Algorithmic Social Sciences Research Unit
ASSRU

Department of Economics
University of Trento
Via Inama 5
381 22 Trento Italy

Discussion Paper Series

11 – 06

HOW SHALL WE PREPARE STUDENTS FOR ATTACKING NEW SCIENTIFIC PROBLEMS WITH COMPUTATION?

Rosalind Reid[1]

March 2011

[1] Executive Director, Institute for Applied Computational Science, Harvard School of Engineering and Applied Sciences, 29 Oxford Street, Cambridge, MA 02138 USA. rreid@seas.harvard.edu

Abstract

Computation is making its way into the mainstream of natural and social science research by fits and starts. "Computational science," once a toolkit, is emerging as a fundamentally new approach to exploration and hypothesis formation as well as analysis. It is not yet clear, however, just how computation will make novel contributions and change the nature (not just the methodology) of science. Harvard is engaged in creating a program that may test new ideas about how scholars should be trained. The rising dominance of large computation means, for example, that scholars must work in large and interdisciplinary groups in the future. And to advance the field, they must learn what it means to make a scientific question computable.What preparation will enable them to create this new field? This article reports on discussions toward establishing a new curriculum at Harvard and observations of the experience of similar programs at Stanford.

Introduction: What is computational science?

Science (whether natural or social) advances by continually testing and refining models of the world—falsifiable, simplifying statements ("x is a function of y") put forward to explain how things work. The traditional scientific method tests those models against data from experiments and observations.

How does the new elephant in the laboratory, the computer, alter this picture? The multiple answers to this question are interesting and controversial. They are likely to change in coming years, and so the challenge of training students to work in computational science is both daunting and exciting.

The impact of computation in science is today seen most clearly the work that it enables: modeling and data analysis of a new complexity and scale. A model that captures the rules and characteristics of a complex system can be built *in silico*, and new hypotheses can be tried out in the virtual environment. An intelligent computer program can even ingest a mass of information about a system, *create* a model or an array of possible models, and evaluate them against new data, performing most steps in the scientific method over and over while the human scientist sleeps.

Does the emergence of computation suggest a qualitative as well as a quantitative shift in science; is "computational science" a radical shift in thinking, a new paradigm?

The answer is probably "not yet." Computation-enabled advances in a few fields—genomics provides an example—have provided a glimpse of how it might work. The demands of science can drive innovations in computation that open new pathways for science in a process much like peeling an onion, each phase of research revealing a new level of complexity that might, in turn, require a new model of computation. The revelation that gene networks may have dynamics similar to those seen when computers themselves are networked, for example, poses deep questions crossing mathematics, computer science, physics and statistics. Many scholars hope computation will help uncover deep realities inaccessible by traditional methods.

In many domains the computer remains primarily a calculation engine, and the interplay of ideas from computer science and mathematics with ideas in science is only beginning to take place. It is perhaps telling that recognition of computational science as a discipline has not emerged within academia; rather, scientists tend to think of computation as a toolkit, one methodology among many. This attitude may have something to do with the suddenly ubiquitous nature of computing. It is understood that truly computational approaches in science are difficult, requiring mastery of mathematics and programming techniques that are not part of standard science training, along with deep domain knowledge. However, it it not widely appreciated that advancing computational science poses its own powerful intellectual challenges.

Harvard University is a vibrant academic environment that brings together science, engineering, mathematics and computer science. Until recently, the university did not attempt to define "computational science" and focus on the creation and application of new computational approaches to science. A new effort to do so, by focusing on the training of the bright young scientists who will redefine computational science, is the subject of this report.

Harvard's experiment

Harvard University, although not fundamentally a technical institution, came early to computation. In the 1940s and 1950s, the pioneers who worked with IBM on the Harvard Mark series of computers (including Howard Aiken and Grace Hopper) established some of the early principles of computer programming. Aiken, driven to programmable computer design by problems in physics, worked first with electro-mechanical systems. The later Harvard Marks were increasingly electronic but still massive computers. An interesting 1950 cover illustration for *TIME* magazine (Fig. 1) depicts the Mark III as the electronic "giant brain" of a naval officer, its human eye reading a tape and one human hand typing on a teletype.

