RESOURCE REVIEW

Automatic Assessment of Mathematical Programming Exercises with Numbas

Chris Graham, School of Mathematics, Statistics & Physics, Newcastle University, United Kingdom
Email: christopher.graham@ncl.ac.uk
George Stagg, School of Mathematics, Statistics & Physics, Newcastle University
Christian Lawson-Perfect, School of Mathematics, Statistics & Physics, Newcastle University
Aamir Khan, School of Mathematics, Statistics & Physics, Newcastle University

Abstract

As programming has become a common feature of undergraduate mathematics degrees, there has been an increasing focus on how to teach and assess the subject to mathematicians. The potential benefits of e-assessment of basic programming exercises have many parallels with assessment in mathematics where e-assessment tools are widely used: the chance to give instant feedback to students offers an opportunity to allow students to work at their own pace, accommodating the disparate background in programming that often exists in undergraduate mathematics cohorts. And the randomisation of question content not only offers a powerful tool for practice, with students able to repeat similar problems over and over, it also can offer some protection against plagiarism in a subject where, just like a solution to some mathematical problems, student answers to identical problems are likely to be very similar. This paper considers an extension to Numbas to automatically assess programming exercises and the successful implementation of the resource in undergraduate modules using the programming languages R and Python.

Keywords: Assessment, E-Assessment, Programming, Coding, Computing, Numbas

1. Introduction

This paper considers the development of the Numbas e-assessment software to automatically mark programming exercises using the R and Python programming languages, and its application to both practice and summative assessment in two modules in the School of Mathematics, Statistics & Physics at Newcastle University. Section 2 gives some background on the use of programming and the motivation for automatically marking programming exercises. Section 3 describes the new extension to Numbas and how programming questions make use of the well-established features of the system. Section 4 gives more detail on how the new programming feature is used in mathematical programming modules, including the format of assessments and feedback from students.

2. Background

2.1 Computing in the mathematics curriculum

Modules dedicated to computer programming have been a compulsory component of the single-honours mathematics degree programme at Newcastle University since 2015. The addition of computing to the curriculum is in common with many other mathematics departments in the United Kingdom (Sangwin, 2017), motivated by the increasing relevance of computers in mathematical teaching and research, and in the future career prospects of undergraduate students.

At Newcastle University, students take dedicated computing modules at stages 1 and 2 of the mathematics and physics programmes, focussing on R and Python, with computing embedded in many modules later in the degree, such as *Mathematical Biology* and *Big Data Analytics*. At stage 1

of the mathematics programme, students take beginner courses in Python, with a focus on problem solving, and in R, with a focus on statistics, before moving on to a module on numerical methods at stage 2. Physics students follow a similar path, focussing purely on Python, following a move away from MATLAB in 2020.

The increasing focus on embedding programming into the curriculum at Newcastle emphasises the need to establish a solid foundation in the early stages of these programmes. Incoming students typically present with very different experiences and competencies with programming and computer skills in general. Some have formal qualifications, or have self-taught themselves one or more programming languages. These students are likely to find some of the content straightforward and effort is required to keep them engaged, though they typically still require a re-wiring of their programming knowledge in the context of mathematics. Other students have no programming experience or may even demonstrate high levels of anxiety about computing. Establishing a foundation requires accommodation of these disparate backgrounds and the related consequence that they work through teaching material at different rates.

For many years, the differing abilities of students has been most evident in practical sessions. The programming modules follow a structure with a one-hour lecture, followed by a two-hour practical each week, run by the module leader and a team of demonstrators. The lectures are used to introduce theory and new ideas, and give worked examples, whilst the practicals offer the chance for students to get hands-on with the programming language under supervision. This is a popular format, with students citing that they particularly benefit from seeing the module leader work through the process of sketching out algorithms, coding, de-bugging and enhancing solutions in the lecture sessions.

The practical computer sessions follow a handout describing programming commands to try out, with embedded exercises to complete as the handout progresses. Though students appreciate being given the freedom to work through at their own pace, those struggling with the content will often rush to the exercises and find it difficult to get started, often manifesting as very ‘low-level’ queries of the form “How do I start?” or “What does this mean?”, which require little more than direction to the relevant part of the handout. Others will side-step their completion of the exercises completely by gathering in a small group around a friend who is more competent. Although a teamwork approach is desirable, in this case the passive students often lose understanding and go off the trail of the handout content as a result.

What we desire is for the exercises to be accessible to everyone in the class to complete individually, and, although some of these issues can be solved with careful wording of the questions and hints, there remains a fundamental question of how you give feedback to students. At a cohort level, the timing of feedback is difficult: Solutions made available immediately can be counter-productive to students completing the work; going through exercises with the class at intervals during the practical, over a room’s A/V is often mentioned in a positive light by students, but is invariably not at the correct time for most, who will either not have reached the relevant exercise, or have gone far beyond it; and releasing solutions after a practical has finished is also of limited benefit, particularly if the mastering of exercises is essential to progressing through the handout material. Automatic assessment of these exercises affords the opportunity to give individual feedback at the correct pace for the student, and to scaffold questions or offer a hint to those struggling.

Early efforts to introduce e-assessment gave moderate success using the Numbas e-assessment system to indirectly mark exercises (Graham, 2020). Questions were presented to students to complete in the programming software, before entering a numerical value to Numbas, which used its own internal functionality to calculate a solution that could be compared to the student’s answer.

Although this still has a place amongst questions asked in the new approach, it is limited by not being able to directly assess code. The following sections build on that work to mark actual student code.

2.2 Motivation for e-assessment

Mathematical e-assessment systems, such as DEWIS (Gwynllyw and Henderson, 2009), Numbas (Foster, et al, 2012) and STACK (Sangwin, 2015) are well established and can automatically mark procedural mathematical exercises and give immediate feedback to the student. For such exercises, it is possible to establish whether a student's answer is correct, either through mathematical equivalence (the same numerical value or expression as the correct answer) or based on its properties (for example, is a root of a given equation).

