INCORPORATING AN INTEGRATED DEVELOPMENT ENVIRONMENT TO TEACH NUMERICAL METHODS

Desmond Adair

School of Mechanical & Aerospace Engineering, Queen's University Belfast

Kairat Ismailov

CPS, Nazarbayev University

Martin Jaeger

Dept. of Design, Architecture and Civil Engineering, IU of Applied Sciences

ABSTRACT

In this work the rationale, methodology and results of using the computer language Python within an integrated development environment, (IDE) as the method of implementing numerical algorithms derived from the theory of numerical methods is described. This work is part of a First-Year core Numerical Techniques for Mechanical & Aerospace Engineering module at master's level, and the integrated development environment provides an easy-to-use interface to develop, execute and provide results, including appropriate visualizations, so enhancing a student's interaction, and hence learning, with both numerical methods theory and practical skills. It is demonstrated here how Python can first be introduced to students through a short course with intensive exercises, followed by several mini projects of increasing complexity, leading to several major projects. Quick turn-around times and detailed assessment of these tasks are necessary for efficient student development. Student engagement and learning outcomes are evaluated through pre- and post-course surveys and focus group discussions. It was found that students broadly reported the course to be engaging, together with usefulness. As the course is at the introductory stage, care was taken in terms of set-up and design of the learning activities, together with thinking as to how to improve the course in future years.

KEYWORDS

Python, numerical methods, inquiry-based learning, computer-programming assessment, CDIO Standards: 8, 12

INTRODUCTION

The central reasoning for incorporating Python programming into teaching numerical methods is to bridge the gap between theoretical mathematical concepts and practical computational implementation**,** using a language that is both intuitive for beginners and powerful enough for professional scientific computing.

Python is an object-orientated language originating from around the late 1980s as a scripting language (Kiusalaas, 2005) and can be viewed as a fast-emerging language which is still being developed and refined. Programs written in Python are not compiled into machine code but are run using an interpreter. This helps with the speed of execution. Also, the use of an interpreted code means that computer programs can be tested and debugged more quickly so allowing the user more time to concentrate on the principles behind the program. However, a disadvantage of the use of an interpreter is that stand-alone applications are not possible. A further advantage of Python, compared to some well-established computer languages, is that it is open-source software. Now many Linux distributions include Python as standard. In fact, Python can be used on all the major operating systems, i.e., UNIX, Linux, Windows, Mac OS, and interestingly a program developed on one system can be exported to other operating systems without change. This paper reports on the use of Python as an integrated component in the teaching of numerical methods for engineers. Modern numerical methods courses must include a computer programming vehicle to perform calculations based on numerical algorithms derived from numerical methods theory. Numerical methods are techniques by which mathematical problems are formulated so that they can be solved with arithmetic operations (Chapra and Canale, 2002). Numerical methods provide a large and diverse choice for engineering solutions, but these methods all have one common characteristic: they invariably involve large tedious arithmetic calculations. This problem is solved with the use of fast, efficient and accurate computers. The normal process is to use or derive theoretical governing equations, sometimes originally continuous in nature in terms of discrete mathematics. These equations are then used to solve problems directly or are used as a basis for a numerical model (usually with assumptions). Numerical algorithms are then generated and solved using computer hardware and software to give output from given input, and, where it must be recognized and remembered that approximations are being used. Therefore, much effort is employed in numerical methods to ascertain the accuracy of given results.

It can be said that two quite different emphases have been, and to some extent still are, prominent when teaching numerical methods. The first, and one favoured by the European Society for Engineering Education (SEFI) (2002), has been called the ‘just-in-time’ approach. Here numerical methods are taught where the main purpose is to teach the mathematical theory involved. A second approach, and the one preferred here, is to teach numerical methods using specialized courses devoted to numerical analysis (Kaukič, 2005). The reasoning behind adopting this option is because of the current and growing trend of using numerical methods as a foundation for numerical simulations and the ubiquitous presence of computers in almost all engineering specialties.

