Computer programming in the UK undergraduate mathematics curriculum

Computer programming in the UK undergraduate mathematics curriculum

Christopher J Sangwin[a] and Claire O'Toole[a]

[a]School of Mathematics, University of Edinburgh, Edinburgh, EH9 3FD, United Kingdom

ARTICLE HISTORY
Compiled March 30, 2017

ABSTRACT
This paper reports a study which investigated the extent to which undergraduate mathematics students in the United Kingdom are currently taught to programme a computer as a core part of their mathematics degree programme. We undertook an online survey, with significant follow up correspondence, to gather data on current curricula and received replies from 46 (63%) of the departments who teach a BSc mathematics degree. We found that 78% of BSc degree courses in mathematics included computer programming in a compulsory module but 11% of mathematics degree programmes do not teach programming to all their undergraduate mathematics students. In 2016 programming is most commonly taught to undergraduate mathematics students through imperative languages, notably MATLAB, using numerical analysis as the underlying (or parallel) mathematical subject matter. Statistics is a very popular choice in optional courses, using the package R. Computer algebra systems appear to be significantly less popular for compulsory first year courses than a decade ago, and there was no mention of logic programming, functional programming or automatic theorem proving software. The modal form of assessment of computing modules is entirely by coursework (i.e. no examination).

KEYWORDS
Programming and programming languages; Country-specific developments; Undergraduate Mathematics

Biographical note:
Chris Sangwin is Professor of Technology Enhanced Science Education at the University of Edinburgh. His learning and teaching interests include (i) automatic assessment of mathematics using computer algebra, and (ii) problem solving using Moore method and similar student-centred approaches.

Claire O'Toole is a student in the School of Mathematics at the University of Edinburgh.

1. Introduction

A basic question university mathematics departments have to address is should we teach our undergraduate students to programme a computer? If so, what should we teach and how? This paper reports the results of a national survey to establish current practice in UK mathematics degrees. Through this paper we hope to (1) inform other

CONTACT C. J. Sangwin Email: C.J.Sangwin@ed.ac.uk, orcid.org/0000-0002-3725-8625

colleagues considering this issue now, (2) record some of the decision making processes used by colleagues in developing curricula and teaching, and (3) create a permanent record of what is happening in 2016 in what is a fast changing area.

It has long been acknowledged that computers serve many purposes in mathematics education, [1] and programming forms only part of a broader use of computing in the curriculum. For example, the previous review articles [2], [3] and [4] developed a hierarchical taxonomy. The abstract study of algorithms, including correctness and limitations, is highly mathematical and forms part of the heart of computer science as a discipline. Software is also regularly used to explore key concepts, e.g. patterns in sequences of numbers or formulae algebraically or graphically see e.g. [5], [6]. Software is also used to compute results which are actually wanted, e.g. numerical simulations, or in statistics. Some courses teach students to acquire specific IT skills, e.g. learning how to use a particular software package to address an anticipated future employment need. The research reported in this article considers only one branch of the taxonomy developed by [2], namely programming computers.

Mathematics, at university and in school, is a rather traditional subject. Indeed, the chalkboard has retained its place for university mathematics, see [7]. The relatively newer subject of computer science has started to articulate its goals and boundaries which at times appear to have significant commonalities with mathematics.

> The essence of computational thinking is abstraction. In computing, we abstract notions beyond the physical dimensions of time and space. [8]

While we do not want to engage in subject turf wars, mathematics departments do have to address the place, if any, which computer programming has in an undergraduate mathematics education. An accurate report of what currently happens in university mathematics departments contributes background evidence to this discussion.

The Subject Benchmark Statement for Mathematics, Statistics and Operations Research, [9], forms part of the UK Quality Code for Higher Education and sets out the expectations that providers of UK higher education are required to meet. Under subject specific skills, they list 'use computers as an aid to mathematical study and for acquiring any further information' (pg 16). They also discuss online learning (§4.9, p. 20) and using mathematical software.

> §3.16 All graduates have some knowledge and understanding of mathematical computing, with direct experience of specialist software and/or of programming, with an awareness of the appropriateness of the software for the problems being addressed and, when feasible, the nature of the algorithms on which it is based. [9, p. 16]

In terms of the benchmark standard for honours degrees, a graduate who has reached the bachelor degree with honours threshold level should be able to demonstrate 'competent use of appropriate computer technology in mathematics'. The word 'use' here could be interpreted solely as the ability to use a package to explore concepts or compute results and our interpretation is that the phrase 'and/or of programming' does not mandate computer programming for undergraduate students.

In the UK, typically, a student will study 120 credits per year, divided into 10 (or 20) credit modules over two semesters. That is, they will study somewhere between 3 and 6 subjects at a time for two semesters of about 11 weeks each. At least in England, in undergraduate mathematics degrees, students commonly study few (if any) courses in subjects outside the discipline and have relatively little choice until their final year of study. This does vary by university and department, but the general picture should be contrasted with a liberal arts tradition in North America where

undergraduate students typically have much more choice over which modules comprise their degree programme and even sometimes which member of staff they choose to teach a particular subject.

2. Background

2.1. *What is programming?*

Computer programming (programming) is a process that starts with a problem and ends in an executable computer program, also called software or code. Our focus is on what is taught and why, rather than an investigation of what is learned, or on any difficulties students routinely have with learning specific topics. Much has been written about the difficulties students have in learning to programme, see [10] and [11] for reviews of teaching programming. Primary hurdles include assignment of values to variables, following sequential logic, recursion/iteration (see [12]) and concurrency. For further background information, including key concept inventories, see [13], [14] and [15].

There is a lot of ground between issuing individual commands to a computer, e.g. in spreadsheet cells or a typed calculator programme (including Wolfram Alpha), and writing complex structured code comprising many lines in separate packages. For the purposes of this paper we mean writing relatively simple code, which has at least some of the following hallmarks.

- Multiple dependent lines (not just a single command or macro).
- Control structures such as conditional logic and branching.
- Appreciation and use of data types both elementary (e.g. integer, float and string) and structured (e.g. hash tables, objects).
- A sub-division of a problem into separate structures such as functions or objects.
- Use of iteration, recursion, or parallelism in the solution process.

