CASE STUDY

Using Python in the Teaching of Numerical Analysis

D. Rhys Gwynllyw, Department of Engineering Design and Mathematics, University of the West of England, Bristol, UK. Email: rhys.gwynllyw@uwe.ac.uk.

Karen L. Henderson, Department of Engineering Design and Mathematics, University of the West of England, Bristol, UK. Email: karen.henderson@uwe.ac.uk.

Jan Van lent, Department of Engineering Design and Mathematics, University of the West of England, Bristol, UK. Email: jan.vanlent@uwe.ac.uk.

Elsa G. Guillot, Department of Engineering Design and Mathematics, University of the West of England, Bristol, UK. Email: elsa.guillot@uwe.ac.uk.

Abstract

In this Case Study we describe the rationale, methodology and results of teaching Python as part of a third year optional Numerical Analysis module taken by undergraduate BSc Mathematics students at the University of the West of England, Bristol. In particular we focus on how we have used programming mini-tasks to engage and prepare students for using Python to complete a more significant piece of coursework, taken later in the course. These mini-tasks are marked electronically using the Dewis e-assessment system which provides the students with immediate and tailored feedback on their Python code.

Keywords: Python, programming, e-assessment, numerical analysis.

1. Introduction

In a recent report, Bond (2018) recommended that computer programming becomes a core part of mathematics degrees. For many years, students on the BSc Mathematics course at the University of the West of England (UWE), Bristol used the Maple (2019) computing environment as a combination of a computer algebra system and a programming language. In the first two years of the three year course, Maple was used primarily as a symbolic engine and for visualisation of solutions. A short course on computer programming was included in one of the first year modules but, for the students' first two years of study, their use of Maple as a programming language was limited. However, in their third year, students taking the optional Numerical Analysis module were required to create and develop their own Maple programs.

In 2018 it was decided to introduce Python into the BSc Mathematics course. There were a number of reasons that led us to this decision. Firstly, although Maple can be used as a programming environment/language, it was felt that this was not its primary function. As such, we were not exposing students to a typical programming environment. Secondly, outside of Mathematics higher education, Maple is not a widely known package and hence a student's experience of Maple is not necessarily significant to potential employers. In addition, there is less information and support for Maple online (via GitHub for example, https://github.com/) than for other languages (RankRed, 2019). In contrast, Python has become one of the most popular programming languages (The Economist, 2018) and as with Maple, Python has capability for symbolic mathematics using the SymPy package (Meurer et al., 2017).

Having made the decision to introduce Python into the BSc Mathematics curriculum, we identified a suitable pilot course to be the optional Level 6 Numerical Analysis module. The module's design already accounted for some students' lack of confidence in computer programming by allocating part of the coursework to the completion of mini-tasks throughout the first semester. These mini-tasks provided the students with template code for them to alter to implement more challenging tasks. Keeping a similar structure for the mini-tasks meant that students would be able to get to grips with the basics of Python through a highly-scaffolded approach.

Our move to Python also meant that Dewis (2012) could be used to electronically assess the mini-tasks. Dewis is a fully algorithmic open-source web-based e-Assessment system which was designed and developed at UWE (Gwynllyw and Henderson, 2009). The e-assessment of computer programs had already been used at UWE for a number of years using the Dewis system in a project involving a collaboration of the Mathematics and Computer Science groups (Gwynllyw and Smith, 2018). In that project, the e-assessment of C-programs was performed with the Dewis system marking both the output and the structure of students' computer programs.

2. Methodology

2.1. Overview of the Numerical Analysis module

In this Case Study we considered the 30 credit optional module Numerical Analysis, which is available to final year students on the BSc (Hons) Mathematics course at UWE. The module covers the implementation and analysis of a number of numerical methods applied to a range of mathematical problems and is taught over the whole academic year. Each week, students attend a two hour lecture, and a one hour computing lab. Additionally, all students are timetabled to an optional one hour drop-in session for the module.

The first semester concentrates mainly on the numerical solution of initial value problems (IVPs) and covers the topics of

- Runge-Kutta (RK) and Linear Multistep methods (LMM);
- Error analysis – mostly local truncation error analysis (including adaptive time-stepping);
- Linear stability analysis.

Also in this semester students are introduced to the topic of the finite difference method applied to boundary value problems. The second semester concentrates on numerical solutions to partial differential equations.

2.2. Teaching Approach of Python

The implementation of the teaching of Python started in induction week, where all final year Mathematics students were invited to an 'Introduction to Python' course run by a team of academics from the mathematics group. Those final year students who had chosen the Numerical Analysis module were informed beforehand that participation at this day-long course was considered to be essential. This introduction to Python course was through the Spyder integrated development environment (IDE). This IDE was chosen due to its simplicity and shallow learning curve. Although this introductory course was not written specifically for the Numerical Analysis module, a part of the course content was motivated by the requirements of that module. The overall syllabus of this course was:

- creating and running simple scripts;
- basic commands in Python;

- the process of identifying and correcting bugs;
- variables and data structures (including lists);
- loops;
- logic and conditional statements;
- creating and using user-defined functions;
- importing libraries and files.

