TEACHING INQUIRY THROUGH EXPERIMENTAL MATHEMATICS

Lara Pudwell
Department of Mathematics and Statistics
Valparaiso Univeristy
Valparaiso, IN 46383, USA
Lara.Pudwell@valpo.edu

TEACHING INQUIRY THROUGH EXPERIMENTAL MATHEMATICS

Abstract: In this paper, we discuss the Experimental Mathematics course taught at Valparaiso University since 2009. We focus on aspects of the course that facilitate students' abilities to ask and explore their own research questions.

Keywords: experimental mathematics, computational math, inquiry, programming

1 INTRODUCTION

Experimental mathematics has gained increasing attention in the past two decades, as evidenced by a series of books [1, 2, 3, 4], a research journal [7] founded in 1992, and even the presence of the Institute for Computational and Experimental Research in Mathematics (ICERM), founded at Brown University in 2011 and initially funded by the National Science Foundation. Certainly, mathematicians have always used experimentation to make conjectures, but in recent decades, the computer has played an increasingly visible role in developing mathematical proofs, sometimes helping provide insights that were not apparent by pencil-and-paper-computation.

Jonathan Borwein and Keith Devlin [4] write that "Experimental Mathematics is the use of a computer to run computations, to look for patterns, to identify particular numbers and sequences, and to gather evidence in support of specific mathematical assertions that may themselves arise by computational means." This is certainly only one possible definition. Views of experimental mathematics in the research community range from using the computer as "paper-and-pencil with power steering" to using the computer as a coauthor capable of writing its own conjectures and proofs.

Regardless of personal philosophy, a course in experimental mathematics offers an opportunity to (i) teach basic programming, (ii) consider the history of proofs that have used computation in nontrivial ways, (iii) challenge students' personal philosophy of mathematics, (iv) explore a variety of problems that appear in disparate areas of the undergraduate curriculum, if they appear there at all, and, most importantly (v) teach students to explore their own questions. A course in experimental mathematics is a venue for inquiry-based learning in its most fundamental form. Such a course could effectively be renamed Mathematical Inquiry, since the focus of an experimental mathematics course is not on a particular list of content, but on a methodology for approaching new problems. Of course there is no single way to teach an experimental methodology either. This paper discusses one such model, as taught at Valparaiso University. Other Experimental Mathematics courses have been taught at Dartmouth College, Grinnell College, Rutgers University, Tulane University, and more, and a growing number of colleges have Introduction to Research courses that may include computer experimentation. One such course for first-year students at Ithaca College is described in [5]. This paper describes such a course designed for upper level undergraduate students, with the goal that interested faculty can replicate or modify the key components to create their own experimental mathematics course. In it, we discuss specific ways to facilitate the transition from students predicting patterns for specific assigned problems to students generating their own questions and conjectures in a technology-oriented course.

The Experimental Mathematics course at Valparaiso University began in 2009 and runs every second spring. The course started as an "advanced topics" course, but was sufficiently well-received in its first two iterations, that it has now been assigned its own course number and is recommended as an elective for mathematics majors. The class typically meets for three one-hour sessions per week in a computer lab. Each student has access to a desktop computer on the perimeter of the room equipped with appropriate software for the class (e.g., `Maple`). There are also tables in the center of the room for group collaboration as well as a white board and a projection screen where computer work can be shared with the rest of the room. The capacity of the class is limited by the number of computers in the room, and the course has run with as many as 16 students. There are no experimental mathematics textbooks to choose from, so Valparaiso's course is partly based on problems chosen by the instructor and partly motivated by topics in Borwein and Devlin's *The Computer As Crucible* [4].

The course has several distinct components which will be discussed in subsequent sections of this paper. First, students spend an introductory period learning the basics of programming (Section 2). Then, for most of the semester, a new problem is introduced each class period, and students use the computer to explore the problem and make conjectures (Section 3). Finally, throughout the semester, each student adopts an individualized mathematical experiment and reports on their progress to the rest of the class at multiple points in the semester (Section 4). Together these components expose students to an interesting collection of mathematical material and hone student skills in computation and in mathematical investigation.