We do not require students to encounter all the ideas listed as hallmarks although we would expect many (if not most) to do so. Where such ideas are encountered they would normally be done consciously and explicitly. We do not expect students to learn particular programming paradigms (e.g. imperative, functional, object oriented) or develop a deep understanding of the differences between them. Much practical programming relies on assembling code from libraries of functions. We expect mathematics students to programme in this way and to consider foundational programming of algorithms from scratch. We certainly do expect students to write some code for themselves in a particular (prescribed) language. That is, we expect a practical component beyond the abstract mathematical study of algorithms.

2.2. *Computing in schools*

In July 2013 there was a significant change to the National Curriculum for schools in England[1]. From September 2014, Information and Communications Technology (ICT) was changed to Computing, see [16] and [17]. Computing, the new subject, is split into three categories; computer science, information technology and digital literacy. School students, from the age of five until they leave the school system, will be learning skills

[1]Responsibility for school education in the United Kingdom is devolved to the four regional administrations in England (the largest), Scotland, Wales and Northern Ireland.

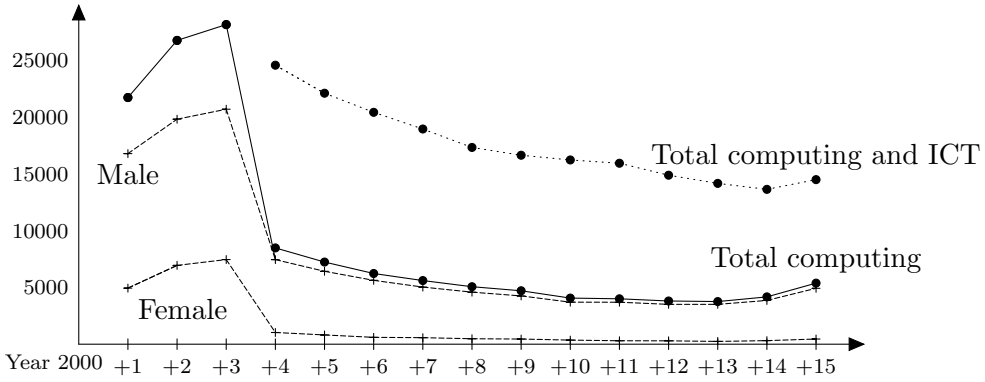

Figure 1. Numbers of students taking examinations in GCE Advanced Level Computing (solid line) and Information and Communication Technology (ICT) from 2001 to 20015. Data from `www.jcq.org.uk` (retrieved 27 June 2016)

such as the foundations of computing science (for example logic, algorithms etc.), will be taught to adapt and write their own programs. If these curricular changes succeed then school students arriving at university will already be significantly more proficient in computer programming than is currently the case. Hence it is very likely that universities will also have to change. This research is in anticipation of potential change in university mathematics courses.

Concurrently with this new curriculum are many initiatives for schools such as the Raspberry Pi [18] and the more recent Micro:bit (see `https://www.microbit.co.uk/` retrieved 18 August 2016). Both of these schemes aim to make programming accessible to all school students across the UK and in 2016 the Micro:bit was given without charge to all children in their first year of UK secondary school.

In England, Wales and Northern Ireland GCE Advanced Level examinations (A-levels) are the final examinations taken in school, typically taken aged 18, and are normally used as entry criteria for higher education. The number of students taking A-level Computing are shown in Figure 1. Note that currently only 8.5% of students taking computing are female, which is by far the lowest proportion of any subject: physics, the next lowest proportion, has 21.5% female students. The introduction of A-level Information and Communication Technology (ICT) in 2004 coincided with a crash in numbers of students taking computing, followed by a steady decline in overall numbers. Students appear to have opted instead for ICT, although [19, p. 87] suggests that computing and ICT GCE A-level subjects are of almost identical difficulty, (and comparable to mathematics A-level in difficulty). This sustained trend was only reversed in 2014-2015 solely due to a significant increase of numbers in A-level computing of 29.1%, the greatest increase that year of any subject. If the current rate of growth is sustained then computing will overtake mathematics, the most popular A-level subject, in under 12 years. This is, of course, unrealistic. While, in the UK context, the numbers of students in each cohort taking A-level computing are modest we note that significant and rapid growth is possible. Notably, A-level Further Mathematics rose from around 5,000 students in 2005 to over 14,000 students in 2014 (160% increase, an average of 11.2% per year) in what is known to be one of the most demanding subjects, [19]. The sustained work of the MEI Further Mathematics Support Programme in promoting and directly facilitating the teaching and learning of the subject in schools appears to have resulted in a significant rise in numbers taking further mathematics, see [20] but also note the wider trends in [21]. It is therefore

possible, with the right support for teachers, to make significant changes in patterns of participation in education.

2.3. *University and Employability*

Computer Science is a very popular subject at university. Indeed, in each of the years 2011-2016, more UK students began a Computer Sciences undergraduate degree course than started Physics, Chemistry and Mathematics combined. However, between 2007 and 2014 computer science consistently and significantly had the highest rate of unemployed graduates of any science, technology, engineering or mathematics (STEM) subject in the UK. [22] suggests that unemployment appears to be worse among universities with low average entry scores and that students who lack awareness of the importance of choosing the right course and who do not develop 'soft skills' alongside their technical knowledge are also less likely to find graduate employment. There are also significantly fewer women studying Computer Sciences at university (13% compared to 32% in STEM generally and 40% in mathematics)[2]. We acknowledge and are mindful of these problems: mathematics departments do not want to unwittingly implement situations which replicate them in mathematics degree programmes.

In other disciplines, particularly engineering, it has long been acknowledged that computers significantly change the educational needs of students. For example, the Société Européenne pour la Formation des Ingenieurs (SEFI) have undertaken regular curriculum development and their mathematics working party report [23] focused on the changes to the mathematical needs of engineers in the light of computing. '§ 2.1 The development of new technology, notably computers, is doubtless the largest single factor that has affected the redesign of a Core Curriculum [...]' [23, p. 6]. The importance of computing to engineers was echoed in the current SEFI curriculum [24].

