Teaching of Computing to Mathematics Students

Programming and Discrete Mathematics

Jack Betteridge
Department of Mathematical Sciences
J.D.Betteridge@bath.ac.uk

James H. Davenport
Dept. Mathematics and Computing
J.H.Davenport@bath.ac.uk

Melina Freitag
Department of Mathematical Sciences
M.A.Freitag@bath.ac.uk

Willem Heijtljes
Department of Computer Science
W.B.Heijltjes@bath.ac.uk

Stef Kynaston
Department of Mathematical Sciences
S.J.Kynaston@bath.ac.uk

Gregory Sankaran
Department of Mathematical Sciences
G.K.Sankaran@bath.ac.uk

Gunnar Traustason
Department of Mathematical Sciences
University of Bath
Bath, UK
G.Traustason@bath.ac.uk

ABSTRACT

This paper describes a course that has been running for over nine years, teaching Programming to large number of Mathematics students. The distinctive features of it include the fact that it was designed as part of a wholesale curriculum review (rather than being a pre-packaged course), that its design took into account the nature of the syllabus and what else the students would be using programming for, both in the rest of their course and beyond, and that the course is more than "just" a programming course.

CCS CONCEPTS

• **Applied computing** → **Mathematics and statistics**; **Education**; • **Software and its engineering** → *Multiparadigm languages*;

KEYWORDS

Programming Education, Mathematics, MATLAB

ACM Reference Format:
Jack Betteridge, James H. Davenport, Melina Freitag, Willem Heijtljes, Stef Kynaston, Gregory Sankaran, and Gunnar Traustason. 2019. Teaching of Computing to Mathematics Students: Programming and Discrete Mathematics. In *Computing Education Practice (CEP '19), January 9, 2019, Durham, United Kingdom.* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3294016.3294022

This paper is an update of [6], which never appeared due to HMG's cuts to the Higher Education Academy.

CEP '19, January 9, 2019, Durham, United Kingdom

ACM ISBN 978-1-4503-6631-1/19/01…$15.00
https://doi.org/10.1145/3294016.3294022

1 INTRODUCTION

This paper describes a novel approach to the integrated teaching of computing and discrete mathematics to mathematics students, based on:

- Close collaboration, and team teaching, by the Departments of Mathematical Sciences and Computer Science;
- Development of a bespoke interdisciplinary syllabus, 50% discrete mathematics and 50% computing, rather than an "off-the-shelf" Computer Science syllabus;
- Using programming concepts as concrete instantiations of the mathematics concepts taught in the course, for example recursion as a counterpart to induction, viewing Fast Fourier Transforms as a "divide and conquer" algorithm;
- Choice of a programming delivery vehicle (MATLAB) close to the immediate needs of the students;
- Attention to the pedagogy of the *craft* of programming [5]. Many of our ideas are similar to those of [10], though this postdates our early work.

2 BACKGROUND

Until 1997, the University of Bath had a School of Mathematics Sciences, including a Computing Group. Programming was taught in Fortran, until in 1984 the second author led a move to C. Relevance to, and preparation for, the future computing streams was the principal criterion. Then a separate Computer Science degree, with a different first year, was introduced. This paper focuses on the evolution and practice of computing as it is taught to *mathematics* students.

Until 2009, first year Mathematics students (of which there are currently 304) took a programming course provided by the Computing Group and later by the separate Computer Science Department. The emphasis was on programming *per se* in a general-purpose language: C until 2000, then Java. The main weaknesses of this, frequently identified by students, were the lack of apparent relevance and the lack of connection with the rest of the curriculum where programming was used in later years, either in MATLAB or R. Following restructuring and detailed curriculum review (though

a complete curriculum review is not a necessary requirement!) the current model evolved.

3 WHERE DOES IT FIT?

The course is called, and delivers, **Programming And Discrete Mathematics**. It runs throughout the first year, as one of five compulsory streams: the others are Algebra, Analysis, Mathematical Methods and Probability/Statistics. The course is 50% Programming and 50% Mathematics, so the "programming" share of the first year is 10%, which is unchanged, but its effectiveness has greatly increased.

The Bath first year in Mathematical Sciences is just over 300 students, typically with A* grades in A-level Mathematics and Further Mathematics, or equivalent qualifications. In addition there are students in Computer Science and Mathematics, Mathematics and Physics etc., but these do not take XX10190.

3.1 Aims

In practice, the aim, which underpinned all the thinking as the course was being designed, was

> The course should **be**, and be seen to be, relevant to the rest of the mathematics curriculum, and not just as "a useful skill for later on".