For most students this was their first exposure to Python. However all students would have been introduced to programming basics, albeit using Maple, in their first year at UWE.

In the Numerical Analysis module itself, the teaching and practice of Python was concentrated within the first semester. In the second semester, students were supplied with Python code to implement the methods with very little coding changes required on their part. Standard Python was used wherever possible to implement and analyse numerical methods without including additional libraries.

For implementing the actual numerical schemes (RK and LMM), as described in Section 2.1, students were supplied with three files, as shown in Table 1, that they were required to modify.

Table 1. Details of the three Python files provided to students.

File name	Description
main.py	The main program, which sets the numerical parameters, defines the initial value problem and implements the method.
method.py	Contains the function 'calculate' that defines the numerical method. This function receives the problem and numerical parameters and returns arrays containing the numerical results.
output.py	Contains functions that present the numerical results, both as files containing data values and as graphical output. The graphical output uses the 'matplotlib' library.

The Numerical Analysis module introduces students to different aspects of computer programming. For example, throughout the semester, students were exposed to the use of loops, both of fixed number of iterations and, for the case of variable time-stepping, conditional (while) loops. Linear stability analysis exercises required students to determine exponential growth/decay and hence, in producing graphical output, they had to consider the use of log scales. Students were required to use complex variables in their computation of linear stability threshold values (e.g. having expressed a coupled system of real-valued initial value problems as a pair of decoupled, possibly complex-valued, initial value problems). In order to implement the numerical methods, students needed to be proficient in the use of arrays (lists). This involved the case of static arrays (fixed known length typically for the case of fixed time steps) and dynamic arrays (typically for the case of variable time-step methods) and to recognise differences in their manipulation. Together with the use of arrays (lists), students are exposed to the use of list comprehension (constructing arrays in a concise manner in one expression incorporating loops and conditional statements). In the coverage of boundary value problems, students had to identify the representation of a matrix by a list of lists. In addition they were introduced to functions from the 'numpy.linalg' library for solving a linear system. We should note that the Numerical Analysis module was not intended to represent a fully comprehensive course on programming, but instead the important basic programming skills relevant for numerical analysis. New concepts were introduced when relevant to the material being covered.

2.3. Assessment overview

The assessment in this module is composed of an end-of-year exam (75% of the module mark) and a coursework (25% of the module mark). The coursework is partitioned into two parts as follows:

Part One: worth 20% of the coursework mark takes the form of four mini-programming tasks and has four staggered deadline dates throughout the first semester.

The main aims of these mini-tasks are to

- encourage student engagement with the programming throughout the first semester;
- prepare students for the more significant programming task in Part Two of the coursework.

Part Two: worth 80% of the coursework mark is a hybrid written report/programming coursework with a mid-February deadline. Most of this coursework involves a given mathematical problem (typically in the form of coupled IVPs) and a given numerical method. Students are required to construct the numerical method in a form suitable to the problem. They are required to perform both theoretical and empirical analysis of the results. This analysis may include stability and/or local truncation error (LTE) considerations and require the students to comment on unusual behaviours in their results.

2.4. e-Assessment of Python

Given the main purposes of Part One of the coursework (engagement and preparation) we felt it was essential that the marking and feedback for all four mini-tasks to be as fast and supportive as possible (Race, 2014). We had employed this partitioning of the coursework model for a number of years (using Maple). Previously a manual marking process was employed but the workload involved in processing these student submissions, resulted in difficulties in producing timely feedback. In the conversion from Maple to Python, it was decided to implement instantaneous electronic marking and feedback of the Part One mini-tasks in order to address the above problem. The previous deployment of Dewis to e-assess computer code in C (Gwynllyw and Smith, 2018) meant that the development time required for Dewis to e-assess Python was significantly reduced. In fact, most of the Dewis code for marking C programs is the same for Python. Necessary alterations to the code included the implementation of a new set of ‘banned keywords’ for Python, that is, keywords in the student’s submission that were not allowed for security reasons. With Python being an interpreted language, as opposed to C being a compiled language, there were other changes required to ensure that memory and CPU limiters were applied to any execution of the student’s code on the Dewis server. It should be stated here that any such execution is made in a sandbox environment on the Dewis server to protect the server from malicious attack.

To access the mini-tasks, students were directed to log-in to the University’s virtual learning environment (Blackboard) and access the appropriate Dewis assessment for the mini-task (using an automatically authenticated Learning Tools Interoperability link). Students were given three attempts at each of these mini-tasks. The Dewis system included an error-checker on the student’s submission so that, if a student’s code would not run on the Dewis server, then Dewis would report back the Python error report as part of the feedback. In such cases, we instructed Dewis not to decrement the number of submission attempts left for the student. Further details of each of the mini-tasks is included in the next section.