3. Research questions

To what extent are undergraduate mathematics students taught to programme a computer as a core part of their mathematics degree programme in the UK? What are they taught, why and by whom?

- Is computer programming specifically mentioned in the programme level specifications, e.g. as a desirable graduate level attribute?
- Are undergraduate students required to learn computer programming as a core part of their mathematics degree programme? If so, how much time is spent and how is programming assessed?
- What programming languages are used and why were these selected?
- Who does the teaching: mathematics staff or computer science staff?
- Did the department review computing recently and if so what where the conclusions?

We are also interested in the attitudes to computer programming held by staff in mathematics departments. Accurate and comprehensive attitudinal data is very difficult to gather. In this study we have chosen to gather and analyse factual data. Unlike schools, university mathematics departments enjoy considerable freedom to develop their own

[2] `www.hesa.ac.uk` (retrieved 21 June 2016), Table 4: HE student enrolments by level of study, subject area, mode of study and sex 2010/11 to 2014/15. Statistical First Release 224.

curricula. We believe the choices made for compulsory undergraduate courses are normally the result of considerable thought, debate and compromise. Current curricula therefore embody the results of departmental reviews and discussion and so enacted curricula are a manifestation of the attitudes of staff in the department. At least, the enacted curricula choices represent the *prevailing* attitudes of staff. However this freedom is constrained by institution wide policies, particularly those associated with the nature of assessments.

4. Methodology

To address our research questions we undertook a national survey of UK mathematics departments. We developed the short online survey recorded in Appendix A. Given much of the factual data we are primarily interested in is openly published, we had the opportunity to engage in follow up correspondence with departments to clarify our understanding and obtain complete module information. Note that this survey asked respondents to reply on behalf of their department, with only one final question on their personal attitudes. Where the responses pointed us to publicly available information such as module descriptions from websites, which we subsequently retrieved, we have attributed the information to particular departments.

To identify mathematics degree/department we used the Universities and Colleges Admissions Service in the UK (UCAS) admissions website. We selected those UK departments offering a Mathematics BSc degree programme (with a UCAS admissions code G100) starting in 2016. This resulted in a list of $N = 73$ departments of interest. We rejected taking the list of departments sending returns to the Research Excellence Framework (REF) as this did not adequately reflect our interest in undergraduate teaching.

In May 2016 a link to the online survey was sent (1) direct to individuals known to us and (2) to the mailing list of the Heads of Departments of Mathematical Sciences (HoDoMS). This resulted in over 30 responses on behalf of unique departments. During May and June 2016 further follow-up correspondence was used as required, e.g. to obtain module specifications which were not publicly available online. We also identified departments which did not respond during the first round. Through an internet search we identified individuals, such as 'directors of studies' or 'heads of administration', who were likely to have a significant role in undergraduate teaching and contacted them directly with one further request to respond to our online survey. This resulted in over 60 responses on behalf of 46 unique departments. Where we had more than one individual from a particular department respond we amalgamated the results and cross checked any information.

We have chosen to concentrate on assembling factual curricula data but we are also interested in attitudes. Where module specifications were not publicly available we then contacted those institutions and asked if they could be sent via email. When advised these were not confidential we have also attributed the information to particular departments.

We acknowledge the unreliability of attitudinal data and the difficulties in gathering a comprehensive and representative sample of such data. Given the method of recruiting participants we are particularly concerned about potential bias in responses to this part of the survey. However, we did include one question on attitudes held by staff to computer programming in our survey. We know discussion of programming can be contentious and anticipated colleagues would possibly hold strong views

on this subject. This question was designed to enable respondents to express those views separately from the factual data. We treat this information as confidential and with caution and do not report it here as representative of staff in UK mathematics departments. Responses to this question are included in our discussion.

5. Results

We received complete responses from 38 (52%) departments and a partial response from 46 (63%) individual departments.

5.1. *Is computer programming specifically mentioned in the programme level specifications, e.g. as a desirable graduate level attribute?*

We found that programme level specifications are often very general, possibly just reflecting aspiration rather than having a serious day to day impact on teaching. As one colleague comments to this question:

> Not to my knowledge... I'm not sure we have programme-level outcomes for our degree programmes, which is an oversight if not. It's possible I'm just not aware of them.

Only 13/46 (28%) of programme level specifications we gathered specifically mentioned computer programming as a desirable graduate level attribute. The majority of programme level specifications make reference to the importance of 'computing' without specifically making the distinctions between computer programming, professional training in packages (such as R or MATLAB), or using computers to experiment and calculate. A number of colleagues suggested that programming was an implicit requirement or there was a strong expectation that students would programme particularly within computational projects, even if programming was not explicitly mentioned.

5.2. *Are undergraduate students required to learn computer programming as a core part of their mathematics degree programme? If so, how much time is spent and how is programming assessed?*

Table 1 shows the extent to which programming is taught, with the percentages calculated on the basis of 46 partial responses. Computer programming is a compulsory part of 78% of the single honours courses in mathematics. However, 11% of mathematics degree programmes do not teach programming to their students as a compulsory subject. We note with interest that while only 28% of programmes specifically mentioned computer programming as a desirable graduate level attribute, computer programming is a compulsory part of 78% of the single honours courses.

We received full module specifications for 65 modules which colleagues identified as focused on, or containing a significant element of, computing. Table 2 records data on the contribution of an examination to the overall module mark. Note that in all modules we examined the contribution was a multiple of 10%, with 19 modules (29%) having no examination (i.e. 100% coursework contribution) and 3 modules (5%) having 100% examination, i.e. no coursework component. When adding in the exam and coursework percentages, we counted exams as any exam which took part during any main exam session. For the purposes of our analysis any class tests etc. were counted as a part of the coursework. For some modules the precise assessment arrangements

All single and joint honours students take a compulsory course where computer programming is a significant learning goal	24 (52%)
Computer programming is optional for all students	5 (11%)
All single honours students take a compulsory course where computer programming is a significant learning goal. This is optional for joint honours	12 (26%)
Computer programming is not taught as part of our mathematics degree programmes	5 (11%)

Table 1. The extent to which programming is taught

0%	19	60%	7
10%	0	70%	10
20%	1	80%	11
30%	0	90%	1
40%	2	100%	3
50%	11		
		Total	65

Table 2. Frequency of proportion of assessment which is examination, by module

were not specified in the module specification and these modules are omitted from our analysis.