From this followed the fact that it could not be *just* a computing course. Certain amounts of discrete mathematics (using the term slightly loosely) were added or moved from elsewhere, and we ended up with a 50:50 mixture with the programming (in MATLAB). This mathematics included orders of growth and the O-notation (which students seem to find more approachable with a concrete application), elementary graph theory, Fast Fourier Transforms, elementary coding theory and cryptography (Diffie–Hellman and RSA). The coding theory relies on the linear algebra taught in the algebra stream: moreover, it uses and emphasises the fact that it is taught over an arbitrary field. The cryptography part is helped by the fact that the MATLAB Symbolic Toolbox allows examples with realistic-sized numbers: indeed the students do two problem sheets, identical except that one has two-digit numbers for hand calculation, and the other has 60-digit numbers for MATLAB-assisted computation. However, by far the most important in terms of relationship with the programming was the teaching of induction.

The first few weeks of the course are based around the thesis that the Mathematical definition of induction is *equivalent* to programming implementation by recursion.

The first example, literally in week 1, is the Fibonacci numbers, which are defined by induction, programmed by recursion, and in the next four weeks have theorems on growth proved by induction, and have these related to the O-complexity of the programs produced earlier. More specifically, three families of solutions to the Fibonacci problem are presented, with the lecturers and students proving the complexity results.

Exercise 1 (Directly recursive). *Use $F_n = F_{n-1} + F_{n-2}$ to work down to the base cases. The complexity is exponential in n: $O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$.*

Exercise 2 (Iterative). *Define the base cases and use $F_i = F_{i-1} + F_{i-2}$ to work up until $i = n$. The complexity is linear in n: $O(n)$.*

Exercise 3 (Matrix-based). *Therefore playing to the strengths of MATLAB:*

$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} \text{ hence } \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix},$$

and using "divide-and-conquer" (recursive) exponentiation we can compute F_n in time logarithmic in n: $O(\log n)$.

3.2 Reinforcing the Link

One challenge when teaching an interdisciplinary course is convincing the students of the benefit of learning a topic outside of the perceived scope of their degree. Whilst many practising mathematicians use programming, or the thought processes behind programming, in their daily lives, this is not obvious to first year mathematics undergraduate students.

To answer the question "why are we learning programming on a mathematics degree?" a short (5–10 minutes) talk is given fortnightly by one of the tutors, or a lecturer who doesn't teach on the unit, to the whole cohort. This is at the beginning of a lecture and the tutor/lecturer explains a little about their research, and how they use programming. These seem to have been well-received, with thoughtful questions from the students after each one.

The link between mathematics and programming is further reinforced by the whole teaching body. The professors and tutors are sourced from both the Mathematics and Computer Science Departments, and range in specialities from Algebraic Geometry via Computer Algebra to Numerical Analysis and Computer Vision.

3.3 Course Text

Initially, we decided to go for an, essentially mandatory, course text. This was a composite text based on [3, 8] with a small amount of our own material. [6] describes the process, but, as the lecturers wrote more material, the mandatory course text became less attractive and we dropped it in 2016. [4, the successor to [3]] is still recommended as a MATLAB reference.

4 DELIVERY

4.1 Overview

The course runs throughout the teaching year (October–May), and is taught on the basis of two lectures and one whole-cohort problems class per week, the same as the other four streams (algebra, analysis, methods and probability/statistics), so is worth 12 ECTS credits. It is team taught, with a computer scientist (the second and fourth authors) and a mathematician (the third and seventh authors) taking responsibility for the lectures and problems class, and tutors, typically postgraduates or final-year MMath students, taking responsibility for the practical laboratories and marking mathematics work. Normally[1], each week has both "mathematics" and "computing" lectures and student work.

4.2 Whole-Cohort Classes

In a typical week, each lecturer gives one lecture, and both share the problems class, going over past and ongoing work. The ratio in

[1] We do take advantage of the flexibiity that team teaching provides, most recently to support parental leave.

the problems class is dynamic, and will be roughly 5% programming and 95% mathematics, unless more time is needed to respond to student queries on programming aspects.

4.3 Laboratory Classes

These are weekly and last for 50 minutes. We are fortunate enough to have a 75-seater laboratory, which we split into seven groups of 10 machines each. Each student is assigned to a *specific* group in a *specific* laboratory session, and the group has a designated tutor. Thus each student has the same tutor for the whole semester, and often for the whole year. The laboratory tutors are also responsible for marking the mathematics homework, and giving feedback during the laboratory classes, though in practice there is rarely enough time to do more than hand over the sheets of paper.

4.4 Virtual Learning Environment

We make heavy use of the University's Moodle VLE. Course materials and problem sheets are distributed via it, Coursework and tickable exercises are collected via it, feedback is given on coursework via it. The lecturers tend to respond to student e-mail queries by posting on the Moodle Forum, rather than replying directly, so that all students can see the (anonymised) question/answer.