Application of inquiry-based learning when teaching numerical methods is used in this course. The pedagogical and design decisions to implement inquiry-based learning (IBL) into a more “hands-on” course are driven by the need to shift from passive rote memorization to active, deep learning that mirrors real-world research practices. These decisions are justified to improve student engagement, development of critical thinking, and the acquisition of transferable research skills. Inquiry-based learning has received a great deal of attention (Lappas and Kritikos, 2018) with the focus being mostly on students’ engagement in active

discovery learning and learning in environments which include some active participation, self-action, observation, exploration and experimentation (Psycharis, 2011; Kyrizis *et al*., 2009; Gormally et al., 2009; Justice et al., 2007; Richardson and Liang, 2008). It can be argued that numerical methods, when the use of computers is employed, is ideally suited to incorporate some and indeed all of these inquiry-based attributes. The opportunity to learn numerical methods effectively does depend on a wide range of factors, but among the most important are those associated with activities and practices within the education process (Martin-Caraballo and Tenorio-Villalón, 2015; Olagide, 2014). Details of programming pedagogy and self-efficacy relevant to this work are also found in the literature (Pham *et al*., 2014; Reng and Kofoed, 2012; Lai *et al*.,2025).

For the current numerical methods course, a didactic framework based on the Process of Scientific Inquiry (PoSI) (Lappas and Kritikos, 2018), which has three main components each using a particular cognitive tool, was used. The framework has the three general aims of: helping with understanding basic aspects of numerical methods; providing space for practice of computer and critical-thinking skills; and, to give an understanding of scientific research (Lappas and Kritikos, 2018). Within these aims lie the more specific important goals of understanding important mathematical concepts like stability and convergence of solutions; selection of the best appropriate numerical method to suit the problem in question; developing and implementing a suitable algorithm followed by testing and debugging. The IDE is heavily integrated into achieving all of these aims when writing computer programs for developed algorithms.

Regarding assessment, theoretical aspects of the course were assessed in a traditional way using quizzes, assignments and a final examination. However, consideration was given in how to combine effective and efficient assessment of computer programming work, mainly in the form of assignments. It was decided to carefully incorporate a form of peer reviewing into the computer programming assessment scheme.

In line with the CDIO 2026 conference theme, this work contains development of professional engineering competencies, skills and attitudes. Integrating numerical methods with computer programming significantly enhances professional development by bridging the gap between theoretical knowledge and practical application across fields like engineering, finance, and data science. This synergy equips professionals with the ability to solve complex, real-world problems that lack exact analytical solution.

The integration of numerical methods and computer programming is also in line with CDIO competencies where technical knowledge and skills are greatly increased with problem - solving and critical thinking coming to the fore. Other competencies used in the design of the course include life-long learning, effective reporting and teamwork. The course offers hands-on learning, integrated learning and active learning techniques.

The sub-theme of simulation and modelling found in numerical methods combined with programming, particularly when integrated with innovative assessment and feedback, enables a "learn-before-doing" approach, allowing students to experiment in low-cost, risk-free environments before engaging with real-world physical systems. This approach acts as a bridge between theoretical knowledge and professional practice.

USING THE INTEGRATED DEVELOPMENT ENVIRONMENT

Python Environment

The Jet Brains PyCharm Community Edition 2021.1.2x64 integrated development environment (JetBrains, 2021) was used for all computer programming work in this course. The Windows installation of PyCharm was used here. The first part of this course was to familiarize the students with the IDE and to show its possibilities. This was achieved with a demonstration followed by very simple exercises. At this point such common, and useful, Python tools such as `numpy` (a library for the programming language which offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more), and `matplotlib` (a comprehensive library for creating static, animated, and interactive visualizations in Python) were introduced to the students.