We found that only 3 (5%) of computing modules were assessed entirely by closed book examination and 23% used closed book examinations for at least three quarters of the final mark. To put these results in context we note that [25] found that over 25% of the modules in their survey were assessed entirely by closed book examination and nearly 70% used closed book examinations for at least three quarters of the final mark. So, while 66% of modules have a traditional examination for more than half the assessment, computing is disproportionately assessed by a significant coursework component compared to other university mathematics modules. Some 29% of modules are assessed entirely by coursework. The modal form of assessment of computing modules is entirely by coursework.

How is mathematics and computing combined? For example, is computing taught through pure mathematics, applied mathematics or statistics courses? We received full specifications for 65 modules and partial specifications (sometimes the title and level only) for a total of 124 modules. To answer these questions we looked at the module titles of the 124 modules available to us. The first author sorted these module titles, identifying codes shown in Table 3. These codes include three broad categories (1) academic area of mathematics, (2) nature of the course (e.g. workshop), (3) references to programming, including languages used. Having identified these codes each author independently applied as many codes as relevant to each module title and compared their coding of the data. For example a module 'Programming and Numerical Methods' was coded twice to reflect the explicit subject matter and the fact that programming is mentioned in the title.

The final data is shown in Table 3. Our data show that numerical analysis is currently the most common mathematical subject for compulsory courses which involve programming. Statistics is also popular, particularly for optional courses. There were very few explicit pure mathematics courses in our sample and it appears that computing is more popular amongst applied mathematicians and statisticians. There were no

Code	Compulsory	Optional
Numerical analysis/methods	13	11
Probability/statistics	6	24
Modelling	2	5
Calculus/ Other ODEs	5	4
Finance	0	4
Other applied course (e.g. waves)	5	6
Pure course (algebra, discrete mathematics)	4	0
Explicit computer science or programming course, e.g. a title such as 'computer programming', or subject matter such as 'algorithms'	5	6
Only a vague general computational title, 'Computational Mathematics', 'Introduction to Scientific Computation'	11	14
Intensions: e.g. problem solving/skills	5	3
Teaching situation: e.g. workshops/projects	2	1
Explicit programming language mentioned in the title	1	6
Unspecified (module code only)	2	0

Table 3. Coding of module titles

courses on logic, proof or automation of proof in our sample. It is also highly unusual to explicitly mention the programming language in the title of a compulsory course.

5.3. *What programming languages are used and why were these selected?*

We identified 26 year 1 compulsory courses and of these 24 had programming languages specified. A Venn diagram showing the popularity of programming languages in year 1 compulsory programming courses is shown in Figure 2. The intersection of regions in the diagram represents courses where more than one language was taught.

For example, two courses made use of both MATLAB (see `http://www.mathworks.com` retrieved 18 August 2016) and Maple (see `http://http://www.maplesoft.com/` retrieved 18 August 2016) and a total of 6 courses taught Maple. MATLAB was significantly more popular than other languages, naturally corresponding to the more frequent choice of numerical analysis/methods as the subject matter to partner programming. General purpose programming languages, such as Python, Visual Basic (VB) and Excel, are not popular choices with most courses using an identifiably mathematical language such as MATLAB, R (see `https://www.r-project.org/` retrieved 18 August 2016) or Maple.

More generally Table 4 shows the frequency of programming languages used, by individual university (rather than by module). MATLAB is the most popular language used at 28/46 institutions (61%), followed closely by R at 19/46 (41%). Maple is only used by ten, about the same popularity as the general programming language Python. Other languages, including Mathematica and GeoGebra (not strictly speaking a programming language), were used by single institutions.

In terms of staff autonomy, respondents to our survey reported that 'The language is always specified and changes must be approved' 22/42 (52%). For compulsory courses the language was specified and changes must be approved in 10/42 (24%) cases. In only 10/42 (24%) of replies to this question were staff free to choose which language

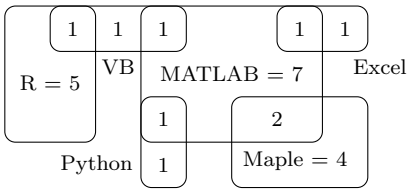

Figure 2. A Venn diagram showing the popularity of programming languages in year 1 compulsory programming courses

Language	**No. universities which teach it**
MATLAB	28
R	19
Python	11
Maple	10
VB (various)	6
C	4
C++	4
Java	4
SAS	4
Fortran	3
LaTex	3
Other[a]	7

[a]Other languages: Fortran95, PHP, Excel, Mathematica, HTML, Geogebra and Xpress.
Table 4. Number of universities who teach each language (compulsory or optional)

to teach. Unsurprisingly perhaps, for core courses the language is normally specified in advance and staff are not free to choose this themselves.

5.4. *Who does the teaching (e.g. mathematics staff or computer science staff?)*

It is relatively standard practice for mathematics staff to teach mathematics to non-specialist students, e.g. in science and engineering, often described as service teaching. We found that teaching programming to undergraduate mathematics students is different. We found that in 25/41 (61%) of the institutions who responded to this question all programming is taught by mathematics staff and in a further 13/41 (32%) cases all compulsory programming is taught by mathematics staff, but students can take optional courses in other departments. In only 3/41 (7%) of departments was there a mix of teaching between mathematics and staff from other departments. In no department who responded was all compulsory programming taught by staff from other departments.