A similar idea can be applied to programming exercises: though a student's method of approaching a problem may vary, just as in a mathematics problem, the expected outcome of their code is often well-defined. Consider the following exercise:

> Write a function `is_prime` that takes a natural number as input and returns a boolean: `True` if the integer is prime, and otherwise `False`.

Consider a test applied after a student's answer, for example, `is_prime(13)`, which would return the value `True` if the student's code is correct. A similar test can be applied with several different input values, sufficient to be satisfied that we can infer that the student's code is correct or incorrect.

The approach of running individual "unit tests" on an answer in this way can offer a lot more than this single point of feedback though, and the potential for running multiple tests on a student's code opens the door to rich, individual feedback. We might also ask any, or all, of the following:

- Does the student's code run without errors?
- Does a function `is_prime` exist in the workspace?
- Does the function accept a single value as input?
- Does the function check if the input is a positive integer?
- Does the function return a single, boolean value?
- Does the function give the expected answer for some test input?
- Does the function treat special cases correctly, `is_prime(1)`, for example?

Each of these can be verified with a single unit test and therefore each gives an opportunity to contribute to the marking of the exercise, or to offer feedback to the student, or both.

The idea of automatically marking programming exercises is not new, particularly to the teaching of computer science (Ala-Mutka 2005, Ihantola et al 2010). And in recent years, with the increased emphasis on programming in undergraduate mathematics teaching, tools have been adopted by some mathematics departments. These include Coderunner (Lobb and Harlow 2016), which has been used in undergraduate teaching at the University of Coventry (Croft and England 2020) and on a mathematics programme at University of Edinburgh (Sangwin 2019), and nbgrader (Blank et al 2019), which extends the functionality of Jupyter notebooks.

Whilst these pieces of software are relatively well established, we were motivated to extend the functionality of the mathematical e-assessment software Numbas to accommodate the marking of programming exercises, as described in the next section.

3. The Numbas code extension

Fast-tracked by the global pandemic in 2020, efforts to extend Numbas to automatically mark computer code have now developed into an official *extension* to Numbas.

3.1 Motivation for using Numbas

Whilst the software mentioned in section 2 may offer the functionality to mark and give feedback on computer code, there are good reasons to develop the provision in Numbas itself.

Runs on the client: Numbas is able to run and assess Python and R code entirely in a web browser, with no dependence on a server.

Familiar integration: Numbas is a system that is a familiar to students at Newcastle, used in almost every other module of their studies at stages one and two. The implementation in Numbas means that there is no need to introduce an unfamiliar interface. On a practical level, such assessments can be deployed through the Numbas LTI tool, with no further installation, server requirements or demand on the IT support teams at Newcastle.

Mixing mathematics and programming: Questions are often not exclusively based on programming code – this might be a part of a larger question, or students might be asked to interpret the output of their code. Using Numbas allows for marking code alongside other question types such as number entry or multiple choice.

Access to the many other Numbas features: Perhaps the most powerful motivation for developing an extension to Numbas was to take advantage the many features of the system that are already established and well developed, and which the systems mentioned in section 2 do not offer. These are discussed in depth in section 3.3 and include randomisation, scaffolding questions into steps, alternative answers and adaptive marking.

3.2 How the Numbas code extension works

A Numbas extension provides new functionality or changes the behaviour of Numbas questions. The functionality to mark code sits alongside extensions for statistical functions, interactive diagrams and others, as an official extension developed by the team at Newcastle University.

When included in a question, the programming extension presents students a code input, which uses the open-source Ace code editor ((Cloud9 and Mozilla, 2022), allowing syntax highlighting. Apart from this new type of input, the question interface is familiar to students who have used Numbas before, with a question prompt above the input, a submit button and an area for feedback and marking notes.

The code input box can appear as an empty area for the student to enter their solution, or the question author can give some initial content for the code box. This could be the structure of some code outlined in comments to fill in, or the first part of a solution left to complete by the student. This feature is also used to give students a full piece of code which contains one or more errors for the student to fix, to build their ‘debugging' skills.

