often develops his own computer tools (L5) to collect evidence about the behaviour of the abstract objects he studies. He proceeds through an *Exploration Cycle* to first gain the insight necessary to formulate plausible conjectures and then build confidence in their truth. He justifies his technique principally in an *epistemic* fashion, exclaiming, for example, "Before starting to prove something, you'd better know it's true beyond a doubt!" Nonetheless, the *pragmatic* character of his programming is undeniable: he is free to control every aspect of his exploration, extend it to any number of otherwise tedious or impossible examples, and adapt his tool for completing future research tasks.

Given his familiarity with creating programs to assist in his own discovery of mathematics, it is not surprising that Omar, like many of our participants, also develops tools within the context of his teaching to support his students' understanding of challenging notions (e.g., spanning sets and linear independence). From the student's point of view, however, the proposed praxeology is quite different from their professor's: Omar's linear algebra students are invited solely to observe the dynamic images produced by their professor (L0), are prompted to make conjectures, and are provided images that verify or refute their voiced conjectures. Their exploration is heavily guided and limited to the time available in class, and the programming activity remains completely inaccessible to them. The fact that Omar does not share his programs with his students parallels the way he (and other pure mathematicians) communicate their research results: once they have arrived at a theorem and its proof, they typically see no need to discuss the programming that assisted their exploratory work. Similarly in teaching, Omar sees no need to encourage further exploration with a program once he believes the main goal of student discovery has been achieved. Indeed, he emphasizes the *epistemic* quality of the proposed technique, claiming that observing carefully chosen computer-generated results enables students to have their intuitions challenged and the abstract

theories they're learning rendered more concrete, interesting, and memorable. When he attributes a *pragmatic* value to the technique, it is in relation to himself: the examples he generates would be difficult, if not impossible, to reproduce on a board, and he would not have the same flexibility of re-executing programs in response to the needs of his students. A summary of Omar's research praxeology and the counterpart praxeology he proposes to his students is given in Table 1.

Paul is a probability professor whose table of praxeologies would differ from Omar's table in terms of techniques and justifications. In his research projects that require the use of computer tools, he always remains at L0 or L1; the programming is done by a collaborator. And yet, he encourages his students to write and use their own programs (L4 or L5). Like Omar, Paul brings forth principally *epistemic* justifications; the difference is his claim that

> It's much better if the students can program it for themselves. If they're sitting in front of the screen and they can play with it and they can adjust parameters, it becomes a kind of a game and it's more interactive for them. And it's better than me just showing them a picture at the front of the classroom.

We will return to this idea that higher-level interactions may have greater *epistemic* value. For now, this claim naturally raises the question: so, why doesn't Omar invite his students to develop their own programs?

Ironically, it is Paul who provides an enlightening story for framing the response to this question. It turns out that the probability course described above is geared towards science students; for math majors, computer technology is completely absent. Of course, there are many ways to "Discover mathematical concepts, properties, or theories", and Paul's pure math students are encouraged to adopt more traditional techniques. Upon reflecting on the addition of programming, Paul concluded: "I think it's the right way to go actually. I think that we're missing an opportunity here." So, why not take the opportunity? At first, Paul discussed curricular constraints: the pure math students may not have the prerequisite knowledge needed for programming and it would take a great deal of time to develop and integrate new activities into an already jam-packed well-defined curriculum. Omar also explained that "There's so much material in [his] course that it seems like it would be an exaggeration to ask them to program as well […] But, if we had more time, well it would be nice." Since Paul already overcame curricular obstacles during the transformation of the probability course for science students, it would seem like something deeper is at play. Indeed, he eventually revealed that "Academia is a very conservative place. And there's a huge amount of inertia. And there's also a huge amount of independence among the different instructors." He added: "I don't hear a lot of people talking about this being a great idea." Omar elaborated on similar constraints imposed within his institution: "I realize that my department is very abstract […] And the students in pure math love that. But I believe it limits some of their abilities that are absolutely essential if they want to become researchers."

As reflected in this quote, the pure mathematicians in our study view programming (L5) as *admitted* amongst them and their colleagues. Phillippe summarizes their perspectives when he says that "Programming is really one tool among many others to do mathematics […] that is not necessary, but that is useful." Why then is programming still deemed by some departments as *contested* for pure math students? In the past, some mathematicians (e.g., Bailey & Borwein, 2005) have implied that computer-based techniques were *contested* within the pure math community. Our participants point out that many Canadian universities are still anchored in this traditional culture that favors the chalk-and-talk paradigm and by-hand exercises.