2 BEGINNING PROGRAMMING

The prerequisites for Valparaiso's Experimental Mathematics course are either to take a proof-based course *or* a programming course. In other words, it is expected that students have vocabulary for logical reasoning, but it is not expected that they have previous programming experience. In general, there is always one subset of students who are complete novices at programming, and there is always another subset of students who have a strong computational background. Any introductory programming lectures done as a class will inevitably be too fast for some students and too slow for others. Therefore, the first few meetings of the course are presented as a series of explore-at-your-own-pace exercises. Students are given problems in (i) arithmetic and algebra, (ii) calculus and graphing, (iii) lists, sets, and programming, and (iv) more programming.

Each exercise set contains a sampling of problems that expose students to commands that will be used frequently in class and ends with some open-ended applications of the new commands. Students who entered the course comfortable with their computational and programming skills are enabled to move quickly towards the open-ended material. Some sample explorations include:

- Find a 10 digit prime number using `Maple`. This number must have **exactly** 10 digits.
- It is well known that the sum of the first n integers is $\sum_{i=1}^{n} i = \binom{n+1}{2} = \frac{1}{2}n(n+1) = \frac{1}{2}n^2 + \frac{1}{2}n$, i.e. the sum of the integers $1 + 2 + \cdots + n$ can be written as polynomial with variable n. Compute the polynomials $\sum_{i=1}^{n} i$, $\sum_{i=1}^{n} i^2$, $\sum_{i=1}^{n} i^3$, etc. in `Maple` and conjecture patterns for the following:
  - What is the leading term of $\sum_{i=1}^{n} i^k$?
  - What is the second leading term of $\sum_{i=1}^{n} i^k$?
  - What is the smallest exponent of n in $\sum_{i=1}^{n} i^k$?

By the end of these initial class meetings, students still have diverse backgrounds, but as a cohort they now have some common programming vocabulary to work with as they progress to new mathematical problems. Although Valparaiso's course has used `Maple` because Valparaiso owns a site license for the software; this work could easily be done in

`Mathematica`, `Sage`, or any other software including mathematical constructs. Most importantly, these initial lessons give students a chance to *explore* mathematical questions with the computer commands they have just learned. Rather than spoon-feeding details to the class, students explore mathematics in the way that mathematicians explore mathematics by exploring real (and sometimes unsolved) problems. They are encouraged to ask questions, look for new commands with the computer's help menu, and make conjectures about mathematical patterns from day one.

3 ASSIGNMENTS

Once students have some common programming vocabulary, the experimental portion of the course begins in earnest. For the majority of the semester, each class period begins with the instructor introducing a new mathematical problem and discussing it long enough to make sure students clearly understand the problem statement. Then, students are provided with a list of exploration exercises to complete individually or in groups. During the remainder of the class period, students work at the computer, discuss problems with classmates, or usually both. At the end of each class, students are given a week to produce reports of their work. These reports include relevant computer code, data generated by code, and an analysis of the data. Students are encouraged to present their work in well-written paragraphs that would be readable to a student not in the class, but with comparable mathematical background.

Whereas the experimental mathematics course at Ithaca College [5] introduces a new lab every two weeks, the Valparaiso experimental mathematics course introduces a new topic every one to two days. There are two reasons for this different pace. First, Valparaiso's course is designed for junior and senior majors who have more extensive background than a first semester student. Second, Valparaiso's course highlights experimentation across the curriculum. Problems come from combinatorics, number theory, algebra, analysis, geometry, and more, with an aim that students see experimentation as something that permeates the discipline of mathematics.

Here, we present two assignments that have been used successfully in previous iterations in the course, one from early in the semester, and one from later in the semester. These examples (a) highlight particular problems that are reasonable for novice programmers, and (b) illustrate the progression of experimentation skills cultivated in the course.