Some Preliminary Remarks

The students taught in this course did not have much prior programming experience, and what they did have was usually using MATLAB. However, it was found that the students who had used MATLAB found it quite easy to adapt to the similar syntax of *numpy* with the help of Python tutorials geared towards engineering users. It was found that a 2-hour laboratory session was sufficient to acquaint the students with the basics although it was necessary to introduce further topics as the course progressed. During this time students were actively encouraged to cooperate and even collaborate with each other to help with their computer skills.

Generally, the students had been introduced to a given algorithm during a lecture period, and this was then reinforced with brief notes containing explanations and exercises during the computer laboratory sessions. During the early stages of the course most of the code needed to complete exercises was made available with students having to change fragments here and there or extending the provided code. However, the amount of code provided became progressively less as the student programming skills improved, and eventually students were asked to implement algorithms of subroutines from scratch. Occasionally quizzes were used to provide incentive and to get feed-back that students were approaching the work in a systematic way.

How much assistance to give to students was also considered. It was important to give students the freedom to fix their code when it failed, even struggle a little to try to find a solution. When assistance was given, it was of a type where the instructor asks questions to give guidance as to how to trace-back and diagnose their code. The main method of instruction during the laboratory periods was one-to-one, as the cohort was only 15 students and this proved very effective. During these periods students were also expected to give explanations of their coding strategy in brief 5-minute presentations to generate discussion and ideas.

The delivery strategy fell into three main areas: intensive exercises; mini projects and finally several major projects. Some details of these three components are now described.

Intensive Exercises

Preliminary intensive exercises were used to introduce common programming procedures necessary to solve numerical problems. Throughout these exercises comprehensive notes and almost complete programs were given to the students together with problems where students simply had to change the given code to affect a solution. The instruction staff were

fully engaged in helping students with well-placed questions and suggestions. The exercises fell neatly into three main areas with the first containing: how to use variables; the use of library functions; and vectorization and plotting.

When instructing how to use variables, a very simple mathematical model was first chosen for introduction, for example, the prediction of the vertical position z of a ball at time t using the equation

$$z = v_i t - \frac{1}{2} g t^2 \tag{1}$$

where v_i is the initial upwards velocity and g is the local acceleration of gravity. The students were given a Python program which solved this model and the program was dissected line-by-line. The students were encouraged to vary the variable to get different results. The use of library functions was introduced with an explanation that much functionality exists in Python libraries, but to activate this functionality the functions must be explicitly imported. An important component of this area was plotting. At this stage only *x-y* plots were introduced as well as necessary libraries to effect plotting. Next to be introduced was the formatting of text, numbers, and arrays, and the printing and input of data.

During the intensive exercise period the very important group of topics were introduced, i.e.: `If` tests, colon, indentation; Python `For` loops; Python `While` loops; and Python Lists.

Mini projects

Some eight mini projects, which were carefully graded regarding the level of difficulty, were used during this course. To give an idea of a typical mini project this sub-section gives a description of one.

Mini Project - “An Oscillating One-Dimension System”.

Many engineering scenarios use oscillating springs. To illustrate this a simple system modelled using the general second-order differential equation was used

$$u''(t) + \omega^2 u(t) = 0 \tag{2}$$

where $u(t)$ is the unknown variable at time t and ω is a physical parameter. No damping is imposed on this system, and to complete the model two initial conditions must be given. A numerical scheme to effect solution has now to be chosen. One way is to write the governing equation as a first-order system of two differential equations and introduce $u = x$ and $v = x' = u'$ as two unknown functions. This gives

$$u' = v, \quad v' = -\omega^2 u \tag{3}$$

The Forward Euler method can now be applied for the differential equation system which results in the solution scheme

$$u^{n+1} = u^n + \Delta t v^n, \quad v^{n+1} = v^n - \Delta t \omega^2 u^n \tag{4}$$

Using Python, the solution scheme was coded together with the exact solution. Calculations were conducted using 20 intervals per period resulting in solutions for the exact and numerical calculations as shown on Figure 1.