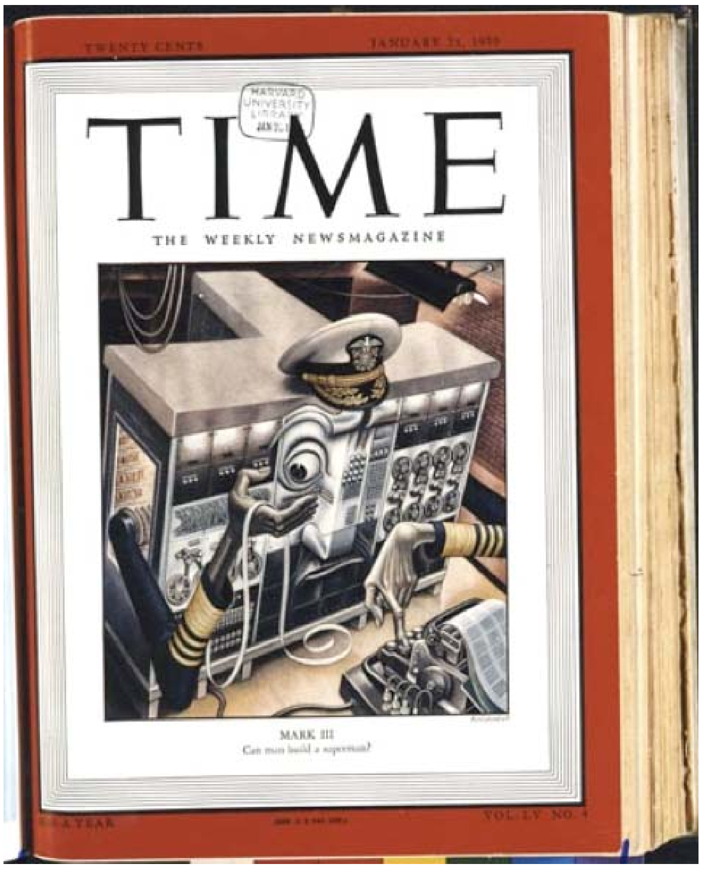

Figure 1. Color image of a *TIME* magazine cover for issue of January 23, 1950. It shows the Harvard Mark III depicted as a naval officer with one eye, a cap and two arms with hands, one holding a piece of paper tape and the other pressing a key on a teletype. The Mark III is shown with dials, buttons and keys. The subscript is "Can man build a superman?" Vol LV No. 4. From the Computer History Museum collection.

Fast-forwarding from this heady era to 2010, we can envision a similar illustration in which the computer serves as the "giant brain" of a Harvard geneticist, materials engineer, neuroscientist, climate modeler or astrophysicist. The biggest production computer at Harvard, Odyssey, has more than 11,000 compute nodes; typically at least half are busy at any moment, sorting out particle collisions or DNA assays.

In the six decades between the Harvard Mark III and today, one might say that science and the computer have co-evolved. Harvard scientific training, meanwhile, remains classical. The Harvard curriculum focuses on what is considered fundamental, giving relatively little emphasis to practical and methodological training outside its professional graduate schools. Although students widely use computer programs to do all their classroom work, computation itself is not widely considered fundamental outside specific curricula such as computer science and biostatistics.

What should a computational scientist be able to do?

The new institute is the brainchild of a physics professor, Efthimios Kaxiras, and the new Dean of the Harvard School of Engineering and Applied Sciences, Cherry A. Murray, also a physicist. Profs. Kaxiras and Murray believe that Harvard has the potential to blaze a new path in computational science through the collaborative development of a new curriculum for graduate students. Although their plan calls for engaging scholars across the social and natural sciences, most of the tools for doing this lie within the School of Engineering and Applied Sciences, which is home to Computer Science, Applied Mathematics, engineering and faculty working in fields including applied physics, materials science and environmental science.

Indeed, a quick survey of computational science programs at U.S. and European universities by this writer indicated that most have grown out of engineering programs. Many universities identify a set of widely applicable tools—parallel programming, numerical methods and so on—and fashion a curriculum from them, typically with emphasis on physics or engineering applications such as computational fluid dynamics. Development of graduate training along these lines has been broadly supported by the U.S. Department of Energy, which provides a large number of computational science graduate fellowships.[1]

A somewhat different approach has been encouraged by the U.S. National Science Foundation. The NSF has established an Office for Cyberinfrastructure[2] and encourages graduate training through grants to university researchers for "cyber-enabled discovery and innovation" projects. During her recent stint at NSF as Assistant Director for Computer and Information Science and Engineering at NSF, Jeanette Wing of Carnegie Mellon University emphasized that cyberinfrastructure must be conceived as incorporating "computational thinking" into science and engineering. NSF is taking a highly practical approach, now supporting the development of reusable software tools for science.