5.5. *Did the department review computing recently and if so what where the conclusions?*

Of the 40 departments who responded to this question 8 (20%) reported a current ongoing review and a further 11 had completed one in 2015, or 2016. Therefore nearly half of the UK mathematics departments have either an ongoing review of computing in the curriculum, or have completed their own review within the last two years. In many cases this review was part of a more general review of teaching and part of regular quality assurance processes. However, no department had a review document which was available for general release, suggesting there is significant duplication of discussion on this issue and confirming our belief in starting this research that such a publication might be of more use generally. One respondent commented

> No School-wide review of [the] role of computer programming — it tends to be done within subject groups. No reports available.

This might suggest even within individual schools or department there is no overall consensus or agreement on what programming should be taught as a compulsory part of the degree programme.

6. Discussion

We believe that our results provide a valuable data set on the state of play of the role of computing in UK undergraduate mathematics education in 2016. Much of the information we sought is (or should be) in the public domain and is published freely on the websites of UK mathematics departments. Nevertheless, it is sometimes difficult to locate programme descriptions or individual module specifications. In some cases institutions choose to restrict this information to current staff and students via password protected websites, even when it is not considered confidential and is freely available on request. Given applicants to UK degree programmes consistently cite information about their prospective course as the most important factor in their choices it is somewhat surprising that a minority of universities do not provide easy

access to information about degree requirements or individual module descriptions. Further, details of course structure (e.g. what is optional, what is compulsory, which courses are pre-requisites etc.) is provided in a different format at each institution. Gathering this factual data in a consistent format and the subsequent analysis, was therefore a substantial task.

Time is limited and so time spent on the development of teaching must, to some extent, displace time available to undertake and publish excellent research, which university academic staff are under considerable pressure to do. Some of our colleagues expressed their frustration openly.

> The problem is that the teaching of mathematics in *** is a complete and utter shambles. Increasingly I feel that no one understands what anyone else is teaching and that there is an emphasis on quantity of teaching rather than quality of learning. On top of this we are also in the midst of a colossal reorganization of the university and everything is thus up in the air in terms of the structure of the course. In terms of module specifications the best I could find is the following attachment, which purports to explain what everyone is teaching in the applied maths and statistics modules here.

It remains to be seen what, if any, effect the UK Teaching Excellence Framework (TEF), a national initiative which seeks to gauge university teaching quality, will have on the teaching and assessment of computer programming. If student satisfaction is a significant component then challenging, but unpopular, courses could come under threat, regardless of their merit. Subjects which are not core mathematics, such as computer programming, might be especially at risk.

MATLAB was the most popular programming language being taught for year 1 compulsory courses. MATLAB had developed from a system for manipulating matrices of floating point numbers efficiently and was designed originally with applied mathematics and numerical simulations in mind. It is therefore not entirely surprising that MATLAB is paired with numerical analysis as the subject matter for mathematical programming. However it is interesting that there were only 10 universities which were using Maple. When we look at [2], Maple was the most popular language. Furthermore, previous research found a wide range of other computer algebra systems, including Derive. In our survey there wasn't a single university which was still using Derive and only one using Mathematica. Previous enthusiasm for computer algebra appears to have diminished, e.g. over 35 years ago proponents of computer algebra were very positive about the potential.

> I propose that computer algebra is an ideal introduction to computer programming language for math, science and engineering students, that it is an ideal principal language for these students, and that the means are at hand for making it fill this role at all levels throughout our educational system. [26].

Why has computer algebra together with pure mathematics as the vehicle for programming lost popularity in UK undergraduate higher education? We have not specifically investigated this question and it remains unanswered here, but given that the research of [5] found that a large proportion of mathematicians use CAS for both research and teaching, we found this surprising.

There were also many universities using non maths-specific languages such as Python, Java, Excel etc.

MATLAB and Maple are proprietary (commercial) products and we found no instances of the open source system Octave (see `https://www.gnu.org/software/octave/` retrieved 18 August 2016), Scilab (see `http://www.scilab.org/` retrieved 18 August 2016) or Maxima (see

`http://maxima.sourceforge.net/` retrieved 18 August 2016) which have very similar goals and functionality. A strong research finding is that to become *a useful tool* a given artifact (e.g. a software package) has to be tightly integrated in practice. See the volumes [27] and [28] and the theoretical work on *instrumentation* such as [29]. If software is only available in laboratories on campus machines it is not surprising that students do not make use of it spontaneously or in other courses.

We found that no university let someone from outside the mathematics department teach the core compulsory programming modules, i.e. this teaching is exclusively undertaken by mathematics staff. This practice is in stark contrast to the teaching of mathematics to engineers. In particular, [23] set out the following principles for the teaching of mathematics to engineers on behalf of the SEFI working group.

(1) Mathematics courses for engineers are best taught by mathematicians.
(2) Mathematics should accompany a student from entry to degree.
(3) The use of computers is an integral part of mathematics courses.
(4) Mathematics courses should be taught with a strong emphasis on applications and projects.
(5) Concentrated mini courses in mathematics should be avoided.

[23, p. vii]

It is interesting to reflect on analogies to these principals as we consider the teaching of computer programming to mathematics undergraduates. *De facto* mathematicians do not agree that computer programming courses for mathematicians are best taught by computer scientists (point 1 in context). In most institutions computing is taught at least in optional courses every year, i.e. from entry to degree (point 2 in context). Our results indicate that there is normally a strong emphasis on applications and projects (point 4 in context). It is interesting that mini courses in computing appears to be the norm (point 5 in context). The consistent evidence from research on the integration of technology in education is that students do not successfully transfer the use of any technology when it is taught only in one isolated course or module. Instead it needs to be an integral part of mathematics courses (point 3). Indeed, much of the thinking and curriculum development of [23] is relevant and instructive when read with the translation mathematics $\rightarrow$ computing firmly in mind.