3.1 THE COLLATZ PROBLEM

One exercise, from early in the semester explores the Collatz conjecture. In particular, consider the function

$$f(n) = \begin{cases} 3n+1 & n \text{ odd} \\ \dfrac{n}{2} & n \text{ even} \end{cases}.$$

The Collatz conjecture claims that for any positive integer n, there is a positive integer k such that $f^k(n) = 1$. For example, if $n = 10$, then $f(10) = 5$, $f^2(10) = 16$, $f^3(10) = 8$, $f^4(10) = 4$, $f^5(10) = 2$, and $f^6(10) = 1$. It turns out that the smallest k such that $f^k(27) = 1$ is 111. Although the Collatz conjecture has been verified for $n \leq 19 \cdot 2^{58} \approx 5.48 \times 10^{18}$, there is no proof. In 1972, John Conway showed that a generalization of this problem is undecidable, but his proof does not apply to the original problem. Nonetheless, programming to explore the Collatz conjecture provides a natural context to explore recursive programming. Also, several nice generalizations of the Collatz problem provide areas that are fertile for cultivating student conjectures.

Together we write code for $f(n)$ (above),

$$g(n,k) := [f(n), f^2(n), \ldots, f^k(n)], \text{ and}$$

$$h(n) := [f(n), f^2(n), \ldots, 1].$$

Here, square braces are computer notation for a list. In other words, $g(n,k)$ produces k iterations of f with the initial value n, while $h(n)$ iterates f until $f^k(n) = 1$. For example, $g(10,2) = [5,16]$, while $h(10) = [5,16,8,4,2,1]$. If the Collatz conjecture is true, $h(n)$ should always terminate, whereas $g(n,k)$ gives the user control over how much work the computer should do.

After writing code together, students explore questions such as:

1. Verify that it takes 111 steps to get the answer 1 if you start with $n = 27$.
2. Try several other starting values for n.
3. Explore what happens when you try a $5n+1$ rule. How many steps does it take to get the answer 1 for $n = 1, 2, \ldots, 10$? (Some of your answers may be precise; others may be conjectures. Either way, support your answers with data!)
4. Edit your code so that it explores the following sequence: if n is prime then $f(n) = 3n+1$, and if n is composite, then $f(n) = n/p$, where p is the smallest prime that divides n. We want to see what happens when you apply $f(n)$ repeatedly. What happens when you start with $n = 2, \ldots, 10$?

5. Try inventing your own piecewise function $f(n)$ that we haven't considered yet. What interesting behaviors can you produce by repeatedly iterating your function?

The bulk of the programming happens as a class, but students execute the code and make appropriate edits for the new problems. Students are pushed to start with small steps to verify that they understand the initial problem well, and then they make conjectures about increasingly open-ended problems. Starting with a relatively straightforward problem statement, students can quickly use tools they have built to explore their own related questions.

3.2 INTEGER RELATIONS

Later in the semester, we visit the celebrated Borwein, Bailey, Plouffe formula for π given by

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left[\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right].$$

Unpacking this formula and why it is computationally important takes several days of class. One of the first parts of the discussion is "How did Borwein, Bailey, and Plouffe discover this formula?" The answer is that they took several other known formulas that included a sum with a $\frac{1}{16^k}$ and used an efficient integer-relation algorithm, known as PSLQ (Partial Sum of Least sQuares). While PSLQ goes beyond the scope of the course, understanding integer-relation algorithms does not. An integer relation for a list of 3 numbers a, b, and c, is three *integer* coefficients α, β, and γ such that $\alpha a+\beta b+\gamma c = 0$ and not all of α, β, and γ are equal to zero. The restriction to *integer* solutions requires different approaches than calculus optimization problems. An integer relation is not always possible for given numbers a, b, and c, but we could still ask "what set of integer coefficients produces a linear combination that gets as close to zero as possible?" With this understanding, students complete the following tasks:

1. Implement your own integer relation algorithm for 3 real numbers.
2. Compare your algorithm's results against another classmate's results and against PSLQ (implemented in `Maple`'s IntegerRelations library).
3. What properties of a, b, and c make it easier to find an exact integer relation? What properties make it more difficult? Are there any sets of irrational numbers where you can guarantee exact integer relations?