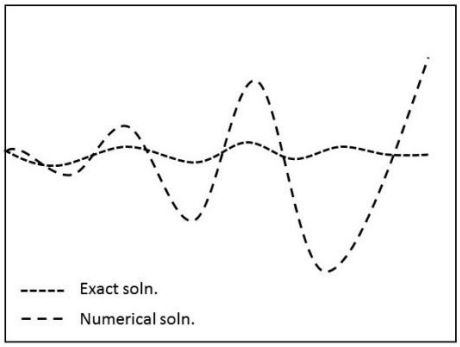

Figure 1. Simulation of an oscillating system.

It can be seen there is very little similarity in the solutions. This discrepancy gives the opportunity for a good learning experience in that, is it due to a programming error, or is there a problem with the numerical solution method. Even though everything was developed logically and seemingly correctly, the two solutions do not match. So how was this rectified by the students? First the computer program was tested against some hand-calculations, but everything appeared correct. The students were then asked what could be changed without altering the numerical solution method. It became clear that the choice of 20 intervals per period was arbitrary and could therefore be adjusted. Therefore, this number was steadily increased, and better correspondence of the two solutions resulted.

Major Projects

Part of the course assessment consisted of two major projects, each of which consisted of assembling the correct equations for solving, assigning boundary conditions, developing the algorithm, writing the solution code and finally writing code for the post-solver visualizations. As an example, the following outlines one of the projects used.

Major Project: “Calculate and Visualize the Characteristics of a Lid-Driven Cavity Steady-State Flow using the Streamfunction/Vorticity Approach”.

The project takes a student through several necessary stages in the development of a suitable numerical method. These are: writing the Navier-Stokes equations in the streamfunction and vorticity form; stating suitable boundary conditions at the walls of the cavity in terms of vorticity and streamfunction; discretization of the transport equations suitable for a Cartesian grid; derive a solver; and display the results.

The first task involves defining vorticity and streamfunction followed by the derivation of the Navier-Stokes equations for incompressible steady-state flow in the vorticity-streamfunction form where ω is the vorticity and ψ is the streamfunction.

The next major task for the students is to discretize the transport equations and write a solution algorithm. The method chosen was that due to Burggraf’s (1966). To include the boundary nodes of the cavity, central differencing is used to discretize the non-dimensional governing equations and the number of nodes in the *x* and *y* directions was set at *nx* and *ny*, respectively. The mesh used by the students was a regular Cartesian grid with the nodes equally spaced

at a distance *h*. Eventually the students are to arrive at the non-dimensional advection-diffusion and elliptic equations. The solution method uses residual functions; that is, if the values of ψ_{ij} and ω_{ij} are exact on the nodes spanned by the residual functions $\mathcal{R}_{ij}$ and $\mathcal{L}_{ij}$, then

$$\mathcal{R}_{ij}^{k+1} = \mathcal{L}_{ij}^{k+1} = 0 \tag{5}$$

where

$$\mathcal{R}_{ij}^{k+1} = \frac{1}{2(1+\gamma^2)}\left(\psi_{i+1,j}^{k} + \psi_{i-1,j}^{k+1} + \gamma^2\left(\psi_{i,j+1}^{k+1} + \psi_{i,j-1}^{k+1}\right)\right) + h^2\omega_{ij}^{k} - \psi_{ij}^{k} \tag{6}$$

and

$$\mathcal{L}_{ij}^{k+1} = \frac{1}{2(1+\gamma^2)}\left(\left(\omega_{i+1,j}^{k} + \omega_{i-1,j}^{k} + \gamma^2\left(\omega_{i,j+1}^{k} + \omega_{i,j-1}^{k}\right)\right)\right. \tag{7}$$

$$\left. -\frac{Re}{4}\gamma\left(\left(\psi_{i,j+1}^{k} - \psi_{i,j-1}^{k+1}\right)\left(\omega_{i+1,j}^{k} - \omega_{i-1,j}^{k+1}\right) - \left(\psi_{i+1,j}^{k} - \psi_{i-1,j}^{k+1}\right)\left(\omega_{i,j+1}^{k} - \omega_{i,j-1}^{k+1}\right)\right)\right) - \omega_{ij}^{k}$$

where γ is the ratio of width to height of the cavity and Re is the Reynolds number. These equations can be solved using a fixed-point iterative procedure, based on the Gauss-Seidel scheme and as the equations have been discretized using central differencing much work is also necessary to give the final boundary conditions.

