Some Thoughts on the Teaching of Mathematics—Ten Years Later

Igor Rivin

This is a somewhat expanded and corrected version of a "manifesto" first posted on my Temple University webpage almost exactly ten years ago. Since then, I have put some of the ideas expressed below into practice—a brief description of my experience is included in the section "My Experience over the Last Decade"—and I have also had a fair amount of feedback. Some (most) of it has been completely positive; other feedback has included some critical ideas. I describe some of it in the section "Some Feedback".

What Is the Problem?

A mathematics professor in a public university has many responsibilities. These include research, administration, and teaching. Teaching, in turn, includes "specialized" teaching (to mathematics majors and graduate students) and "service" teaching: teaching mathematics to first and second year students. These thoughts will center primarily on service teaching which, for me, combines some of the most exciting and some of the most depressing aspects of my job. Some thoughts on teaching mathematics majors are added in the section "What Can We Do?".

The Product

Why depressing? Consider the following: the vast majority of service courses are concerned with differential and integral calculus and linear algebra. These are both rather deep subjects, as evidenced by the fact that mathematics had been practiced for thousands of years by rather talented people before the basic principles of the calculus were laid down in the late seventeenth century (although some of these principles were discovered, in an *ad hoc* way, by Archimedes—considerably earlier). It took another two hundred years of extensive work to make the foundations of the subject truly solid. Linear algebra, as used today, is an even later bloomer. The current machinery of matrices and linear transformations was not put into a truly modern form until the beginning of the twentieth century. We have no choice but to agree that these subjects are quite deep and require some considerable technical skill to use successfully.

The Consumers

Who are we teaching them to? In a public university (such as Temple) our students are, in the main, somewhat above average products of the U.S. public secondary education system. This means that their technical ability is already quite severely taxed by arithmetic with fractions. Their abstract reasoning skills are essentially nonexistent and the very concept of proof is foreign to them.

The Results

A consequence of all this is that it is well-nigh *impossible* to teach them what we purport to teach them: higher mathematics presupposes a certain level of abstraction, and even if we commit the crime of forgetting that and define calculus as "a collection of computational techniques without understanding," the students' technical weakness renders even that aspect essentially worthless. They *cannot* compute. The result is that our calculus and linear algebra classes consist of a collection of trivial examples, which the students must memorize by rote. This has the consequence of not teaching the students anything except the fear and hatred of mathematics. There is more still: the majority of the students never use calculus in their future lives (small wonder since they don't actually know any, as discussed above), but they never had any intention of using advanced

Igor Rivin is professor of mathematics at Temple University and ICERM visiting professor at Brown University. His email address is igor.rivin@temple.edu.

The author was supported in part by the NSF DMS when the original version was written and is now supported by ICERM under its block NSF grant. The author would like to thank the many people who commented on the previous version of this document. He would like to thank the anonymous referees for helpful suggestions. He would like to thank Brown University and ICERM for their generous support.

DOI: http://dx.doi.org/10.1090/noti1131

mathematics even *before* taking the courses. They are required to take the courses because of the (not unreasonable) belief that mathematics should be a part of every college-educated person's intellectual makeup. The result is that the *loathing* of mathematics is part of the intellectual make-up of a sizeable majority of Americans. The amazing (and exhilarating) observation is that, despite all of the above, some students actually manage to understand something of the subject. This exhilaration is, however, tempered by thoughts of the huge amount of wasted time and by the thought of what these talented students could achieve if taught properly.[1]

What Can We Do?

What, then, is the solution? We could drop the distribution requirements in mathematics (and I could easily see this happening), but the fact of the matter is that the ability to reason logically and abstractly really should be something (perhaps the *main* thing) everyone takes from his or her higher education and something we, as mathematics educators (which we are, even if the term does produce a visceral reaction in most people), ought to instill in our students. How? The first step is a step back—a step back from "higher" mathematics, the mathematics of the infinite and the infinitesimal—back to conceptually simpler forms of mathematics.

In the (not so distant) past, Euclidean geometry was such a subject, taught exactly for the above-stated reasons (the fundamental concepts—of line, circle, distance, etc.—are quite intuitive, while the basic components of mathematical reasoning are all present). I would not necessarily advocate a return to this, however. Firstly, the subject has been dead for several hundred years, and secondly, it is quite far from the modern American experience.

Instead, it is my opinion that we should start at the very beginning—with reasoning (logic) and counting (which means naive set theory and combinatorics and graph theory) and probability. These subjects are ever more visibly important in our lives due to the ubiquity of computers. They are both easier to learn and more immediately rewarding for the students than what we are currently teaching. In addition, there is another principle which can be used to clear at least some of the mush out of the students' minds. That principle is

(1) A COMPUTER PROGRAM *IS* A PROOF.

For example, the student who wrote the program in the section "Source Code for a Modular Arithmetic Package in Scheme" had to have complete understanding of the Chinese Remainder Theorem, and his program *is* the Chinese Remainder Theorem in that, given some quantities satisfying the hypotheses of the theorem, it never fails to produce a quantity satisfying the conclusion.