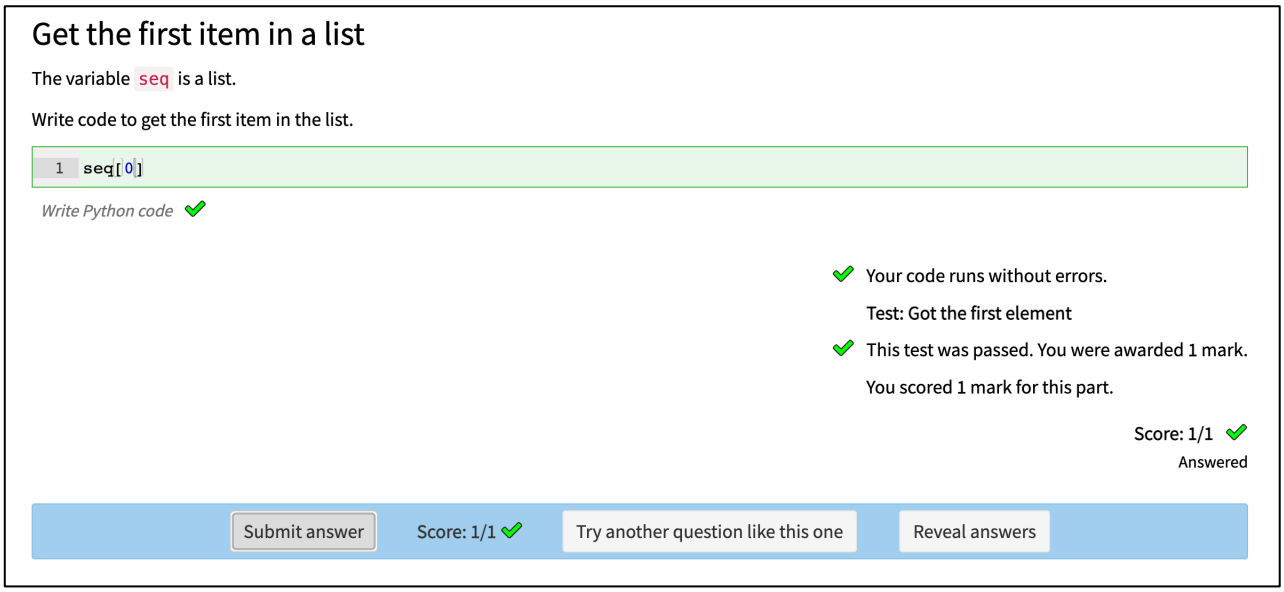


Figure 1. A basic code part, which asks for the first element of a Python list. The question is set up with a built-in validation test, to check if the Python code runs without error, and then a single user-defined marking test, which checks the output against the expected answer.

Early versions of the extension followed a similar process to the software cited in section 2: the student's code and a set of unit tests were sent to a server, to run the code and return the outcome of the tests. Whilst this version was used extensively at Newcastle, the reliance on a server to carry out the tasks makes it a risk, in terms of robustness (if the server gets into trouble and is no longer able to respond to requests, then the assessment can no longer function), and limits the scalability of the set up: the server can only support a limited number of simultaneous users. It also has another significant disadvantage: the reliance on a server in Newcastle limited the ability to share the extension and question content in the same way as other Numbas material, to the wider community of teachers using Numbas.

In the new programming extension, code in R and Python is run in the web browser itself, with no dependence on an external server. This is both desirable, in terms of speed and robustness, and is in keeping with the Numbas project, which runs assessments entirely on the student's device. The code runners Pyodide (Pyodide contributors and Mozilla, 2019-2021) and WebR (Stagg, 2022) are both built using WebAssembly (WebAssembly Community Group, 2022).. WebAssembly is a binary instruction format compatible with most modern web browsers, allowing complex applications, including interactive programming languages, to run in a web browser environment at near native performance.

When the student's code is submitted, the code runner of the appropriate language is loaded. The student's code is combined with other code defined by the question author:

- variable definitions, allowing the student's answer to be marked according to randomisation of a question, as specified by the question author.
- a preamble, to set up anything that needs to run before the student's code, for example variables or functions that they will use in their answer.
- a postamble, executed after the student's answer, to set things up for the marking tests that follow.

- validation tests check that the student's answer is valid, for example to reject an answer that does not define a specific variable or function. These are run after a built-in validation test, which checks whether the student's code runs without error.
- Marking tests, which decide how much credit to give to the student.

The outcomes of the validation and marking tests feed into the Numbas marking algorithm, to apply credit and give feedback to the student. Figure 1 illustrates a basic question using the programming extension. The student receives immediate feedback on their work. They are also able to reveal a correct answer to the question (this feature can be disabled for summative assessment), and a worked solution or explanation can be provided.

3.3 How the code extension uses Numbas features

The functionality of accepting code, marking and presenting feedback is enhanced by a number of features in Numbas which, even though originally designed for mathematics, have very clear applications to programming exercises.

3.3.1 Alternative answers

The question presented in Figure 1 has a highly anticipated incorrect answer: students on our programme study both Python and R, where the indexing of lists and arrays begins from 1, and not 0, as they do in Python. Numbas has an *alternative answers* feature, which can be used to catch, optionally give credit, and provide feedback for specific answers that the author anticipates. In the case of the question in Figure 1, Numbas can give feedback for the case where the student answer retrieves the list element at index 1, as illustrated in Figure 2.

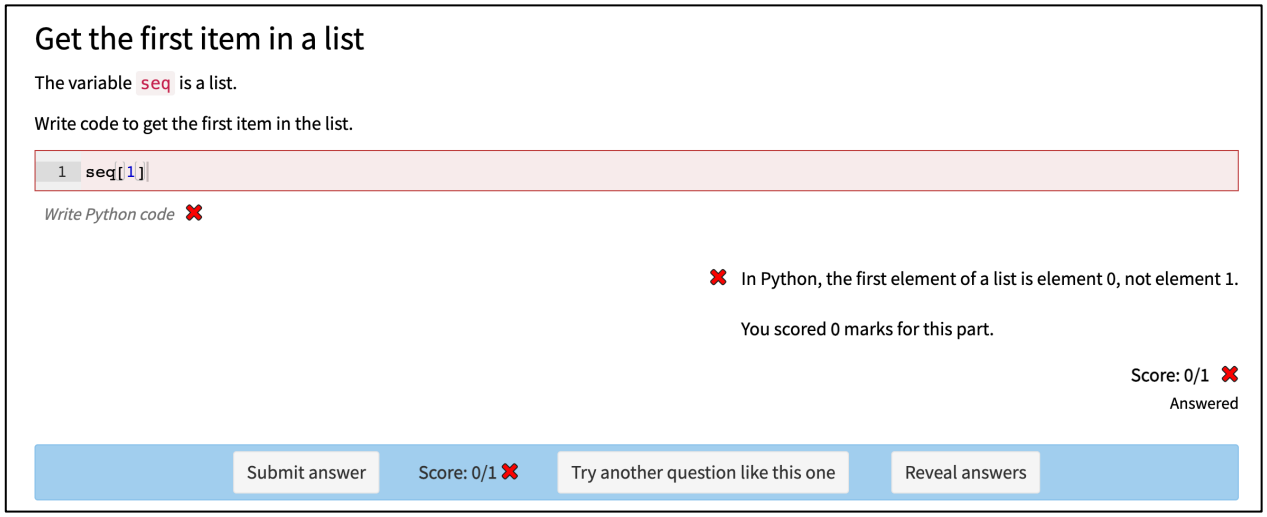


Figure 2. Alternative answer feedback provided for the anticipated incorrect answer to the question in Figure 1, where a student enters `seq[1]` (correct for some other programming languages, including R) instead of `seq[0]`.