COURSE FEEDBACK

Student Feedback and Experience

Two surveys, here referred to as the pre-course survey and the post-course survey, were conducted to assess any change in student self-efficacy and interest in the inclusion of Python as suitable for solving numerical methods. The sample size in each survey was $N = 15$. Included in the post-course survey questions were those needed to assess the design and deliver of the course. As well as the surveys, focus group discussions were held at the end of each session of the course to elicit further details concerning the content. The verbal data collected during the focus group discussions was documented and used in conjunction with the survey data to give a deeper understanding of student responses.

The survey instrument design was intended to get feed-back concerning pre-course instructions, high activation barriers, and confidence concerning using programming with numerical methods. The scales used were of a Likert type, with some expanded beyond the usual five-point scale, to try to lessen the effect of subjectivity. Each focus group consisted of five students. The first few moments of the discussion were critical. In a short time, the moderator created a thoughtful, permissive atmosphere, provided ground rules, and set the tone for the discussion. Much of the success of the group interviewing was attributed to the development of this open environment. The pattern for introducing the group discussion was: welcome, overview of the discussion and ground rules, followed by the discussion.

Informed consent was obtained from each student involved and from the university ethics committee. The participants confirmed they understood the process, why their participation was required and in what way the research findings would be used and reported.

In this section the analysis of student feedback both verbal and from the survey data is divided into three sub-sections: Introduction to Python and code familiarization; content and structure of the parts of the course involving Python; and lastly reaction to how the Python computing language was delivered and integrated into the course. Extensive notes with annotated screenshots as to how to set-up the programming environment were compiled and the student reaction to the comprehensiveness and ease of following these documents is summarized on Figure 2.

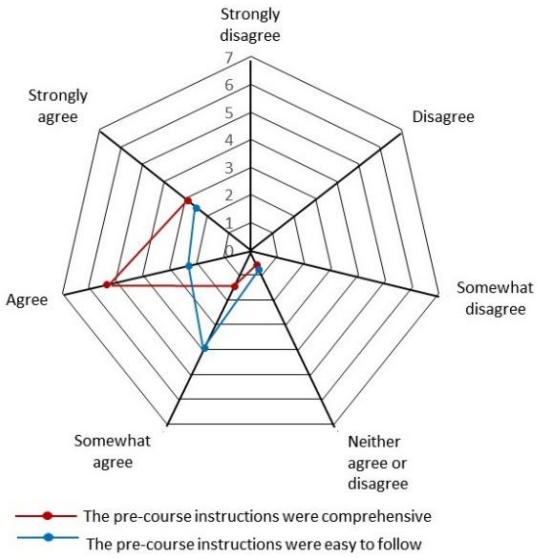

Figure 2**.** Student reaction to questions related to the pre-course instructions and set-by-set guide for setting up the PyCharm IDE.

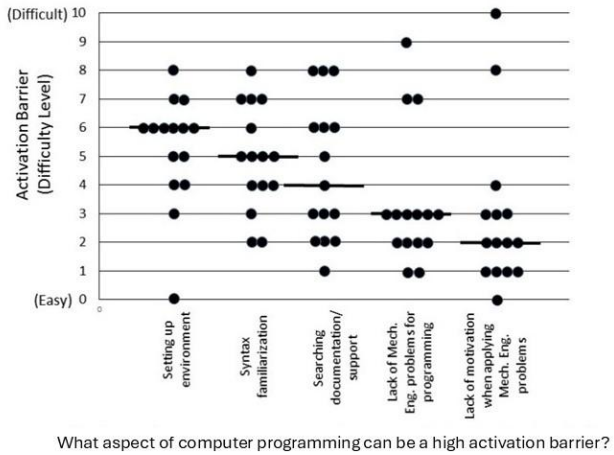

Figure 3. Student assessment of what they consider high activation barriers: 0 here means – no barrier, and 10 means - a high activation barrier. The horizontal solid lines represent the medians.