This correspondence between program and proof (though far from perfect) allows us to make mathematics hands on (programming a computer gives very rapid feedback, both positive and negative); it is closely related to the students' experience and visibly "useful". (Of course, the real utility of mathematics lies much deeper, but ….)

One problem with this approach is the potential need to waste a lot of time introducing students to the subtleties and idiosyncrasies of some (possibly proprietary) programming language or a scientific computing system. It is important to start the students off on a mathematically clean system, preferably running on a mathematical *abstract machine*. Luckily, such a system is available and had been used for over thirty years in computer science education at MIT, the University of Indiana, and many other schools. This is the Scheme programming language.[2] The justly acclaimed book *Structure and Interpretation of Computer Programs* [1] by H. Abelson and G. J. Sussman introduces the fundamentals of computer programming and, together with a companion book on *Structure and Interpretation of Mathematics*,[3] this would constitute the core of a modern introduction to mathematics. It should be noted that Scheme is no longer used as the introduction to computing at MIT, having been supplanted by Python, but the reason for this is that Python is a more useful tool for practical work (having libraries for almost any task one might need to accomplish) and has absorbed much of the Scheme semantics. It is, however, worse as a first language because it is less pure and has less well-defined semantics.

Given the very high level of functionality presented by Python, Mathematica, or other "kitchen sink" languages, it may be a reasonable compromise to start with Scheme as a way of building a foundation and then proceeding to use one of these languages when (more precisely, when and if) more advanced topics in mathematics need to be introduced.

[1] *A typical cafe conversation usually starts with:*
NONMATHEMATICIAN: *What do you do?*
MATHEMATICIAN: *I study mathematics.*
NONMATHEMATICIAN *(one of these two responses) I am terrible at math OR I used to love math!*
This is a very American phenomenon; I have never had such a conversation in Europe.

[2] *See* `http://scheme.org` *for more details.*

[3] *Writing such a book is a project I am very keen on, but this has to be done in parallel with using these ideas in teaching—the book of Abelson and Sussman was circulated as lecture notes for several years.*

Possible Objections

One could foresee some objections to the above program:

But What about Calculus?

There is no suggestion of eliminating calculus completely from the university curriculum: students of the sciences and engineering (not to mention mathematics) do need to be acquainted with it. However, it would enter somewhat later, when the students are more mathematically mature and thus more capable of actually understanding something of the subject. It is true that a lot fewer students will be *required* to study calculus but, on the other hand, the number of people studying mathematics may actually increase.

But What about the Teachers?

It has been my observation that many of the TAs, having been educated in a "traditional" way, are a little rusty on "finite" mathematics. The same is true (in spades) for many of the professors. Many faculty members and graduate students have no familiarity with computer programming at all. The first problem is easily fixable, the second slightly less so, but my contention is that anything that is teachable to freshmen should be even easier to teach to faculty (and is no less useful to them).

My Experience over the Last Decade

Since the first version of this document was published, I have taught a number of relevant courses (with multiplicity). One was Junior Problem Solving, which is an introductory course in mathematics for computer science majors. Another was Mathematical Patterns, which is a sort of a Math for Poets course. A third was Senior Problem Solving, which is (in principle) the last mathematics course a mathematics major at Temple takes. The fourth was Mathematical Computing—a course for mathematics graduate students.

Poets

In the first two courses (Junior Problem Solving and Mathematical Patterns) I taught the basics of logic, as presented in the very nice book [2] by Harry Gensler, who is an ethical philosopher, not a mathematician. This means, in particular, that while the book eventually goes into symbolic logic, even there a lot of the questions are about deciding validity of English sentences or (more generally) philosophical arguments. In both instances, the course was a success in that, while, as far as I could tell, *none* of the students could carry out a logical argument at the start of the course, *many* could at the end (the fact that they could not at the beginning validates my belief that the students were not ready for anything resembling calculus). Some additional remarks:

- At one point in the Junior Problem Solving course, I decided to try something more "mathematical", and presented the proof that $\sqrt{2}$ is irrational. Despite going very slowly, it was quite clear that the arithmetic involved was too much for the students.
- Many of the students thanked me at the end of the course (in one case, I was walking down the street in Center City, Philadelphia, when one of my former students crossed the street, dodging a number of fast-moving cars, to tell me how much she appreciated the material). This sort of thing had never happened at the conclusion of any calculus course I had taught before.

Budding Mathematicians

In the graduate course, I started by teaching the elements of programming à la Abelson-Sussman, with more emphasis on mathematical problems. The experience was definitely bimodal. The good students (by which I mean ones having a lot of mathematical potential) had extremely good programming skills, while the not-so-good students had essentially none. This, of course, is quite unfortunate since most of Temple's Ph.D. candidates would wind up in industry, which means that programming skills are essential to their future livelihood—the best students are the ones most likely to pursue a career in pure mathematics and, in that sense, *need* programming skills the least (although, as Gauss already knew well, mathematics is an experimental science, and computing is the experiment).