3.3.2 Scaffolding using 'steps' and 'explore mode'

Whilst the alternative answers feature can help to give feedback on common errors, other Numbas features assist students struggling to actually get started with a question. These features have been used extensively in the formative material for the modules, including the handout exercises.

The *steps* feature has been a part of Numbas since its inception, inherited from the CALM Project for Computer Aided Learning in Mathematics (Beevers, 2003). Steps allow a basic hint to be presented to the student, for example a reminder of the syntax to use for a particular programming instruction, or of the relevant in-built function to use. Steps can also be used to scaffold a question into smaller chunks, as illustrated in Figures 3a and 3b, for an example which calculates the sum of a series of numbers through operations on numeric arrays. This is particularly useful for questions which require a more substantial block of code to be entered by the student, giving the opportunity to get feedback on each step.

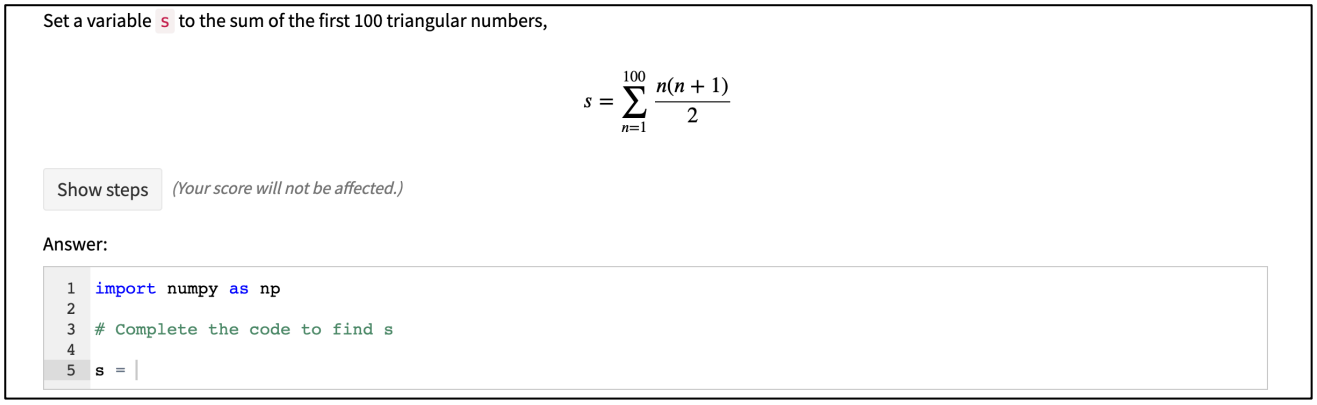


Figure 3a. A question presented as a single answer box, with a *Show steps* button. This sort of question would be typical of a handout exercise where students will often struggle to get started answering the question. The steps offer an "in" to the student and could take the form of a hint or individual answer boxes (Figure 3b).

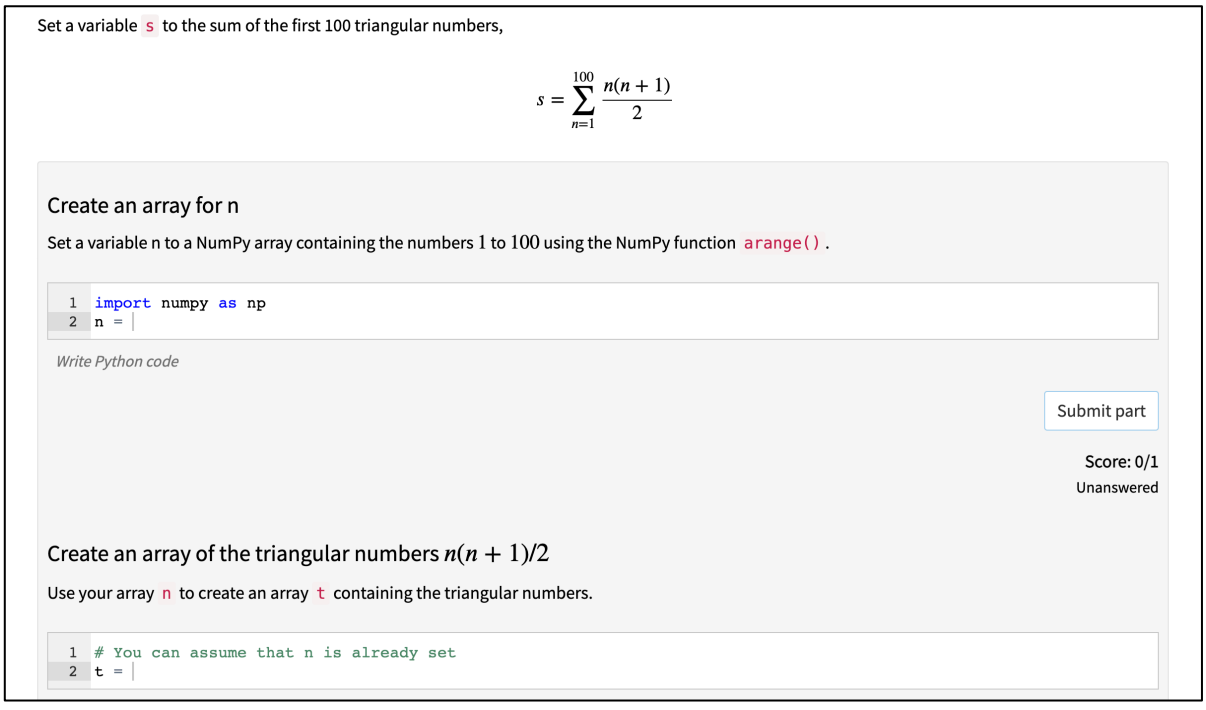


Figure 3b. The steps in this question break the task down into individual one line responses from the student. Each step is a fully-featured code question part which can give feedback to the student using marking and validation tests, and utilise other features such as alternative answers.