Students often try to write code that makes special assumptions on the kinds of values a, b, and c might be. For example, if a, b, and c are integers, the task is trivial. On the other hand, when pushed to consider three irrational numbers, students generally switch to a search through $-n \leq \alpha, \beta, \gamma \leq n$ for some input integer n. At the end of the exploration, students have a better appreciation for how efficient PSLQ really is.

3.3 BUILDING SKILLS OVER TIME

These two lessons, taken from different parts of the semester, illustrate problems for two different levels of student confidence. Each lesson includes three parts: (i) a problem statement, (ii) generating code for exploration, and (iii) using data computed via the code to make conjectures. Table 1 presents the planned trajectory of the course. Throughout the semester, the instructor brings problems to class that will help students grow their mathematical intuition and programming skills. At the beginning, as in the Collatz exercise, students are capable of programming with group input, but may feel lost if left to write such code completely independently; however, they are able to make conjectures about patterns they see in the computer output from the first day. As the semester proceeds, students take increasing ownership of their code, as in the integer relations exercises. By the end of the semester, students complete a final project, as described in Section 4, where they take ownership of all three aspects of a mathematical experiment, including formulating the problem statement.

	problem statement	**code**	**data**
early classes	in-class	in-class	independent
later classes	in-class	independent	independent
final project	independent	independent	independent

Table 1. Transitions in student skills

As an intermediate confidence-building exercise between assignments and projects, students complete two midterm exams during the course. The midterm exams explore problems that are new, but related to recent assignment problems. Students may need to modify previous code or generate new code. Each exam ends with a reflection essay, asking students to analyze how their approach to doing mathematics has (or has not) changed over time. Exams are given as independent takehome explorations and are due within a week.

Students who are used to viewing math as a quest for a “final” answer are sometimes taken aback by open-ended questions in an experimental course. One important recurring conversation is a reminder that experimental mathematics is focused not on a particular list of skills, but on an overarching approach for tackling new problems. This conversation starts early, when students learn to write code. Often, students who struggle with programming go immediately to the computer and get frustrated when their buggy code fails to produce the desired response. Breaking the job of writing code into two phases: careful planning (on paper) and execution (putting their paper-tested plan to work in the computer) helps students more easily pinpoint their programming issues. A similar analogy works with experimental exercises: if one focuses solely on execution to get a particular piece of data, she or he could bypass a lot of helpful insight. Thinking carefully about *why* you plan to do things raises more questions to explore and can result in a more thoughtful implementation or more interesting result. Once students are willing to separate planning and execution, they are often more willing to adopt the inquisitive and exploring attitude that the course tries to cultivate.

4 PROJECTS

The highlight of the Experimental Mathematics course is that students complete a research project, taking ownership of all phases of inquiry from selecting a problem, to writing code, to making conjectures from the resulting data, and even to proving their own conjectures. In most courses, students gain experience in answering someone else's questions; in Experimental Mathematics, students develop their own questions and bring their own novel approaches to answering those questions.

The rest of this section provides one model of structuring project requirements so that students have the tools to complete such an inquiry-driven project, often for the first time in their mathematical education. Staggering deadlines throughout the semester has produced projects of higher quality than in the first iteration of the course. In particular, the staggered deadlines include picking a project topic (early), giving oral presentations (mid-semester and end-of-semester), and writing a project report (one rough draft, and one end-of-semester final report).