There is often a point reached in the early stages of programming where frustration encroaches for the programmer, even when writing the simplest of codes. Of course, such

experiences can be very instructive, but they can also be very de-motivating for beginners. It was important therefore to try to ascertain from a student's point of view as to what they saw as "activation barriers/difficulties" associated with beginning and applying Python programming. The results of the surveys are shown on Figure 3. As can be seen there was a fair spread of opinions on each of the proposed barriers with setting up the environment and syntax familiarization thought of as giving high barriers to programming. It is clear therefore that more care will be needed in future to reduce this problem and create a more enabling environment for experiential learning (Wood and Bhute, 2019; Shah *et al*., 2020).

On Figure 4 is shown results for how the students would regard incorporating Python into the numerical methods calculations. The results shown on Fig. 4 are encouraging as the students generally stated that their confidence in using Python computing language veered mostly in the positive direction.

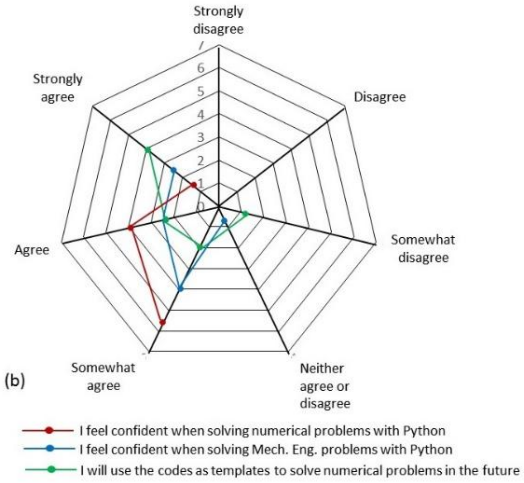

Figure 4. incorporating Python into the numerical methods calculations.

Regarding the delivery of the course, some students expressed a desire for more active and structured learning components which related to the actual code and running of simulations. The students did like formal and structured questions rather than being left to explore too much. However, every opportunity was still maintained to encourage the students to go deeper into a given problem.

CONCLUSIONS

The use of PyCharm as a suitable integrated development environment for the introduction of the Python programming language to students with very little coding experience has been demonstrated. The course was designed so that each exercise was built on previously taught material which help with delivery efficiency. The open-source nature of PyCharm meant that availability is not an issue, and it is recognized that Python itself is a rapidly developing language, and new libraries must be continually reviewed. Also, the students are developing skills in programming which are immediately transferable to later employment in industry and academia. Within this course there was a good variety of problems (projects) with different complexities which could, without much resistance, be given to the students to help with a variety of learning objectives. There was an improved student perception of Python post-course, and confidence and self-efficacy improved with much satisfaction. It is recognized that the student surveys were conducted with a small sample size, and further studies will be important to assess whether the findings here are transferable to a much larger cohort.

ACKNOWLEDGEMENTS

The authors received no financial support for this work.

REFERENCES

Kiusalaas, J. (2005). *Numerical methods in engineering with Python*. Cambridge University Press.

Chapra, S. C., & Canale, R.P. (2002). *Numerical methods for engineers*. 4th ed. McGraw-Hill Higher Education.

SEFI Math. working group, (2002). *Mathematics for the European Engineer*. SEFI HQ.

M. Kaukič, M. (2005) Open-source software resources for numerical analysis teaching, *Int J for Mathematics and Learning*, ISSN1473-0111.