A similar approach to scaffolding a question can be made using the *explore mode* feature of Numbas. In this mode, Numbas presents individual parts of a question one at a time to the student, with subsequent parts that can vary depending on the choices made by the student, or their interaction with previous parts. By presenting a question in explore mode, students can be guided step-by-step through a more substantial coding task. In Figure 4, an example is given of an object representing a rectangle, constructed as a *class* in Python, in which the first part of the question asks the student to make a basic class definition. After submitting that part, they can add more code to their question in subsequent parts to add methods to calculate the rectangle's area and perimeter, and to use their class in a practical application. The question uses the variable replacement feature to include the student's code from the first part of the question as the placeholder for the second, and so on, so that they can build up a solution.

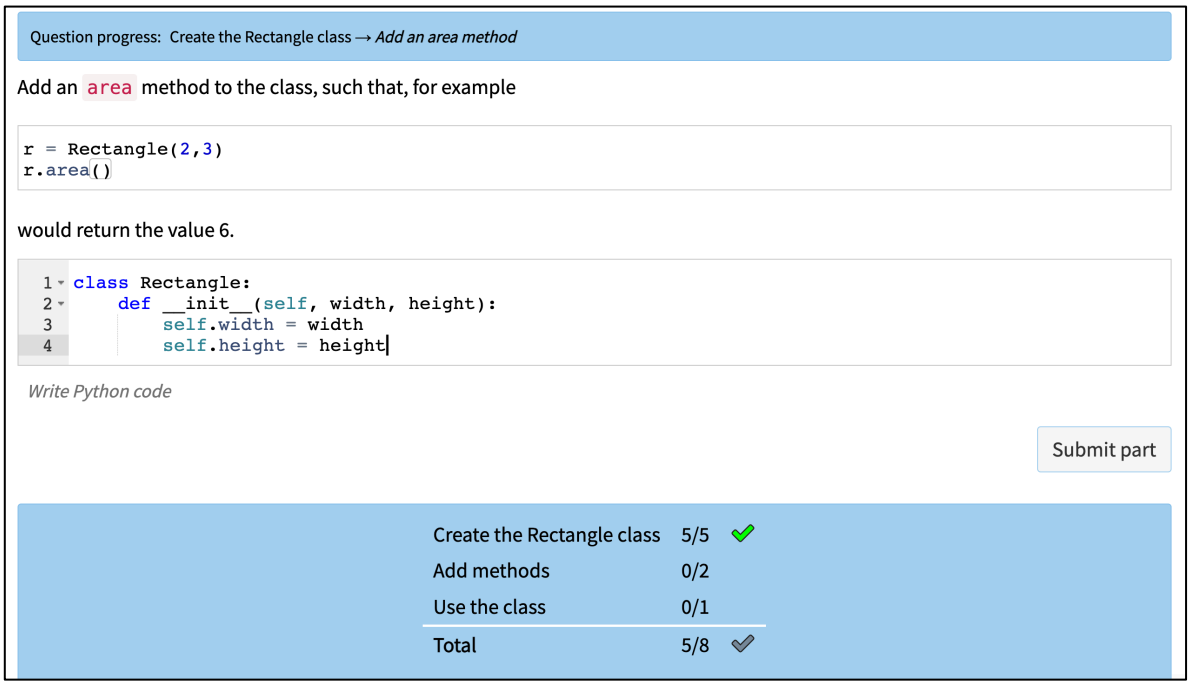


Figure 4. A question on Python classes using explore mode in Numbas. In the first part, the student is asked to create a basic definition of a class for a rectangle. Once they have successfully completed this step, the second part (pictured) asks the student to build on their existing code, adding a method to calculate the area. They can then later move on to add more methods or use the class.

3.3.3 Randomisation

Randomisation is a key feature of mathematical e-assessment, whereby similar questions generated using, for example, a different coefficient of an equation, or numeric value of a property can provide substantial practice for students in a formative mode, or to provide students with different assessment questions, encouraging students to work independently. These motivations are entirely consistent for programming questions, particularly in the context of mathematics.

Randomisation could be different data to work with, or different equations to solve numerically, or even the names of functions or variables. The randomisation of the question itself can make use of the extensive functionality of Numbas, and is passed to the marking test that is applied to the student's code. Figure 5 illustrates a basic example.