In the Ithaca College experimental mathematics course for first year students [5], the final three weeks of the semester are entirely devoted to course projects. In Valparaiso's course, students are expected to start their project early and work on it side by with their other work. Although their attention is divided between assignments and project deadlines for most the semester, the longer span of 4 months to work allows time for students to generate more possible avenues of independent exploration early in the term. By the time the final few weeks of the semester approach, the students have a solid understanding of their project's background and can delve more deeply into their own questions when the focus switches to projects at the end of the term.

4.1 PROJECT TOPICS

Students choose a project topic early so that they can think about how to apply new programming ideas learned in other assignments to their topic. Many natural projects arise from branches of discrete mathematics such as number theory or combinatorics. Two sample project areas appear below:

- **Variations on Fermat's Last Theorem** – It was proven by Andrew Wiles in the early 1990s that there are no positive integers a, b, and c such that $a^n + b^n = c^n$ if $n > 2$. Experiment with $n = 3$ and possibly $n = 4$ and $n = 5$. Since we know that there are no triples of integers where $a^3 + b^3 = c^3$, you may try to find triples of integers where $a^3 + b^3 - c^3 = m$ (and m is a fixed integer, $m \neq 0$), or find 4 integers a, b, c, and d where $a^3 + b^3 = c^3 + d^3$, but a is not equal to c or d. See what patterns you can find.
- **Restricted permutations** – A permutation is a list $a_1a_2 \cdots a_n$ where each integer in $\{1, 2, \ldots, n\}$ appears exactly once. Up-down permutations avoid the "patterns" $\{uu, dd\}$ because there are no three digits where $a_i < a_{i+1} < a_{i+2}$ (up-up) or $a_i > a_{i+1} > a_{i+2}$ (down-down). However, there are many other pattern sets one could explore. What can you say about permutations of length n that have no copies of a given subpattern? Are there some patterns that force more predictable permutation structure than others?

Although project ideas are provided, students quickly realize that they must ask well-defined sub-questions to get started. Periodically students propose their own project ideas. If they can state a clear mathematical question to answer, they are encouraged to explore it computationally.

It is less important for a student to choose a topic that is new in the larger body of mathematics than it is to choose a project that is new *to that student*. Indeed, students who have adopted the same general project topic in different iterations of the course have often gone in very different directions by the end of the semester. No two students in the same iteration of the course can choose the same project, however, and projects are given on a first-come, first-served basis, so students must consider project topics early to guarantee their first choice area of inquiry.

4.2 PRESENTATIONS

By the midpoint of the semester, students have learned basic programming skills and data structures so that they are ready to begin their project in earnest. At this point, each student gives a 5-minute presentation to the rest of the class. The students are not expected to have solved their problem or to have done significant coding yet. However, they should be able to clearly state a question they plan to begin investigating, give relevant definitions, and give an idea of how they plan to use the computer in their project. These preliminary presentations serve two purposes: (a) students are interested in one another's projects, and with some encouragement they ask questions that give the presenting student additional avenues of exploration to consider as the semester progresses, and (b) the presentations get the students jump-started on the projects. The exercise of investigating their problems enough to make a preliminary presentation to the class ensures that the students have sufficiently organized their thoughts so that they can begin writing code that is relevant to their projects.

A rubric for presentation grading is shared with the class ahead of time, and considers organization, quality of material, and length of the presentation. If the talk is not well-organized or is missing a clear question to start experimenting with, this is the point where the student receives detailed feedback, early enough in the semester that they have ample time to correct their course of action before the final project presentation.

The final week of the course is a series of conference-length talks from students about their projects. Students talk for 15–20 minutes and are graded both by the instructor and their peers. The presentations show evidence of all three phases of work indicated in Table 1. Students reintroduce their topic and indicate how their questions evolved over the course of their experimentation. Students may demonstrate code they wrote during the course of the semester, especially if they chose a more visual project. Finally, students highlight conjectures and proofs that resulted from their experimentation. After staggered deadlines throughout the semester, the final week of presentations is a celebration of ownership in discovered results. Students also provide written feedback on one another's talks, exercising their ability to ask thoughtful questions of others, and sometimes bringing to light interesting perspectives on the material that the faculty instructor may not have considered.