Lappas, P.Z., Kritikos, M.N. (2018). Teaching and learning numerical analysis and optimization: A didactic framework and applications of inquiry-based learning. *Higher Education Studies,* 8(1), 42-57.

JetBrains PyCharm. Python IDE–PyCharm by JetBrains URL: https://www.jetbrains.com/, (2021).

Burggraf, O.R. (1966). Analytical and numerical studies of the structure of steady separated flows. *Journal of Fluid Mechanics*, 24(1), 113–151.

Pham, A.P. et al. (2014). Learning computer programming in CDIO's team settings. 10th *International CDIO Conference*, Barcelona, Spain, June 15-19.

Reng, L. Kofoed, L.B. (2012). Enhance student's motivation to learn programming the developing process of course design. 8th *International CDIO conference*, Queensland University of Technology, Brisbane, July 1-4.

Lai, C-H., Ho, L-C., Liao, Z-Y. (2025). Sustainability of programming education through CDIO-orientated practice: An empirical study of syntax-level structural visualization for functional programming language. *Sustainability*, 17(12), 5630.

Psycharis, S. (2011). The computational experiment and its effects on approach to learning and belief on physics. *Computers and Education,* 56930, 547-555.

Kyrizis, A., Psycharis, S., Korres, K. (2009). Discovery learning and the computational experiment in higher mathematics and science education: A combined approach. *International Journal of Emerging Technologies in Learning*, 4(4), 25-34.

Gormally, C., Brickman, P., Hallar, B., Armstrong, N. (2009). Effects of inquiry-based learning on students' science literacy skills and confidence. *International Journal for the Scholarship of Teaching and Learning*, 3(2), 1-22.

Justice, C., Rice, J., Warry, W., Inglis, S., Miller, S., Sammon, S. (2007). Inquiry in higher education: Reflections and directions on course design and teaching methods. *Innovative Higher Education*, 31(4), 201-214.

Richardson, G., Liang, L. (2008). The use of inquiry in the development of preservice teacher efficacy in mathematics and science. *Journal of Elementary Science Education*, 20(1), 1-16.

Martin-Caraballo, A., Tenorio-Villalón, A. (2015). Teaching numerical methods for non-linear equations with GeoGebra-based activities. *Mathematics Education*, 10(2), 53-65.

Olagide, A. (2014). Teaching numerical analysis to non-mathematics major students, in: *Proceedings of the International Conference on Science, Education, Arts, Management and Social Sciences*, Afe Babalola University, Ado-Ekiti, Nigeria May, 939-942.

Wood, P.M., Bhute, V. (2019). Exploring student perception toward online homework and comparison with paper homework in an introductory probability course. *Journal of College Science Teaching*, 48(5), 68-75.

Shah, U.V., Chen, W., Inguva, P., Chadha, D., Brechtelsbauer, C. (2020). The discovery laboratory part II: A framework for incubating independent learning, *Educ.Chem. Eng*., 31, 29–37.

BIOGRAPHICAL INFORMATION

Desmond Adair obtained his PhD in Mechanical Engineering from Imperial College, University of London and is now working as a Teaching Fellow in SMAE, Queen's University Belfast. His current research interests include renewable energy and engineering education.

Kairat Ismailov has got his doctor degree in Nuclear Engineering from Tokyo Institute of Technology (Japan) in 2011 and is working at the Centre of Preparatory Studies at Nazarbayev University as Physics and Math Teaching Fellow of pre-master program. His current research topics are nuclear reactions, modelling of the thermal management of the Li-ion batteries.

Martin Jaeger holds a PhD in Civil Engineering from the University of Wuppertal, Germany, and worked as site manager, consultant, and lecturer in Germany and the Middle East. He is a professor of civil engineering with the IU International University of Applied Sciences, a certified international engineering educator (Ing.Paed.IGIP).

Corresponding author

Desmond Adair
School of Mechanical & Aerospace Engineering
Queen's University Belfast
Ashby Building, BT9 5AH, UK.
d.adair@qub.ac.uk