In envisioning a new curriculum in computational science, Drs. Murray and Kaxiras were concerned that although Harvard students use computation extensively, it remains a "black box" to many of them. A science graduate student who is good at mathematics and handy with software and programming might write much useful code for her lab but is not someone who will transform her field through more powerful computation. Likewise the many students who use popular software packages for their physics and mathematics homework are not necessarily thinking about the nature of computation; they simply compute as a way of solving hard problems. Although the first step toward

[1] http://www.krellinst.org/csgf/

[2] http://www.nsf.gov/dir/index.jsp?org=OCI

creating computational scientists is to have young scientists start computing, the next step is equally important: having them understand computation on its own.

Recently a group of Harvard faculty were asked what a well-prepared computational scientist *should* be able to do. I would like to build an imaginary training program in computational science by starting with their answers, which are the learning outcomes for our new curriculum.

> *1. A computational scientist should be able to produce a computational solution to a problem that is understandable and reproducible.*
>
> *2. A computational scientist should be able to communicate across disciplines and collaborate in a team.*
>
> *3. A computational scientist should be able to model complex systems.*
>
> *4. A computational scientist should be able to use computation for advanced data analysis.*
>
> *5. A computational scientist should be able to make or enable a breakthrough in a domain in science.*
>
> *6. A computational scientist should be able to parallelize computer code.*
>
> *7. A computational scientist should be able to evaluate and compare multiple computational approaches to a scientific challenge and choose the most appropriate and efficient one.*

These deserve a bit of elaboration.

> *1. A computational scientist should be able to produce a computational solution... that is understandable and reproducible.*

An essential quality of scientific research is that a result must be able to be replicated by another scientist. Many scientists are suspicious of complex computer models profferred as explanations of natural phenomena. Perhaps they produce very interesting-looking results, but how do we rigorously show that they provide insights into the real world? Imagine that computer code developed by a former graduate student in a geophysics lab is ported to a new computer cluster, modified and fed new data. The fact that the previous results obtained with this software were well validated does not mean that the new code and the new results are robust in the new computing environment. Nor does success in this endeavor mean that anything important and new has been learned.

A computational scientist must understand computation deeply enough to create code that can be used again to replicate results, to choose an appropriate computational approach, to devise and perform good tests on the computation and to build new software that is also robust. Put simply, a computational scientist must be able to do more than “hack” her way to solving a problem.

The collaborative nature of computational science (see 2 below) adds further definition to this requirement. Software engineers have developed sophisticated methods and tools for working together to iterate and test computational solutions. In order to produce a computational artifact that will be useful by others, the scientist must follow best practices such as versioning, testing and documentation, and collaborators must work in a shared development environment where code is developed iteratively.

2. A computational scientist should be able to communicate across disciplines and collaborate in a team.

One of the characteristics of computational methods in science is the way that they force large-scale collaboration. A lone scientist can certainly build models and run experiments on a personal computer. But the power of computation has altered the scale of science: It has enabled the collection and manipulation of very large data sets and the exploration of new levels of complexity through powerful modeling. (In fact, the ability to collect and store data now outstrips our capacity for using those data intelligently.) It has also created whole new fields, such as genomics, that have large-scale pattern recognition and analysis at their core.

A scientific team might include people from statistics, informatics, computer visualization and applied mathematics all working with theoretical and experimental scientists trained in a domain such as systems biology. A special challenge of a training program in computational science, the Harvard group has realized, is the sociological one: Computational scientists are never going to work in independent-investigator mode, nor will they prosper if they can speak only the language of their own fields. We must train them as first-class collaborators.