4.3 PROJECT REPORTS

Along with oral presentations, each student is expected to write a report explaining the question they explored, how they explored it, and what they learned. There is no specific page quota, but students are told that the paper should give thorough evidence that they really did work on the project throughout the semester. Students submit a rough draft of their paper midway between the preliminary presentation and final presentation so that they can get instructor feedback. While the rough draft is a participation grade, the more seriously they take the writing exercise, the better feedback they get to prepare for the final report. The instructor can clear up misconceptions, point out conjectures that should be provable before the semester is over, and suggest additional threads of exploration for the final few weeks of the course.

5 STUDENT FEEDBACK

At the time of this writing, Valparaiso's Experimental Mathematics course has been through four iterations. Student feedback is informal, either taken with permission from expository exam essays, or via anonymous course evaluations, and has been extremely positive. For mathematics majors, the transition from computation to abstract reasoning can be a complicated jump, but a necessary one to truly get a sense of the discipline. One math major who completed the course described his change in approach during the semester as,

> "I'm learning math isn't just about memorizing formulas and plugging in numbers, but building on what you know, asking your own questions, and realizing not everything has a known answer just yet."

Yet other students take the course as an elective from other STEM disciplines. One engineering major described his personal transformation in approach as,

> "I was always taught: here is a concept, here is what it does, here is how to do it. I figured stuff that I need to learn would always just be given to me. This class has given me an appreciation for actually getting to explore concepts and learn on my own, which is something I would previously never thought would have worked."

Both are indications that by encouraging students to approach open-ended questions daily and to conduct their own semester-long research project, participants are learning the art of inquiry and leave the course with greater confidence for future mathematics endeavors.

ACKNOWLEDGEMENTS

The author thanks the editors and anonymous referees for suggestions that helped improve the content of this paper.

REFERENCES

[1] Borwein, J., D. Bailey, and R. Girgensohn. 2004. *Experimentation in Mathematics: Computational Paths to Discovery*. Boca Raton, FL: AK Peters/CRC Press.

[2] Bailey, D., J. Borwein, N. Calkin, R. Girgensohn, D. Luke, and V. Moll. 2007. *Experimental Mathematics in Action*. Boca Raton, FL: AK Peters/CRC Press.

[3] Borwein, J. and D. Bailey. 2008. *Mathematics by Experiment: Plausible Reasoning in the 21st Century*. Boca Raton, FL: AK Peters/CRC Press.

[4] Borwein, J. and K. Devlin. 2008. *The Computer as Crucible: An Introduction to Experimental Mathematics*. Boca Raton, FL: AK Peters/CRC Press.

[5] Brown, D. 2014. Experimental Mathematics for the First Year Student. *PRIMUS*. 24(4): 281–293.

[6] Conway, J.H. 1972. Unpredictable Iterations. Proc. 1972 Number Th. Conf., University of Colorado, Boulder, Colorado. 49–52.

[7] *Experimental Mathematics*, `www.tandfonline.com/loi/uexm20`.

[8] Oliveira e Silva, T. 2008. Verification of the 3x+1 Conjecture. `http://www.ieeta.pt/~tos/3x+1.html`.

BIOGRAPHICAL SKETCH

L. Pudwell is an Associate Professor of Mathematics and Statistics at Valparaiso University in Valparaiso, Indiana. She received her undergraduate degrees from Valparaiso, and then earned her Ph.D. in Mathematics at Rutgers University. Her research is in enumerative combinatorics, and she is passionate about involving undergraduate students in her work. She has co-directed an NSF-funded summer Research Experience for Undergraduates program since 2010. She is also a councilor for the Mathematics and Computer Sciences division of the Council on Undergraduate Research.