> Where engineering mathematics teaching is provided by user engineering departments, there may be a risk of omission of general principles, nugatory duplication of others, inexpert and old-fashioned usage, and an emphasis on rapidly ageing training methods of approach at the expense of time-proven principles of education. [23, p. 10]

Computer programming languages come in a wide variety of paradigms, including imperative, object oriented and functional. Notable by their absence in our analysis were functional programming languages, such as LISP or Haskell, which are highly mathematical. Similarly there was also no mention of logic programming, such as Prolog. Both functional and logical programming paradigms have their origins in lambda calculus and logic, see [30]. It is therefore somewhat surprising that they are not currently taught and since they are at best optional, the vast majority of undergraduate students will never encounter these programming paradigms as part of their undergraduate education. Another notable absence in our analysis were interactive theorem provers, proof assistants or any mention of the role of computers in proof. For example Coq is an interactive theorem prover which allows the expression of mathematical assertions, mechanically checks proofs of these assertions and claims to help users to find formal proofs, see [31]. These ideas have been developed for more than a quarter

of a century, indeed Coq was first released in 1989. Again, the vast majority of undergraduate students will never encounter these tools. We did not investigate if these are taught at the graduate level, or widely used by graduate students, but they have yet to make any impression on undergraduate teaching. The lack of undergraduate students with knowledge of (even perhaps knowledge of the existence of) CAS and automatic theorem proving software has implications for the whole mathematical ecosystem: in industry, teaching and future mathematical research. If undergraduate students do not encounter these tools who then were they invented for, and what impact (if any) will they have on the future of mathematical practice?

6.1. *Personal views of colleagues*

As a part of the survey we asked respondents if they had any personal comments on the subject of programming in the mathematics curriculum. 28 colleagues left personal comments in the final question, and in this section we briefly review those comments in the light of our previous discussion. Both authors read these comments independently and conducted an informal thematic analysis, noting again that there has been no attempt to elicit an unbiased and representative sample of colleagues. For this reason we have incorporated these comments in the discussion, rather than the formal results.

Not surprisingly given that these colleagues responded to a survey about computing in the curriculum there was a strong theme in these comments that computer programming is essential (12 comments).

> I think all mathematics undergraduates should take some compulsory programming. I think we offer about the right amount of choice so that students can do significant amounts of programming in optional course units.

One colleague even going as far as to support the initiatives taking place in schools: 'I would say computer programming is essential and in fact should be already core to A-level maths'. Some colleagues echoed the view that programming was an essential part of mathematics, but more cited its future use in employment. In particular, that employers provided a driver for computing within the curriculum (6 comments).

> Almost all our graduates are using some form of programming and, whilst they did not always see the point of it at uni, they are enjoying it in the workplace. It was comments from employers that encouraged us to teach VBA in the new final year option.

However, these comments must be taken in the context that 11% of departments have chosen not to teach programming to their undergraduates. There are a variety of possible reasons. Perhaps there are a group of colleagues in mathematics departments who believe that programming should not be taught, perhaps either because it is not sufficiently important or does not belong in a mathematics degree. Alternatively these departments may not teach computing because of a lack of resources (including staff time), or because of choices about how the resources available best meet their priorities.

Another common concern was the challenge students face in learning to programme, combined with the result that programming was often not popular.

> It is a skill that students would find increasingly useful to have, but one which they appear reluctant to learn. Beyond the initial compulsory module in which programming is taught, take-up of later programming modules is low. In a way I find this surprising given that this generation of students has grown up with computers and computing surrounding them for the entirety of their lives, and yet they still seem fearful of the

programming aspect.

Colleagues also commented on the challenges of teaching, including the practical difficulties associated with staff time when providing small group practical sessions to large cohorts of undergraduate students. A number of colleagues mentioned predisposition, enjoyment and motivation.

> There is a strong split in the cohort between students that take maths precisely to avoid issues like computing, and those that embrace it.

This is a question which deserves to be addressed more closely, particularly in the context of a mathematics degree where programming is a secondary focus. Related to motivation is the issue of transfer of programming skills more generally in the mathematics degree. 'The object really is to use Maple throughout the course, but most students only program if they absolutely have to.' Staff have a critical role in shaping students' beliefs about the importance (or otherwise) of computer programming. Students' beliefs appear to play an important role in how students learn new information: [32]. For example, a meta-analysis of the relationship between self-beliefs and achievement found early self-beliefs correlated positively with later achievement, [33]. One colleague mentioned non-examinable additional courses on offer by the University Information Services which keen students can take to learn programming. Many universities offer such courses, but we have no data on the number of mathematics students who choose to sit them.

Joint honours students often have many conflicting demands on their time, and computer programming (and sometimes laboratories on the science side of the joint degree) are not included as compulsory. Many universities already include programming for joint honours students, and one university has recently moved to include them.

> It's [i.e. computer programming] important, reflected by our move to effectively moved bring teaching of computer programming forward a year in our single honours programme, and to ensure that joint honours/major-minor students also pick up these skills through compulsory modules.

In summary, programming is seen as essential to an undergraduate mathematics degree, both as part of the discipline of mathematics and for future employability. However, computer programming is a challenging subject to teach, with colleagues reporting a split in the cohort between students who enjoy programming and those who strongly do not, some having actively taken mathematics to avoid programming. Perhaps those who already enjoy programming make subsequent use in other courses, while those who did not enjoy programming are keen to minimise their future use outside the formal assessments. In this respect, it is important for university mathematicians to decide whether programming really is an important part of mathematics. If programming is important then this needs to be demonstrated through genuine use for substantial mathematical tasks outside the formal programming modules.

6.2. *Limitations and future directions*

This is a study about curriculum and teaching not about learning and a study of what students actually experience and learn by the end of a mathematics degree would be a separate question. Attitudes to programming have been studied elsewhere, e.g. [32]. We are planning a comprehensive study of students' attitudes to computer programming as part of their undergraduate mathematics degree, which will be a separate and self-

contained study. We have also initiated a longer term study to track any changes in the experiences of computer programming in schools by students at the point they arrive in university. These two studies will complement the factual curriculum data we have presented here.