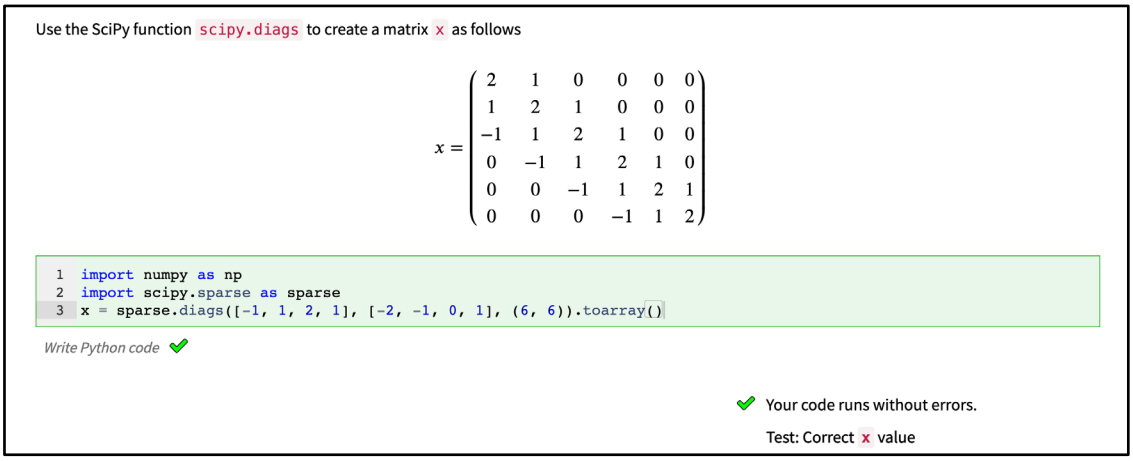


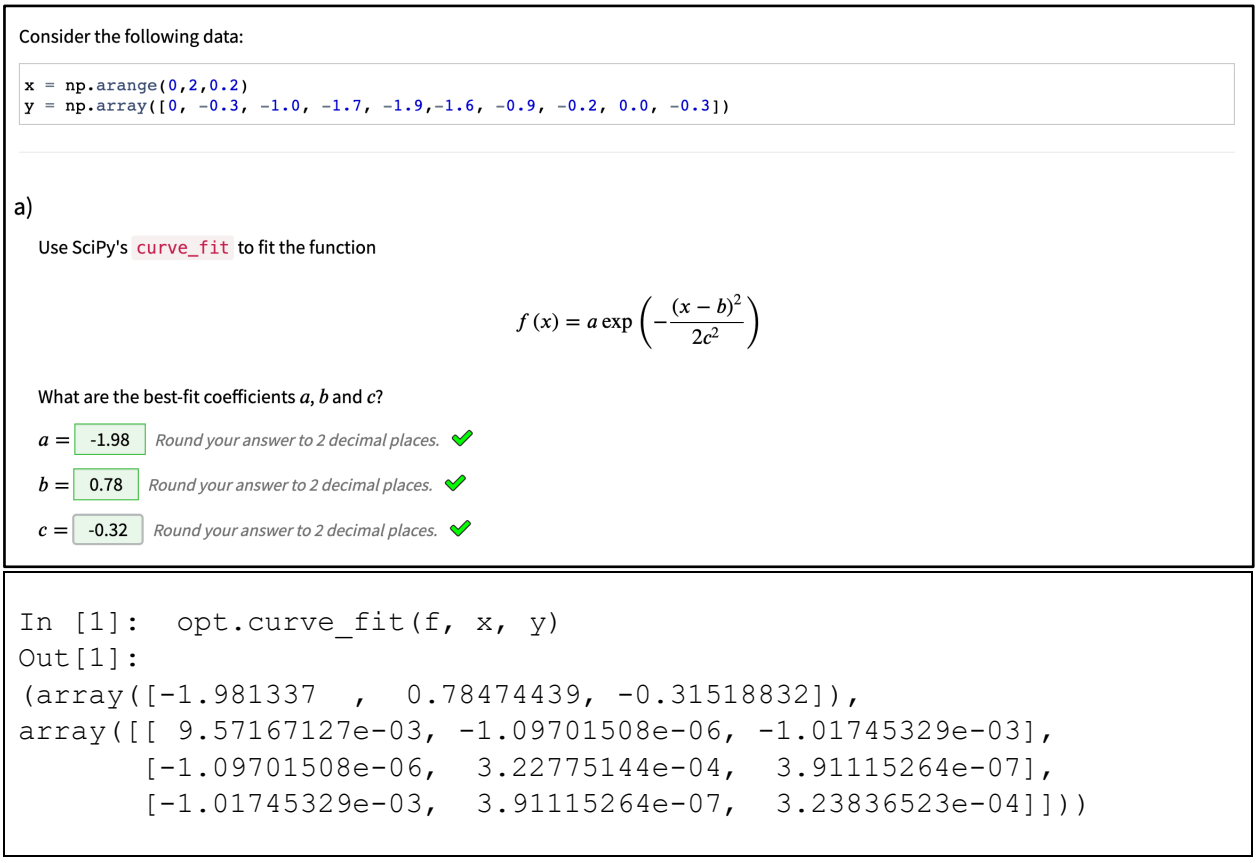
Figure 5. An example of a randomised question. The matrix is randomised to change the values and locations on the diagonals. Different variants of the question can help to reinforce the syntax of the function used to generate the matrix, or to give each student a different version, whilst assessing the same learning outcomes in an equivalent way.

3.3.4 Other part types

Asking for code input is not always necessary or the most appropriate way to assess a programming question. Sometimes a number entry box to accept the output of a computation is a good alternative. In the example in Figure 6, the student is asked for the value of the best fit coefficients of a function, fitted using a Python curve fitting function. In this case, as it is a handout exercise, there is no pressing need to ensure that they have used Python to carry out the task, and by asking for the numeric value it encourages the student to interpret the output of their code. In this case, this requires the student to understand the output of the function, but this could also be critical analysis of whether the code gives sensible values for their problem.

The question in Figure 6 cannot be easily randomised using standard Numbas functionality: it is not practical for a question author to rewrite the algorithm used by the `curve_fit` function to identify the expected answers. There is the option of a fixed question, with hard-coded data values and answers, but the programming extension offers another more sophisticated option: as part of the part's marking algorithm, Numbas can invoke the code extensionprogramming extension to calculate and set the correct numeric answers for the part, by providing it with the code for the correct answer.

In practice, in the application of the programming extension to our modules, many other part types are mixed with the code input, including number entry, mathematical expressions, multiple choice and parts which were marked offline.




```
In [1]:  opt.curve_fit(f, x, y)
Out[1]:
(array([-1.981337   ,  0.78474439, -0.31518832]),
array([[ 9.57167127e-03, -1.09701508e-06, -1.01745329e-03],
       [-1.09701508e-06,  3.22775144e-04,  3.91115264e-07],
       [-1.01745329e-03,  3.91115264e-07,  3.23836523e-04]]))
```

Figure 6. A curve fitting question (top) which asks for numeric values of the best fit coefficients of a given function, rather than the code to obtain them. The question requires the student to interpret the output of the function (bottom), which is a Python tuple, in which the first element is an array of the best fit coefficients, in an order consistent with that specified in their user-defined function (the second value in the tuple being a covariance matrix). Students must understand how the function constructs its output, in order to interpret it and answer the question correctly. In this case, marking the student's code may not be the most approriate means of checking their understanding.