4.5 Programming Videos

In the past a live coding demonstration formed part of the programming component of the problem classes in Semester 1, which allowed students to see how programs are written and debugged in real time. Starting in 2017 this was replaced by screen capture videos, with commentary, (produced by the first author) to allow more time for lecturing. These videos outline the solution to the previous week's programming assignment (tickable), but also include intentional mistakes and debugging advice. Furthermore, this format allows the demonstration of some more advanced features of MATLAB, e.g. plotting and timing. This video resource allows students to revisit topics in their own time as well as providing a reference during labs, and reduces their use of third party sources available online, which may not be relevant to this course.

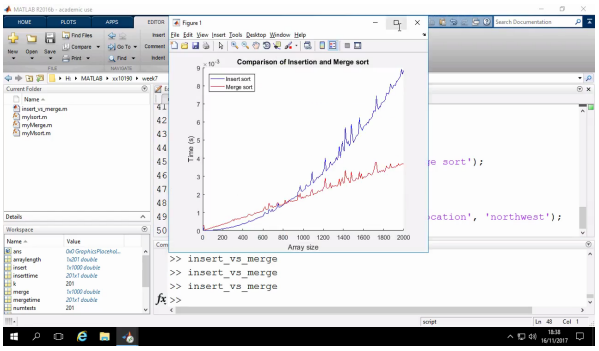

Figure 1: Frame from video demonstration: Using MATLAB to plot execution times for merge and insertion sort performed on different sized arrays.

4.6 Assessment

The Discrete Mathematics half of the course is assessed traditionally and in line with the other courses: formative exercise sheets done weekly, and summative end-of-semester exams (at 20% and 30% of the total marks). The *summative* assessment of the programming is done by two courseworks (one per semester) each worth 25%.

The novelty is in the formative assessment of the programming. In weeks when there is no significant effort on summative practical work, there is a weekly "tickable" exercise — in practice 12 in the year. By "tickable" we mean that they are assessed as pass/fail by the laboratory tutors in the sessions, and the students (and tutors) are told that we expect students to be able to pass every exercise with reasonable diligence. For example, the first such exercise is to write a recursive MatLab function to compute Fibonacci numbers from $F_n = F_{n-1} + F_{n-2}$, by analogy (this is made explicit in the exercise) with the supplied programme for computing factorials based on $n! = n*(n-1)!$. The supplied programme was constructed in front of the students in the lecture, and hence this follows the paradigm described as *modeling/scaffolding* by [10, §2.1].

Lest this seem too trivial, this is the specification of the last tickable in semester 2.

Exercise 4. *Write two MatLab functions:* `TreeAdd` *and* `traverse`*, in files of the same name.* `TreeAdd(t,str)`*, where* `t` *is a binary tree of strings and counts, and* `str` *is a string, returns the new tree with* `str` *inserted in order (or with the corresponding count increased if* `str` *was already in the tree).* `traverse(t,@f)` *should traverse such a tree* `t`*, calling the function* `f` *on each node in turn, in alphabetical order of the strings. In the parlance of [the programming] lectures, it should therefore do an* **inorder** *traversal.*

The incentive for the tickables is that the tickables develop the students' programming skills, and lead up to the summative practical work: with the motivation that failure to get 80% ticks will result in the summative coursework marks being reduced *pro rata*. In practice we have very rarely had to apply this restriction: the (very few) students who do not do the tickables fail the practical work anyway.

Many tickables from 5 onward are supported by a quiz: good students answer a quiz based on running their code, and then can submit their code using "conditional assignments" in the Moodle online platform. This is done in advance of the laboratory session, so that the tutors can look at it in advance, and spend the lab session concentrating on the weaker students who have not done the quiz.

5 DOES IT WORK?

It has certainly reduced the failure rate on programming: the Java course was frequently seeing a failure rate of over 30%, while the 2017/18 run of this module had a failure rate of 6%, of whom 1% had condoned failures and the other 5% had enough other module failures that they had to withdraw or repeat the year.

As a first-year course, it has several customers.

Numerical Analysis lecturers in subsequent years. These were initially identified as the prime beneficiaries, and the have indeed benefitted. The first Numerical Analysis course no longer disappears under the weight of teaching programming. In particular the students have met floating-point

numbers in XX10190. Even though the Scientific Computing Course is taught in a different language (Currently Fortran/MPI, soon more likely C), the students have experience with programming and the numbers on this Scientific Computing Course have more than tripled in the past 5 years, since the students know the concept of programming.

Statistics lecturers were an unexpected beneficiary. It turned out that there was enough MATLAB in the first semester to allow the second semester statistics course (following on from the first semester probability course) to issue a one-page MATLAB/R conversion sheet, rather than spend several weeks explaining R.

Computer Science lecturers in subsequent years. Several tens of the Mathematics students take Computer Science options such as Cryptography in later years, and having experience with programming greatly helps them with these options.