This study only considered United Kingdom undergraduate mathematics. While this is a limitation in an international context a wider study would have necessitated consideration of the very different undergraduate degree structures which exist elsewhere. For example, the extent to which courses are optional or compulsory, and how assessments are undertaken. Informal discussions with colleagues elsewhere indicate that the role of computing for mathematics students, and the split between computer science and mathematics more generally, are current concerns in many jurisdictions and at all educational levels. This focused study considers one part of this story.

7. Conclusion

We undertook an online survey, with significant follow up correspondence, to gather data on current curricula for undergraduate mathematics degrees in the UK. We received complete replies from 52% of departments and a partial response from 63% of UK mathematics departments, and we conclude that this data set provides a useful summary about the current teaching of computer programming to undergraduate mathematics students. We found that 78% of BSc degree courses in mathematics included computer programming as a compulsory module, however 11% of mathematics degree programmes do not teach programming to all their students. The modal form of assessment of computing modules is entirely by coursework (i.e. no examination), which is not typical of undergraduate assessment. We found no examples of compulsory computing modules being taught by computer science staff as a service subject to mathematics. In 2016, programming is most commonly taught to undergraduate mathematics students through imperative languages, notably MATLAB, using numerical analysis as the mathematical subject matter. Statistics is a very popular choice in optional courses, using the package R. A surprising result is that computer algebra systems appear to be significantly less popular for compulsory first year courses than a decade ago, and there was no mention of logic programming, functional programming or automatic theorem proving software.

Acknowledgements

We are grateful to the colleagues for the time they gave us in responding to the initial survey and follow up requests for further information. We gratefully acknowledge the Centre for Science Education at the University of Edinburgh for the significant contribution to funding this research.

References

[1] O'Shea T, Self J. Learning and teaching with computers. Brighton, UK: Harvester Press; 1983.

[2] Crowe D, Zand H. Computers and undergraduate mathematics: I: setting the scene. Computers and Education. 2000;35(2):95–121.

[3] Crowe D, Zand H. Computers and undergraduate mathematics 3: Internet resources. Computers and Education. 2000;35(2):123–147.
[4] Crowe D, Zand H. Computers and undergraduate mathematics 2: on the desktop. Computers and Education. 2000;37(3-4):317–344.
[5] Lavicza Z. Integrating technology into mathematics teaching at the university level. ZDM: The International Journal of Mathematics Education. 2010 November;42(1):105–119.
[6] Drijvers PHM. Learning algebra in a computer algebra environment. Utrecht, Netherlands: Freudenthal Institute; 2003.
[7] Greiffenhagen C. The materiality of mathematics: Presenting mathematics at the blackboard. The British Journal of Sociology. 2014;65(3):502–528.
[8] Wing JM. Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society A. 2008;366:3717–3725.
[9] Lawson D, Arrowsmith D, Bailey T, et al. Mathematics, statistics and operational research. Gloucester, UK: The Quality Assurance Agency for Higher Education; 2015. Report no.:.
[10] Robins A, Rountree J, Rountree N. Learning and teaching programming: A literature review. Computer Science Education. 2003;13(2):137–172.
[11] Pears A, Seidman S, Malmi L, et al. A survey of literature on the teaching of introductory programming. SIGCSE Bulletin. 2007;39(4):204–223.
[12] McCauley R, Grissom S, Fitzgerald S, et al. Teaching and learning recursive programming: a review of the research literature. Computer Science Education. 2015;25(1):37–66.
[13] Jan Vahrenhold J, Paul W. Developing and validating test items for first-year computer science courses. Computer Science Education. 2014;24(4):304–333.
[14] Herman GL, Zilles C, Loui MC. A psychometric evaluation of the digital logic concept inventory. Computer Science Education. 2014;24(4):277–303.
[15] Taylor C, Zingaro D, Porter L, et al. Computer science concept inventories: past and future. Computer Science Education. 2014;24(4):253–276.
[16] DfE. Computing programmes of study: key stages 1 and 2. London, UK: Department for Education; 2013. National curriculum in England DFE-00171-2013.
[17] DfE. Computing programmes of study: key stages 3 and 4. London, UK: Department for Education; 2013. National curriculum in England DFE-00191-2013.
[18] Upton E. Raspberry pi. Computer. 2013 October;46(10):14–16.
[19] Coe R, Searle J, Barmby P, et al. Relative difficulty of examinations in different subjects. Science Community Supporting Education, CEM Centre, Durham University; 2008. Report no.:.
[20] Lord K, Stripp C. Improving access for state-school students. Mathematics Today. 2015 April;302:80–82.
[21] Noyes A, Adkins M. Reconsidering the rise in A-Level mathematics participation. Teaching Mathematics and Its Applications. 2016;35(1):1–13.
[22] Shadbolt N. Shadbolt review of computer sciences degree accreditation and graduate employability. London, UK: Department for Business, Innovation and Skills; 2016. Report no.:.
[23] Barry MDJ, Steele NC. A core curriculum in mathematics for the european engineer. Société Européenne pour la Formation des Ingenieurs (SEFI); 1992. Report No.: 92.1.
[24] Alpers B. A Framework for Mathematics Curricula in Engineering Education: A report of the mathematics working group. Brussels, Belgium: SEFI Mathematics Working Group; 2013. Report no.:. ISBN 978-2-87352-007-6.
[25] Iannone P, Simpson A. Mapping university mathematics assessment practices. Norwick, UK: University of East Anglia; 2012.
[26] Stoutmeyer DR. Computer symbolic math and education: a radical proposal. Bulletin of the Special Interest Group in Symbolic Algebraic Manipulations of the Association of Computing Machinery. 1979;13(2):8–24.
[27] Guin D, Ruthven K, Trouche L. The didactical challenge of symbolic calculators: Turning a computational device into a mathematical instrument. (Mathematics Education Library;

Vol. 36). New York, USA: Springer; 2005.
[28] Hoyles C, Lagrange J, editors. Mathematics education and technology – rethinking the terrain. (ICMI Study; Vol. 17). New York, Dordrecht, Heidelberg, London: Springer; 2010.
[29] Trouche L, Drijvers P. Webbing and orchestration. two interrelated views on digital tools in mathematics education. Teaching Mathematics and Its Applications. 2014;33(3):193–209.
[30] Doets K, van Eijck J. The haskell road to logic, maths and programming. (Texts in Computing; Vol. 4). London, UK: Kings College Publications; 2004.
[31] Bertot Y, Castéran P. Interactive theorem proving and program development. coq'art: The calculus of inductive constructions. Berlin Heidelberg: Springer; 2004. Texts in Theoretical Computer Science.
[32] Dorn B, Tew E. Empirical validation and application of the computing attitudes survey. Computer Science Education. 2015;25(1):1–36.
[33] Valentine JC, DuBois DL, Cooper H. The relation between self-beliefs and academic achievement: A meta-analytic review. Educational Psychologist. 2004;39(2):111–133.