"Applied" problem solving and the cases of Barbara and Ben

The institutionally-driven introduction of programming in Paul's probability course for science students is likely related to the importance attributed to programming in the "applied" math community. As Barbara suggests, "It's absolutely indispensable for applied mathematicians." When it comes to solving "real-world" problems, programming is part of the techniques *shared* by all of our participants, allowing them to analyse data (to develop/validate mathematical models), calculate parameters (to specify such models to particular situations), and understand mathematical models (either for validation purposes or to describe, explain, or predict real-world phenomena). As above, I elaborate on the praxeologies for one (the last) type of task.

According to our applied participants, whenever they must explore the behaviour of the mathematical models they develop and/or study, programming (L5) is simply a necessity for *pragmatic* reasons: not only does it create tools capable of performing a massive number of calculations and varying parameters to consider different scenarios, but first and foremost it permits the simulation of models that lack analytic solutions. As Alice explains, "It's highly unlikely that a mathematical model will give you the quadratic formula in the end. It would be nice, but that doesn't happen. And so, computer programming is essential." Though it was not emphasized by the applied group, the underlying *epistemic* character of programming is also clear: it is the visual and dynamic output generated by computer programs that enables the recognition of patterns leading to descriptions, explanations, or predictions.

Given their *pragmatic* justifications, it is not surprising that all the applied researchers engage in programming at the highest level: L5. It may also not come as a surprise that we observed the least dramatic differences between the place of programming in research and in learning within this category. Still, there were some notable differences and interesting debates. Barbara's students, for example, are not asked to develop their own programs. Instead, in addition to observing some results shown by their professor in class (L0), they are invited to receive explanations of her code (L2), manipulate her programs at the interface level (L1), and modify them (L3) to analyse different models. Barbara explains that "[she] want[s] [students] to see that programming isn't that bad. You can do lots of interesting stuff with just a few lines of code." Through the proposed techniques, her students may learn more about

programming itself (e.g., syntax and structure), and may come to appreciate the computer as a powerful tool. Having some insight into the code may also support their understanding of the corresponding output and models. Nevertheless, Barbara justifies their mid-level interactions by saying things like, "It wasn't so much how to program a vector field, it was how to use a vector field to understand the model." Her ultimate goal is for students to understand models, not necessarily programming.

<table>
<tr><td></td><td colspan="2">Task Type: Understand the behaviour of a mathematical model</td></tr>
<tr><td></td><td>Technique</td><td>Justification</td></tr>
<tr><td>Ben</td><td rowspan="2">L5 + Experimentation (i.e., variation of parameters to observe different output)</td><td>(1) Pragmatic: Otherwise impossible due to lack of analytic solutions and number of calculations/scenarios to consider
(2) Epistemic: Visual/dynamic output for various parameter values enables descriptions, explanations, and predictions</td></tr>
<tr><td>Ben's Students</td><td>(1) Pragmatic and (2) Epistemic as above, plus: L5 leads to deep understanding and control of the tool, output, and model</td></tr>
</table>

Table 2: Ben's research praxeology vs. the praxeology proposed to his students.

In comparison, Ben believes that inviting students to do the programming (L5) might have a higher *epistemic* value. On the one hand, he suggests that "It's very hard to write a program and not understand what it's doing. You know, it's a different level of comprehension." On the other, he reflects on his experience asking students to manipulate a pre-developed program (L3): "It was an exercise in typing. They really didn't know what it was doing or why it was doing it." In Ben's view, if students write their own programs, it is more likely that they will deeply understand the tool, enabling them to modify it according to their needs, more effectively interpret the results, and, by extension, better understand the models. Other mathematicians add that while constructing a program, students may come to better grasp the concepts, processes, and methods that they must structure into an algorithm and transpose into a programming language. Then, having created their own tool, students may feel a sense of empowerment and excitement that may further enhance their engagement and understanding. And finally, students may also gain more insight into elements of programming itself (e.g., algorithms, data structures, code efficiency) that could not only allow them to better understand and use existing software (previously "black boxes"), but also provide them with the knowledge required to develop their own tools in the future. After all, the more the power of programming is shifted into the hands of students, the more they may be convinced of the *pragmatic* value of such techniques. In sum, many mathematicians agree with Paul that "It's much better if the students can program it for themselves."

Once again, we may wonder: why doesn't Barbara ask her students to develop their own programs? Throughout her interview, the professor complained that her university lacks a mandatory training in programming for math students and that the

activity is not widely implemented by her colleagues; some of her students are even afraid of programming! In contrast, learning and using programming is integrated throughout the curricula for all math students in Ben's department. But, as Ben explains, this systematic institutionalization of programming is not necessarily easy:

> There's a lot of inertia in Universities. [...] You don't just introduce something and it happens. [...] You introduce it one year, and everybody talks about it, and it's a no. And then there's lots of conversations about it [...] because you want people to have something that they truly need, and that has to evolve through discussion.