2.5. Particulars of the Python Mini-Tasks

All four tasks in Part One of the coursework relate to numerical methods applied to the solving of the first order initial value problem:

$$\frac{dy}{dt} = f(t,y), \qquad y(t_0) = y_0\,, \tag{1}$$

or a coupled form of (1). Currently these four mini-tasks are as follows:

Mini-task 1: This is the only task that does not require the student to submit Python code. The purpose of this task is simply to ensure that students know how to operate Spyder and understand the structure of the supplied code. The student is given access to Python files for implementing Euler's method to solve (1). The supplied Python files implement Euler for specific values of the problem parameters $f(t,y)$, t_0, y_0 and numerical parameters h and n. On attempting an e-assessment, the student is given different values of these parameters and they are asked to obtain the numerical approximation of $y(t)$ for a given t. Therefore the student is tasked with altering the 'main.py' file (see Table 1), to run their code in Spyder and to recognise which time-stage (i) corresponds to the required value of $y(t)$.

Mini-task 2: For this task students are required to alter the code supplied to them in mini-task 1 so that the code implements the Modified Euler method. Specifically, the student is required to submit to Dewis their modification of the 'method.py' file which contains the function

$$\text{calculate } (f, t_0, t_n, y_0, n). \tag{2}$$

where the system parameters in (2) are as for the initial value problem (1) and f represents a Python function. The numerical parameter t_n is the final time value and n is the number of time steps. These two numerical parameters thus determine the size of the time-step, h.

With regards altering the supplied code, the students were instructed that they needed to only alter the interior body of the 'calcuate' function in (2), that is, they were required to leave the function's parameter listing and return types (the arrays $[t_i], [y_i]$) unaltered.

On submission of their 'method.py' Python code into Dewis, the system performs some security checks and then runs the student's code *four* times using the following instances of the derivative function $f(t,y)$:

$$(i)\ f(t,y) = 0, \quad (ii)\ f(t,y) = c_1, \quad (iii)\ f(t,y) = \cos(c_2 t), \quad (iv)\ f(t,y) = \cos(c_3 t) + \sin(c_4\, y),$$

where $c_1,\ c_2,\ c_3$ and c_4 are constants randomly generated by Dewis.

For each instance (i)-(iv) of running the student's code on the Dewis server, Dewis also runs the corresponding model solution code and compares the $[t_i], [y_i]$ output arrays from the two runs. If the generated arrays have the same values, then the student's code is deemed to have been successful in its running of the modified Euler method for that particular derivative function. However, the student only receives full marks if all four runs are deemed successful.

The purpose of the four distinct runs is to help provide tailored feedback for cases where the student code does not achieve full marks. In such cases, we want students to investigate for themselves the reason for any errors. However, the four runs give a mechanism for suggesting to the student the areas of their error. For example, if the student's code gives correct results for derivative functions (i)-(iii), but not for (iv), the feedback would suggest that the student investigate whether the y dependency in the f function has introduced the error. The student would be advised to check the values they used in the second parameter of any call made to the $f(t,y)$ function.

Mini-task 3: Students are required to alter the code supplied to them in mini-task 1 so that the code implements the Runge-Kutta 3/8 code for a *coupled system*. Specifically, the student is required to submit to Dewis their modification of the 'method.py' file to contain a function

$$\text{calculate } (f,\ g,\ t_0,\ t_n,\ y_0,\ u_0,\ n) \tag{3}$$

to numerically solve the coupled system

$$\frac{dy}{dt} = f(t,y,u); \quad y(t_0) = y_0;$$

$$\frac{du}{dt} = g(t,y,u); \quad u(t_0) = u_0.$$

The students are instructed that function (3) is required to return the arrays $[t_i], [y_i], [u_i]$.

The marking process in mini-task 3 is similar to that in mini-task 2 in that four different runs are performed with different characteristics of the f, g functions in order to facilitate the feedback in the event of any errors. For example, it is only in the fourth run that the case of a fully coupled system is considered, whilst the third run corresponds to the case of two *decoupled* problems, i.e., effectively $f = f(t,y)$ and $g = g(t,u)$.

Mini-task 4: Students are required to alter the code supplied to them in mini-task 1 so that the code implements an Adams-Bashforth-Moulton (ABM) method (3/2-step) together with Heun's 3rd order method as the 'start-up' method (two time-steps). In addition, students are required to implement this method in as efficient a manner as possible with respect to the number of calls made to the derivative function $f(t,y)$.

The marking process in mini-task 4 is an extension of that used in mini-task 2. The same function types are used but, in addition, for the case of errors occurring, Dewis investigates whether the student errors occurred in the Heun start-up method or in the subsequent ABM method.