Students on the course The course is still not especially popular, and we do get in past students, especially those that have used these skills in placement, or are now using them in their research, to talk briefly about the benefits at the start of the problem classes. It does get discussed in Staff-Student Liaison Committees, often with the students from later years helping the staff explain the benefits.

Students after the course Many students use these skills later, either in courses or on placement, and some are explicitly grateful to the staff. This is hard to capture quantitatively. It is also worth noting that any unpopularity has worn off, and this course does not feature in NSS comments.

Placement students More than 100 students in the Mathematics department take placements in year 3 of their course. Placement visitors and students frequently report that programming in XX10190 was the most useful skill learned in their first 2 years which prepared them for their placement.

Former Students It is hard to unpick the effects of different elements of a programme, but we note that, *after* allowing for prior attainment etc. (see [7]), Bath mathematics students come sixth (male) or fourth (female) [1] amongst all mathematics courses for earnings five years after graduation.

6 WHO ELSE HAS DONE THIS?

The idea came out of the 2008/9 whole course review, with which the sixth author was heavily engaged, and discussions between him and the second author.

We know of no university adopting a very similar approach, though Surrey's *Professional Skills Development* course has some similarities (and we have talked with that course's main author). There is growing interest, though, in the light of the recommendation in [2] that every mathematician should learn to program.

7 WHAT WILL YOU DO NEXT?

We are coming up to another whole-course review. It is too early to pre-empt the decisions, but the following seem clear.

- The next curriculum must incorporate programming.
- The language need not be MATLAB, and Python is mooted as a successor.
- The decision is a major one affecting many courses, not just the "programming" course, but also Statistics and Numerical Analysis courses in particular.

8 WHY ARE YOU TELLING US THIS?

We learned various lessons, some Mathematics-specific but many not, which we think are useful.

(1) "Off-the-shelf" generic programming courses, with little or no linkage to the main subject the students are studying, can be very unpopular, and an approach that says "why do *these* students need to learn programming" can produce a better result. As more Departments look at incorporating programming for employability reasons (especially Mathematics ones in the light of [2]), this is worth noting.
(2) In particular, although programming is fundamentally a generic skill, significant thought has to be given to the most appropriate language for a first programming course, and Java, despite its general popularity in the U.K. [9], was not an appropriate choice for this course's predecessor. Equally, many subjects would find MATLAB utterly the wrong choice.
(3) Custom texts are perfectly viable, even down to class sizes of fewer than 100, and in the early years of course development were a valuable support.
(4) The "Tickable" idea, now adopted by our Computer Science Department for their introductory course, is a useful way of getting students, particularly the weaker ones, working on programming immediately, rather than waiting for the coursework then panicking.

Acknowledgements

We are grateful to Ivan Graham and Alastair Spence, who have also lectured on this course, and William Saunders for being a co-producer of the videos. We are also grateful to the many tutors on the course over the years, and to the students, and especially former students, who have fed back on the course as it has evolved.

REFERENCES

[1] BBC. 2018. Which university courses boost graduates' wages the most? https://www.bbc.co.uk/news/education-44413086. (2018).
[2] P. Bond. 2018. An Independent Review of Knowledge Exchange in the Mathematical Sciences. https://epsrc.ukri.org/newsevents/news/mathsciencereview/. (2018).
[3] S.J. Chapman. 2009. *Essentials of MATLAB Programming.* Cengage.
[4] S.J. Chapman. 2013. *MATLAB(R) Programming with Applications for Engineers.* Cengage.
[5] J.H. Davenport, A. Hayes, R. Hourizi, and T. Crick. 2016. Innovative Pedagogical Practices in the Craft of Computing. In *Proceedings LaTiCE 2016.* 115–119.
[6] J.H. Davenport, D. Wilson, I. Graham, G. Sankaran, A. Spence, J. Blake, and S. Kynaston. 2014. Interdisciplinary Teaching of Computing to Mathematics Students: Programming and Discrete Mathematics. *Accepted by MSOR Connections* (2014). http://opus.bath.ac.uk/37841/
[7] Department for Education. 2018. Undergraduate degrees: relative labour market returns. https://www.gov.uk/government/publications/undergraduate-degrees-relative-labour-market-returns. (2018).
[8] S.S. Epp. 2011. *Discrete Mathematics with Applications, 4th edition.* Brooks/Cole Cengage Learning.
[9] E. Murphy, T. Crick, and J.H. Davenport. 2017. An Analysis of Introductory Programming Courses at UK Universities. *The Art, Science and Engineering of Programming* 1, 2 (2017), 18–1–18–23.
[10] A. Vihavainen, M. Paksula, and M. Luukkaine. 2011. Extreme Apprenticeship Method in Teaching Programming for Beginners. In *Proceedings 42nd ACM technical symposium on Computer Science Education.* 93–98.