4. Application to programming modules

Used throughout two modules in Python programming in the academic year 2021/22. 100% of question content was delivered via Numbas, though not all assessed automatically.

4.1. Practical handouts

The original motivation for developing a code marking feature to Numbas was to offer feedback on exercise questions in practical sessions. These exercises are embedded inside "virtual handouts", which have in recent years replaced physical handouts and are presented in a web-based format using the Chirun software (Stagg et al, 2022). The format allows students to seamlessly move to an exercise from the relevant handout content.

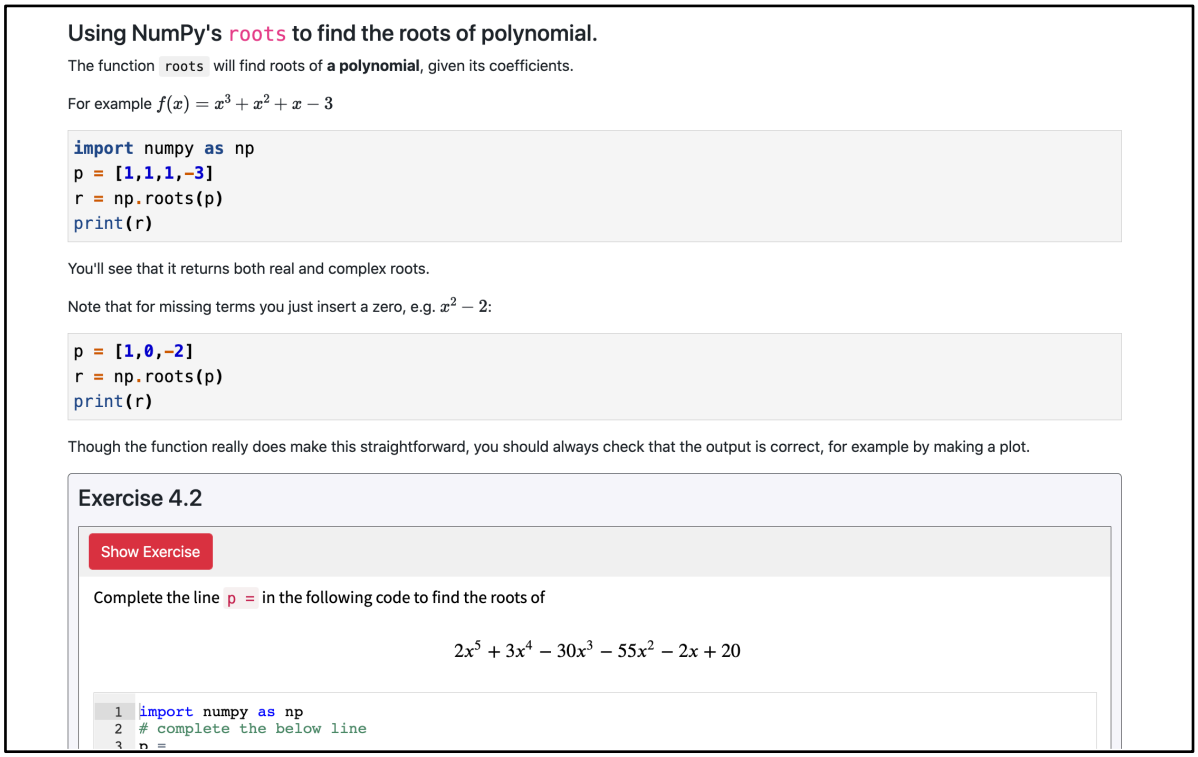


Figure 7. A sample of a "virtual handout" used to deliver practical material. The handouts mix theory, commands to try out, and exercises to complete.

Using the new Numbas extension, the handouts provide instant personalised feedback that students can access at their own pace, with the opportunity to get a hint on a question that they cannot start or break it down into more manageable steps. Whilst the move towards Numbas exercises successfully allowed the modules to be delivered without practical sessions in the pandemic-affected 2020/21 academic year, the most noticeable impact has been on the running of practicals since the return to in-person teaching, where students are more self-sufficient, reducing the low-level queries for demonstrators and allowing them to focus on more meaningful conversations and focussed assistance.

Feedback from students was obtained through two evaluations: the first four weeks into the semester, to capture any early issues and suggestions, followed by a second at the end of the module. In both cases, this took the form of qualitiative, free-text feedback. The format is popular with students:

> *"The handouts strike a good balance between being accessible to the students who've never used Python as well as challenging those who have had more practise. I like the freedom of the practical sessions to work through the handout at your own pace."*

> *"The delivery of the material is interactive and something we can work through and come back to if needs be."*

> *"I like that I can work through the handout so that I'm learning in the best way for myself, at my own pace."*

However, students commented that despite feedback and solutions being available in Numbas, they still like seeing the module leader go through the solutions to handout exercises live, or in a video.

4.2 Practice material

Supplementing each week's handout is a set of formative "Test Yourself" exercises available throughout the semester. These are split into three groups of questions:

- 'Warm up' questions allow an easy route into the material. They might focus on some of the key theory from the week's content presented as questions to remove the coding element. They sometimes involve simple tasks focussing on common errors: for example, students are presented with complete code that contains an error and are asked to make a fix, such that the code gives the expected outcome.

- A group of standard questions that are based on the week's handout material. These were designed to be comprehensive, covering the entire week of material even if questions overlapped with the Numbas handout questions.