Appendix A. Online survey questions

(1) [Detailed participation statement omitted]. I consent to take part.

(2) What is your university?

(3) Does your single honours course explicitly mention computer programming in the programme outcomes or as a graduate attribute? Please can you quote from the relevant programme specification or graduate attributes?

(4) Are undergraduate students required to learn computer programming as a core part of their mathematics degree programme?

Please choose an option from

- All single and joint honours students take a compulsory course where computer programming is a significant learning goal.
- All single honours students take a compulsory course where computer programming is a significant learning goal. This is optional for joint honours.
- Computer programming is optional for all students.
- Computer programming is not taught as part of our mathematics degree programmes.

(5) Which programming languages do you teach and to which years (single honours)?

(6) What is/are the module code(s) for courses/modules which are dedicated to programming, or in which programming is an important intended learning outcome?

Are your module specifications available online? If so, please provide links.

If not, please could email module specifications to: [...]

(7) Are specific languages listed in the module specifications or are staff free to choose a programming language?

(8) When did your department last review the role of computer programming in the curriculum? Is a review report available? Please could you email anything relevant to [...]

(9) Do staff from the mathematics department teach programming, or is this taught by other departments as a service subject to mathematics?

Please choose an option from

- All programming is taught by mathematics staff.

- All compulsory programming is taught by mathematics staff, but students can take optional courses in other departments.
- All compulsory programming is taught by staff from other departments.
- There is a mix of teaching between mathematics and staff from other departments.

(10) Do you have any personal comments about the role of computer programming for UG students?

Appendix B. Typical module descriptions

The syllabus of two typical modules is recorded below.

B.1. *University of Bath: XX10190, Programming and discrete mathematics 1*

This year 1 course is worth 12 credits and assessment is by coursework 50%, and examination 50%. Students must have A Level Mathematics grade A or equivalent in order to take this unit.

The role of computing in mathematics. MATLAB programming environment: Editing, importing and exporting data, getting help, functionality, graphics and mathematical applications. Control structures for programming: Pseudocode, selection statements, repetition statements. Elementary logic in programming. Elementary mathematical applications: e.g. summing series. Data types: integers, real and complex numbers. Binary representation. Arrays, user-defined data structures. Procedures, types of variables, parameter passing, scope of functions, functions as arguments. Applications, e.g. to modular arithmetic. Graphs, trees, sorting and searching: applications. Complexity of algorithms, order of an algorithm, examples of algorithms with logarithmic, polynomial and exponential order. Proofs by induction. Recursion and its relation to induction. Mathematical applications of recursion, e.g. Pascal's triangle, Euclid's algorithm, computing fractals. Matrices and matrix operations. Applications, e.g. Fibonacci numbers, modelling networks. Object-oriented programming. The object oriented paradigm, defining classes in MATLAB, encapsulation. Examples from graphics. Software development. The difficulty of formal proof, and the importance of testing. 'black box' versus 'white box' testing. Test coverage, and its importance in MATLAB. Cryptography, RSA. Error correcting codes. Digital signal processing and FFT.

B.2. *Queen Mary, University of London: MTH4105, Introduction to Mathematical Computing*

This year 2 course is worth 15 credits and assessment is by coursework 10% (class test), and examination 90%.

1. Mathematical documents: Running Maple; inputting and evaluating mathematics; writing mathematical documents; palettes, templates and placeholders; Maple help; errors and how to fix them.
2. Introduction to Maple functions: Execution groups and output labels; elementary mathematical functions; floating-point; operators as functions; computational functions diff, int.

3. More Maple functions: The functions sum, product, add, mul, eval, limit; case study Taylor series; case sensitivity and inert functions; subscripts; evalf; simplify, expand and factor; prime numbers and integer factorization.
4. Finite sets, lists and sequences: Names; assignment and unassignment; predicates; set membership and equality; subsets and power sets; union, intersection, difference and symmetric difference; cardinality; binomial coefficients and factorials; lists and sequences; seq and $.
5. Plotting, strings and drawing: 2-D plotting, plot; discontinuities; strings and plot annotation; drawing; multiple plots; plotting points and geometrical plotting; assigning plots; the plots package, implicit plots; 3-D plotting, plot3d; animation.
6. Boolean logic: Logical values and relational operators; lhs and rhs; is versus evalb; Boolean algebra and set theory; De Morgan's laws; the exclusive-or and implies operators; Boolean operators as functions; testing properties of sets.
7. Repeating, deciding and indexing: Statements and terminators; repeating for...from...by...to...do; parallel assignment; generating multi-step recursive sequences; deciding if...then...elif...else...; break and next; for...in...do; indexing; while...do.
8. Defining and using functions: Applying functions to elements of data structures; the domain, codomain and range of a function; piecewise-defined functions; functions on finite sets plotting, equality; injectivity, surjectivity and inverses of functions on finite sets.
9. Relations and divisibility: Relations on finite sets; properties of relations; partitions of finite sets; equivalence relations on finite sets; solving equations and inequalities solve and fsolve; division and divisibility; integer division; integer greatest common divisors.
10. Procedures: Local variables; the return statement; recursive functions; tracing and debugging procedures; computing permutations recursively.
11. Complex numbers: Defining and manipulating complex numbers; complex variables and expressions; De Moivre's theorem; the Argand diagram plotting complex numbers.