In addition, each running of the student code is checked for efficiency. In running the student's code, Dewis monitors the number of times the derivative function is called and hence measures the efficiency of the student's code in this regard. The reason for doing this is that, with the ABM method being a linear multistep method, the number of calls to the derivative function can be reduced significantly by storing the most recent values of the derivative within variables (or an array) that is updated at each time-step. This is a desirable approach and one which the students were encouraged to take.

In terms of the marking of this mini-task, three of the five marks were awarded for correct results for the $[y_i]$ array and two of the marks were awarded for the efficiency of the implementation. If a student's scheme was close to optimal efficiency their submission would be awarded one of these two marks.

3. Results

With Part One of the coursework, students were strongly recommended to use the weekly drop-in session to discuss any issues they had with their submissions. The intent of this part of the coursework was to encourage participation and discussion. At any time during which these mini-tasks were open, students could access their Dewis feedback which included a link to retrieve their

submitted code. This facilitated the feedback process with students using this feature to discuss their code submission with the academic on duty.

At the end of the suite of Part One mini-tasks, we investigated all the cases where students' code was in error, to determine whether the feedback they received was appropriate to their submission. This was found to be the case. In addition, we recognised further possible avenues for enhancing feedback which is to be implemented in future years. Further additions to the mini-tasks are planned based on our experience in marking Part Two of the coursework. For example, from marking the 2019/2020 coursework, it seems clear that students struggle to implement a LTE estimator based on Richardson's extrapolation. Hence we plan to include an additional mini-task for obtaining LTE estimators. Our experience of common student errors in the coding of LTE estimators will be used in the design of this task's marking and feedback mechanisms.

4. Discussion

Following the successful introduction of Python into the Numerical Analysis course, it was decided to teach Python, instead of Maple, to our first year Mathematics students in the 2019/20 academic year. Python was taught within an existing Calculus and Numerical Methods course and students attended a two hour computer practical class every week for the first semester. The aim of the course was to learn Python while performing mathematical investigations. Students were introduced to the SymPy, NumPy and Matplotlib libraries which provide extra commands for symbolic and numerical calculations and plotting. Towards the end of the course programming concepts were introduced such as functions, conditional statements and loops.

Python has been used for the two most recent runs of the Numerical Analysis module. For both runs, the final year students have had extensive exposure to Maple at previous levels, but neither have benefitted from our newly introduced Python teaching in the first year. Hence, it is encouraging to note that we have not detected any decrease in student performance nor understanding in programming when compared with previous years' module runs (when Maple was used). This is evidenced by average coursework marks and pass rates for this module being at similar levels to previous years. Student feedback from end of year module evaluations shows that the use of Python has been very positive; students recognise the importance of learning a programming language that is relevant to industry. Students have also stated that they appreciate the Dewis-generated feedback augmented by tutor support in the drop-in sessions.

Having introduced Python in the first year of the BSc Mathematics course the students will develop their skills further through its implementation in second year modules from the 2020/21 academic year. Further, the teaching of Python is embedded into our new problem based learning curriculum which will roll out from 2020/21 onwards; Dewis mini tasks will be used to support students learn Python in their first year, which will enable more challenging tasks to be set throughout the rest of their studies.

5. References

Bond, P. ed., 2018. *The Era of Mathematics – Review Findings on Knowledge Exchange in the Mathematical Sciences*. Engineering and Physical Sciences Research Council and the Knowledge Transfer Network. Available at: https://epsrc.ukri.org/newsevents/pubs/era-of-maths/ [Accessed 4 March 2020].

Dewis Development Team, 2012. Dewis welcome page. Available at: http://dewis.uwe.ac.uk [Accessed 3 March 2020].

Gwynllyw, R. and Henderson, K., 2009. DEWIS: a computer aided assessment system for mathematics and statistics. *CETL-MSOR 2008 Conference Proceedings*. pp. 38-44.

Gwynllyw, R. and Smith J., 2018. E-Assessment of Computer Programming. In *Proceedings of 12th International Symposium on Advances in Technology Education*.

Maple, 2019. Available at: https://maplesoft.com/products/Maple/ [Accessed 3 March 2020].

Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, Š., Saboo, A., Fernando, I., Kulal, S., Cimrman, R. and Scopatz, A., 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3:e103. Available at: https://doi.org/10.7717/peerj-cs.103 [Accessed 10 March 2020].

Race, P., 2014. *Making Learning Happen*. Sage Publications.

RankRed, 2019. *Python Is Now The Second Most Popular Language On GitHub*. Available at: https://www.rankred.com/python-the-second-most-popular-language/ [Accessed 5 March 2020].

The Economist, 2018. *Python is becoming the world's most popular coding language*. Available at: https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-language [Accessed 4 March 2020].