The Senior Problem Solving course was, in a way, the most disappointing—while some of the students were extremely good, most did not have much better technical skills than the computer science students in Junior Problem Solving (to make it more depressing, these were mostly mathematics education majors) and would tend to get just as confused by proofs, especially ones which required computation. I am quite sure that had they been required to take a course of the kind I was teaching to their poetical brethren, together with a rigorous algebra course (of the sort those of us who grew up in the Soviet Union took at the age of twelve), they would have been much more deserving of a mathematics degree. However, by the time I got them, it was already too late.

Source Code for a Modular Arithmetic Package in Scheme

The code below (taken verbatim from a homework assignment done by a student in one of my classes) defines all of modular arithmetic in Scheme (which

already has most of it, so such a program would be a lot shorter and more efficient in practice). At the end, an implementation of the extended Euclidean algorithm is given, followed by an implementation of the Chinese Remainder Theorem algorithm.

```
(define (intMod k m)
  (if (= m 0)
      (lambda (s) 0)
      (let* ((j (remainder k m))
             (l (if (< j 0)
                    (+ j m)
                    j)))
        (lambda (s)
          (if s
              l
              m)))))

(define (sameMod? x y)
  (= (x #f) (y #f)))

(define (eqM? x y)
  (and (= (x #f) (y #f))
       (= (x #t) (y #t))))

(define (displayM x)
  (display (x #t))
  (display " mod ")
  (display (x #f))
  (display "\n"))

(define (+M x y)
  (if (sameMod? x y)
      (intMod (+ (x #t) (y #t)) (x #f))
      (intMod 0 0)))

(define (-M x y)
  (if (sameMod? x y)
      (intMod (- (x #t) (y #t)) (x #f))
      (intMod 0 0)))

(define (*M x y)
  (if (sameMod? x y)
      (intMod (* (x #t) (y #t)) (x #f))
      (intMod 0 0)))

(define (Modulus x)
  (x #f))
```

```
(define (Coset x)
  (x #t))

(define (gcde a b)
  (letrec ((aux (lambda (x1 x2 y1 y2 r1 r2)
                  (let ((r3 (remainder r1 r2)) (q (quotient r1 r2)))
                    (if (= r3 0)
                        (list r2 x2 y2)
                        (aux x2
                             (- x1 (* q x2))
                             y2
                             (- y1 (* q y2))
                             r2
                             r3))))))
    (aux 1 0 0 1 a b)))

(define (/M x y)
  (let ((z (apply gcde (list (y #t) (y #f)))))
  (if (and (sameMod? x y) (= 0 (remainder (x #t) (car z))))
      (intMod (* (cadr z) (/ (x #t) (car z))) (x #f))
      (intMod 0 0))))


(define (crt l)
  (if (null? (cdr l))
      (car l)
      (let* ((w (map Coset (list (cadr l) (car l))))
             (x (map Modulus (list (car l) (cadr l))))
             (y (apply gcde x))
             (z (car y)))
        (if (apply = (map remainder w (list z z)))
            (crt (cons
                  (intMod (/ (apply + (map * w x (cdr y))) z)
                          (/ (apply * x) z))
                  (cddr l)))
            (intMod 0 0)))))
```

Some Feedback

Most feedback has been overwhelmingly positive. Here are some examples and some comments of my own ([IR]):

IR: Some have suggested probability as a good introduction to discrete math. Ward and Gundlach (of Purdue) have written a book for the purpose.

- I read your diatribe and found it very compelling—in fact, something that I agree with and find in accordance with my own experience as an undergraduate (an electrical engineering major). It wasn't until I took a discrete mathematics course in my sophomore year that I began to understand mathematics. I'm now at Caltech studying applied math, so to say that discrete math was the start of an enormous change in my career path would not be an understatement.

  At the same time, you will experience a large number of engineers who feel that a lack of, say, vector calculus in three dimensions will seriously hinder students in (say) electromagnetic engineering. I found this to be the case even though I had taken the course and ostensibly "knew" the material.

[IR]: Of course, I certainly agree that engineers should study multivariate calculus, but not as the first thing.

IR: It was suggested that the "integrated approach" (where mathematics and the science that uses it are used in parallel) is a solution to some of the problems I am trying to address. My view on this is that, while this is an excellent idea, it cannot also be done in parallel to teaching people to think.

- (from a chemist at Temple):

  Hey, I was reading the first part of your diatribe on mathematics education and so far I agree with everything you said. I feel the same way about teaching organic chemistry—I feel like the course is more about identifying the 5% of the population that can handle organic chemistry than teaching.

- (from an applied mathematician friend):

  So I ask you:

  What is the value of an education system which starts at "the very beginning" at eighteen years old ?

  [IR]: My answer is: Very little.

  So. Can we do better? The answer is *yes*. Google Montessori. Both of my children spent from age three to age twelve in Montessori schools. When I try to explain any logical arguments to them, they look at me as if I am a complete moron because by age twelve in a Montessori system logical reasoning is natural.

References

[1] Harold Abelson and Gerald Jay Sussman, *Structure and Interpretation of Computer Programs*, Taylor & Francis.

[2] Harry J Gensler, *Introduction to Logic*, Routledge, 2010.