Moreover, even after all the discussion, the institutional context may impose serious constraints. Alice, for example, works at a university where programming-based techniques are completely normalized. Yet, she feels she must settle for encouraging lower-level interactions (L4 or L3) because she does not have enough human resources to adequately grade students' code; and in her view, "If it's not assessed in detail, the requirement is shallow." In the end, while programming (L5) may be part of the *shared* practices of applied mathematicians, institutions may render it only *admitted* within the community of students in applied math courses.

SUMMARY AND CONCLUSIONS

In 2014, Buteau et al. reported an intriguing gap: Of 302 mathematicians, 43% claimed to use programming in their research and only 18% said they use the activity in their teaching. In this paper, I presented some results of a qualitative study that sought to gain a deeper understanding of this situation. Our analysis of interviews with 14 mathematicians shows that "using programming" may be interpreted in significantly different ways. The word "programming" itself does not have the same meaning for every mathematician and future research could benefit from clarifying the boundaries of this activity. But in addition to this, when professors "use programming", their students may actually interact with the activity on various levels, from strictly observing the results of programs (L0) to independently developing their own computer tools (L5). The identification of six levels led us to note important differences between the practices of individual mathematicians and those they propose to their students, suggesting that the gap highlighted by Buteau et al. (2014) may actually be greater and more complex than expected. Even if programming (L5) may be *shared* or *admitted* within applied or pure research communities, respectively, it may be *admitted* or *contested* within applied or pure learning communities. Adding to previous predictions as to why such gaps might occur, our participants spoke of different kinds of institutional constraints: curricular (objectives, prerequisites, time), departmental (academic freedom vs. coordination, class size vs. human resources), and cultural (deep-rooted traditions in mathematics). And yet, they spoke equally of the potential benefits of bridging the gaps. Not only might it encourage techniques of high *epistemic* value or make students aware of the *pragmatic* character of programming, but it may also be important for *social/cultural*

reasons: Programming may widen students' vision and appreciation of all mathematical activity, while also encouraging the development of mathematician-like practices that could diversify their options beyond their undergraduate degree. Of course there is the question of the actual experiences of students, which we have not yet addressed: what benefits (and obstacles) do students *actually* experience while programming? Or, more critically, what are the benefits (and obstacles) of each programming level? After all, any level might be required in doing mathematics, whether due to collaborations (recall Paul's research practices) or the development and sharing of tools, which constitutes another category of practices we identified. Indeed, our participants mentioned observing their colleagues' results (L0), using other programs (L1), making sense of existing code (L2), or even reworking such code (L3). I hope to pursue a deeper analysis of these different levels in future research. In the meantime, it is important to note that all 14 of our participants believe that while programming should not constitute the essential element of undergraduate mathematics, it should receive more attention than it does in current Canadian curricula. A path towards change may not yet be clear, nor may it be easy; but the shared perspectives of our participants lead us to conclude that it exists!

REFERENCES

Artigue, M. (2002). Learning mathematics in a CAS environment: The genesis of a reflection about instrumentation and the dialectics between technical and conceptual work. *International Journal of Computers for Mathematical Learning, 7*, 245-274.

Bailey, D.H. & Borwein, J.M. (2005). Experimental mathematics: Examples, methods and implications. *Notices of the AMS, 52*(5), 502-514.

Broley, L. (2015). *La programmation informatique dans la recherche et la formation en mathématiques au niveau universitaire* (Unpublished master's thesis). Université de Montréal, Montréal. Supervised by F. Caron and Y. Saint-Aubin.

Buteau, C., Jarvis, D., & Lavicza, Z. (2014). On the integration of computer algebra systems (CAS) by Canadian mathematicians: Results of a national survey. *Canadian Journal of Science, Mathematics, & Technology Education, 14*(1), 1-23.

Chevallard, Y. (1998). *Analyse des pratiques enseignantes et didactique des mathématiques : L'approche anthropologique*. Retrieved from yves.chevallard.free.fr/spip/spip/IMG/pdf/Analyse_des_pratiques_enseignantes.pdf

Morrissette, J. (2011). Vers un cadre d'analyse interactionniste des pratiques professionnelles. *Recherches qualitatives, 30*(1), 10-32.

Van der Maren, J. (1996). Le codage et le traitement des données. In *Méthodes de recherche pour l'éducation* (pp. 427-457). Montréal/Bruxelles: PUM et de Boeck.

Vermersch, P. (2006). *L'entretien d'explicitation*. Paris, France: ESF éditeur.