3. A computational scientist should be able to model complex systems.

A fearless embrace of complexity—computational complexity, but also complex systems as objects of study—may be what distinguishes a savvy tool-user from a top-rank computational scientist. If one chooses to use computation to investigate problems in biology, multiscale and nonlinear phenomena, complexity will be a persistent theme. Some systems lend themselves to algorithmic approaches; others do not. Continuum and discrete systems are often coupled, creating an additional layer of complexity. A computational scientist needs to be prepared to deal with the multilayered complexity of the real world, the algorithmic complexity of computer models, the complexitiy of computer architectures and the peculiarities of floating-point mathematical operations; he must understand the dependencies among all of these and be able to discern the ways in which digital models succeed or fail as models of reality.

4. A computational scientist should be able to use computation for advanced data analysis.

This statement may seem trivial, but the halls of science are littered with the results of poor data analysis. At today's leading edge, an experiment might entail something like capturing and looking for anomalies in video streaming from cameras watching the night sky. Efficient algorithms that are tunable and can learn are demanded almost everywhere that data are being collected. Fundamental questions about data structures, data quality and data management, as well as unsolved questions in optimization, scalability, parallelization and signal processing, must be faced as science scales. Computational scientists will tackle these challenges.

> *5. A computational scientist should be able to make or enable a breakthrough in a domain in science.*

This is simply an aspirational statement about our graduates. We wish to recognize as first-class participants in computational science both those who build tools that open up new kinds of science, and those who use them to transform their domains. The necessary partnership between tool-makers and tool-users is part of the new collaborative ecosystem of computational science.

> *6. A computational scientist should be able to parallelize computer code.*

A participant in one of our planning sessions pointed out that computers will not get markedly faster in the near future; instead they will get *wider*. The new paradigm implied by parallel computing requires a program to decide when processors can be working alongside each other, what is stored in what sort of memory, and when parallel processors need to exchange information. This change in computing is sufficiently basic—students must understand the constraints and opportunities presented by parallelism and organize algorithms and data to exploit it—that it is an important part of our training approach.

> *7. A computational scientist should be able to evaluate and compare multiple computational approaches to a scientific challenge and choose the most appropriate and efficient one.*

Part of the debate at Harvard involves the traditional pressure on graduate students to master the core of their subject matter and move swiftly into a specific area of research. The participants in the new program wish to train a different, mathematically sophisticated and broad type of thinkers who understand numerical, probabilistic and inverse methods, no matter whether those methods are typically used in their particular domains, in order to choose the approach best matched to the challenge they face. In addition, a computational scientist must be an algorithmic thinker with practical experience in cleaving the natural world into computable parts, abstracting and representing them and finding an appropriate language amd computer architecture in which to work.

The Stanford experience

Harvard comes somewhat late to the "computational science" discussion, a fact that we hope to turn to advantage. Many early attempts by universities to develop programs integrating computation and science (including an earlier faculty initiative at Harvard) ran into roadblocks, and our colleagues elsewhere are generously offering advice from their experience.

Stanford launched the Scientific Computing and Computational Mathematics program[3] in 1988, offering M.S. and Ph.D. degrees. The program was housed by the School of Engineering and drew faculty from computer science, mathematics and several engineering fields; the late George Forsyth, Gene Golub and Joseph Oliger were among its leading lights. It had an emphasis on numerical analysis and aimed to provide interdisciplinary training for academics.

This early and important program was replaced in 2004 by the young Institute for Computational Mathematics and Engineering[4]. We have met some of the young people involved in this program, which offers a Ph.D. The program offers a home for bright students of great depth, who understand and are inspired by the possibilities of mathematics and computation for science and engineering. These are students who might otherwise have trouble finding acceptance in a single discipline. They are allowed and expected to range widely; for example, they are encouraged to do "computational consulting" for departments across the university; in this role they are called upon by the management, economics and business faculties, which at Stanford are closely integrated with engineering.[5]

Our Stanford colleagues had a special reason for welcoming the launching of Harvard's Institute for Applied Computational Science. An academic career path for the students who enroll in ICME's Ph.D. program today is not yet well defined; other institutions have not created academic homes for the field. Nevertheless, the rigor of Stanford's program has helped its students find a ready welcome in mathematics and engineering departments elsewhere.