- 'Bonus questions' which offer an additional challenge for those who are excelling at the module. These would usually stretch the material beyond the module content, or apply the ideas to something completely left field, for example generating pixel art using the knowledge gained from creating and manipulating 2D arrays.

Engagement with the "Test Yourself" practice material was lower than the practical handout exercises, but very high for optional material, in comparison to other modules on the programme: taking the stage 2 Python numerical methods module as an example, 76% of students tried the first set of Test Yourself exercises, with a steady decline to 50% attempting the later sets, which were perhaps superseded by the release of a mock exam.

4.3. Summative assessment

Each of the modules was structured into three assessments: the first was an assessment open for an extended period, covering the foundations of the respective modules; the second a report-style assessment; the third an off-campus class test. In all cases, assessments were open-book, so as to be more authentic, since students will rarely be programming without access to resources.

A key aspect of the summative assessment was their hybrid format, where parts of some questions presented in Numbas were not marked automatically, rather solutions were uploaded to our institution's VLE for human marking at a later date. These parts typically did not lend themselves to online marking. For example, a question on curve fitting, such as that in Figure 6, might go on to ask the student to plot the data and best fit curve. The presentation element of this part of the question is difficult to mark automatically. Similarly, in the report-style assessment, students were asked to upload their code, and feedback was given on the structure, the code efficiency and other aspects such as the use of comments and appropriate variable names.

The hybrid format was very effective in allowing the marking time to be focussed where it is most impactful, and the response from students in the evaluations was favourable:

> *"The feedback from our assignments was detailed and personal to us and gave us information on what we did well and where we can improve."*

Another Numbas feature used extensively in the summative assessments was the re-marking provision in the Numbas LTI tool, which manages student attempts. Since the introduction of programming assessments is fairly new, it was often the case that alternative approaches, deserving of credit, were identified on inspection of student attempts. The re-marking feature allows assessment questions to be updated, in this case to add additional marking tests, before attempts are bulk re-marked, ensuring fairness in marking across the cohort.

5. Future work

The academic year 2022/23 will see a full implementation of the latest client-side version of the Numbas programming extension in our Python teaching, as well as an expansion of its use for R teaching. A more substantial set of example programming questions is planned for the Numbas Open Resource Library.

6. Resources

A demonstration of the Numbas programming extension is available at:
https://numbas.mathcentre.ac.uk/exam/26300/programming-extension-demo/preview/

7. References

Ala-Mutka, K.M., 2005. A survey of automated assessment approaches for programming assignments. *Computer science education*, *15*(2), pp.83-102.

Blank, D.S., Bourgin, D., Brown, A., Bussonnier, M., Frederic, J., Granger, B., Griffiths, T.L., Hamrick, J., Kelley, K., Pacer, M. and Page, L., 2019. nbgrader: A tool for creating and grading assignments in the Jupyter Notebook. *The Journal of Open Source Education*, 2(11).

Beevers, C.E. and Paterson, J.S., 2003. Automatic assessment of problem-solving skills in Mathematics. Active Learning in Higher Education, 4(2), pp.127-144.

Cloud9 and Mozilla. Ace - high performance code editor for the web [Computer software]. https://ace.c9.io/ [Accessed 1 March 2022].

Croft, D. and England, M., 2020, January. Computing with CodeRunner at Coventry University: Automated summative assessment of Python and C++ code. In *Proceedings of the 4th Conference on Computing Education Practice 2020*, pp. 1-4.

Foster, B., Perfect, C. and Youd, A., 2012. A completely client-side approach to e-assessment and e-learning of mathematics and statistics. *International Journal of e-Assessment*, 2(2).

Graham, C., 2020. Assessment of computing in the mathematics curriculum using Numbas. *MSOR Connections*, *18*(2).

Gwynllyw, R. and Henderson, K., 2009, August. *DEWIS-a computer aided assessment system for mathematics and statistics.* In CETL-MSOR Conference 2008.

Ihantola, P., Ahoniemi, T., Karavirta, V. and Seppälä, O., 2010, October. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli calling international conference on computing education research*, pp. 86-93.

Lobb, R. and Harlow, J., 2016. Coderunner: A tool for assessing computer programming skills. ACM Inroads, 7(1), pp.47-51.

Lawson-Perfect, C., Foster, W., Youd, A., Graham, C. and Stagg, G. 2021. Numbas (Version v6.0) [Computer software]. Available at: https://github.com/numbas/Numbas/ [Accessed 1 March 2022].

Perfect, C., 2015. A demonstration of Numbas, an e-assessment system for mathematical disciplines. In *CAA Conference*, pp. 1-8.

Pyodide contributors and Mozilla. 2019-2021. Pyodide (Version v0.19.1) [Computer software]. Available at: https://github.com/pyodide/pyodide [Accessed 1 March 2022].

Sangwin, C., 2015. *Computer aided assessment of mathematics using STACK.* In Selected regular lectures from the 12th international congress on mathematical education, pp. 695-713. Springer, Cham.

Sangwin, C.J. and O'Toole, C., 2017. *Computer programming in the UK undergraduate mathematics curriculum.* International Journal of Mathematical Education in Science and Technology, *48*(8), pp.1133-1152.

Sangwin, C.J., 2019. *Automatic assessment of students' code using CodeRunner*. IMA Higher Education Teaching and Learning Series. https://maths.mdx.ac.uk/wp-content/uploads/2019/09/2019-6-IMA.pdf [Accessed 1 March 2022]

Stagg, G., Graham C. and Lawson-Perfect, C., 2022, Chirun [Computer software]. Available at: https://github.com/chirun-ncl/ [Accessed 1 March 2022].

Stagg, G. 2022. WebR [Computer software]. Available at: https://github.com/georgestagg/webR/ [Accessed 1 March 2022].

WebAssembly Community Group, 2022, WebAssembly System Interface [Computer software]. Available at: https://github.com/WebAssembly [Accessed 1 March 2022]