The long view

Immediately, Harvard will begin offering courses that broaden the opportunities for computation-minded graduate students: In February, stochastic methods and kinetic modeling will be offered in new courses started by the Institute. Over the longer term, the evolution of the program will be influenced by the Harvard environment, which is not that of a traditional engineering school. Students will come to the program with a wide variety of interests and applications for their computation; many will wish to apply computational methods in economics and finance. It was recently reported that of the Harvard baccalaureate graduates in June 2010, one-third of those who went directly into employment accepted jobs in the financial sector.

We hope to provide a Master's degree-level program that meets the seven key objectives above, drawing on faculty in applied mathematics and computer science for new courses that complement the instruction and research support that the student receives in his or her chosen application domain.

[3] http://icme.stanford.edu/history/sccm_degree_programs.php
[4] http://icme.stanford.edu/
[5] Information about this program comes primarily from conversations with faculty during a recent visit.

After considerable faculty discussion, we have decided to offer both a Master of Science degree that can be integrated into a science Ph.D. program, and a two-year, stand-alone Master of Engineering degree. The Master of Engineering will incorporate a research project of at least a half-year's duration. We believe this degree will provide exceptional training for computational scientists who would like to pursue industrial careers.

Table 1 presents the course matrix for these two degrees. Faculty have examined current courses to assemble lists of elective courses that meet learning objectives 1–7 identified above. The core requirements for these students will consist of a pair of two-course sequences in Applied Mathematics and Computer Science.

	Master of Engineering			**Master of Science**	
	min	**max**		**min**	**max**
Core	4	4		3	4
Applied Mathematics electives	1	3		1	3
Computer Science electives	1	3		1	3
Domain electives	0	max of 2 total	2	0	2
299R research courses	0		2	0	2
Thesis	4	4			
Total	**16**			**8**	

Table 1. Proposed requirements for Harvard degrees in Applied Computational Science[6]

In the future Harvard may strive for the goal that Stanford has achieved—a stand-alone Ph.D. in computational science/mathematics/engineering. The scholarly support for this new field, in the form of journals and academic appointments, does not yet indicate that this is a practical move on behalf of our prospective students. A leading computational biologist suggested to us that the opposite goal is better: make computational science so widespread that it redefines the existing disciplines, and no one needs to use the term "computational."

Either future is possible, and indeed if computation became the way to do science, the need for a separate educational program would fall away. The good news, for all universities, is that if we create rich new ways of infusing traditional teaching with computational learning and experience, and making computation a prominent part of scientific training across the disciplines, all scientists will have the potential of becoming computational scientists, and the next generation will be empowered to transform science, just like the generations before them.

Conclusion: What is computational science?

[6] Kaxiras, Efthimios, and the Applied Computational Science Curriculum Task Force. 2011. *Proposal: Graduate Program in Applied Computational Science.* Submitted to Dean Cherry A. Murray February 25.

A fascinating dimension of the Harvard discussions has been the interplay between faculty members who come to the table with different ideas about what might be meant by computational science. Sitting on the sidelines during these discussions, I have come up with a sense of the intellectual core of the problem. It is a tapestry with many threads. As with any science, the threads are not statements but questions.

Computational science, as expressed in this new program, asks a series of interrelated and inseparable questions about many classes of problems in numerical, stochastic and inverse methods and data analysis and exploration. Here are just a few:

- In science, what is computable?
- Can we cleave nature into computable parts?
- How do we represent these for the purpose of computing?
- How do we create useful abstractions for scientific data?
- What are the important properties of scientific data, especially very large data?
- What models of computation are useful or can be created for science?
- How can we represent and understand uncertainty and work with heterogeneous data?
- How do we choose and express in code appropriate numerical methods for modeling a system?
- How do we couple models of different type, as in multiscale modeling?
- How do we find efficient algorithms for scientific computing?
- How do we optimize calculations and code?
- How best can we use machine learning to explore data?
- What are the properties of streaming data, and how can meaningful anomalies be detected in a stream being processed in parallel?
- What new architectures and algorithms are needed to work with models and data analysis at the petascale or terascale?
- How do we improve inference and feature detection in very-large-scale image data?
- Can we design new architectures that will expand the boundaries of science?

The core sequences and electives will cover these topics. Several will be organized to require students to work in groups and develop projects iteratively and collaboratively, for this is how large-scale computational science must be done.

We wish not only to open the black box but to find students wanting to work entirely inside it to create a new and important science.