International Review of Economics Education 49 (2025) 100310

Contents lists available at ScienceDirect

International Review of Economics Education

journal homepage: www.elsevier.com/locate/iree

Integrating programming into the modern undergraduate economics curriculum☆

Nigar Hashimzade [a], Oleg Kirsanov [b], Tatiana Kirsanova [b,*]

[a] Department of Economics and Finance, Brunel University of London, United Kingdom
[b] Economics, Adam Smith Business School, Gilbert Scott Building, University of Glasgow, Glasgow G12 8QQ, UK

A B S T R A C T

The increasing demand for computational skills in economics necessitates the integration of programming into undergraduate economics curricula in the UK. This paper argues for a systematic incorporation of programming courses tailored to economics students, addressing the limitations of current approaches and highlighting the benefits of such integration. We propose a sequence of introductory and intermediate-level integrated courses, and argue that this curriculum change will enhance students' understanding of economic concepts, improve their employment prospects, and better prepare them for postgraduate studies. This paper aims to initiate a discussion and exchange of ideas and experiences on this subject at the national level.

1. Introduction

The rising demand for programming skills in economics has highlighted the necessity of integrating programming into undergraduate economics curricula in the UK. Despite the increasing relevance of these skills in both the job market for economics graduates and postgraduate education, a notable gap persists between the programming competencies expected at the graduate level and those typically acquired during undergraduate studies. This gap, acknowledged by economics educators (Solis-Garcia, 2021; Neumuller et al., 2018), is particularly evident in the inadequate technical training for solving complex optimization problems and dynamic models that require numerical methods and programming expertise. Unfortunately, such skills are generally not developed within the standard undergraduate economics curriculum, to the best of our knowledge.

We want to use this paper as an opportunity to initiate a discussion and exchange of ideas and experiences at the national level about integration of programming in the undergraduate economics curriculum in the UK. We propose a sequence of two programming courses (or modules) and describe how they can be integrated into otherwise standard curriculum, with macroeconomic and microeconomic modules at the introductory and intermediate levels, followed by optional courses at the advanced level. We argue that it is time to introduce mandatory computer programming training in economics at the undergraduate level, in the same way mathematics and data analysis became integrated into the curriculum over thirty years ago. In this paper, we discuss this proposal in detail, outline the trade-offs and practical considerations involved, describe potential teaching and assessment methods, and give an illustration for a three-year (England) and a four-year (Scotland) undergraduate economics courses.

This paper is structured as follows. Section 2 describes the evidence of the demand for programming skills in the modern job market

☆ The views expressed are those of the authors and do not necessarily reflect those of Brunel University of London and the University of Glasgow. We thank the Editor and two anonymous referees for helpful comments and suggestions.

* Corresponding author.
E-mail addresses: nigar.hashimzade@brunel.ac.uk (N. Hashimzade), oleg.kirsanov@glasgow.ac.uk (O. Kirsanov), tatiana.kirsanova@glasgow.ac.uk (T. Kirsanova).

https://doi.org/10.1016/j.iree.2025.100310

Available online 11 March 2025

for economics graduates, outlines the current limitations in economics education, and presents the insights from the relevant literature on education in other subject areas. Section 3 outlines our proposed model of integration of programming courses into the undergraduate economics program and explores the trade-offs and practical considerations in implementing these changes. The paper concludes with a summary of the key arguments and implications.

2. The case for programming in undergraduate economics education

2.1. Programming skills and the modern job market

Employers increasingly seek economists – including those holding undergraduate degrees – who are adept in computational methods. A diverse range of organisations expect their entry-level hires with undergraduate degree in economics to possess programming skills for applications in both macroeconomic and microeconomic contexts. In the 2019 survey of employers of economics graduates,[1] conducted by the Economics Network, a prominent academic organization dedicated to enhancing the teaching and learning of economics at universities, programming was mentioned in the answers to the question "Which skills (general and specific) and knowledge do you believe most need to be developed further in economics degree courses?" Proficiency in programming tailored for economic applications will significantly improve employability of economics graduates. This is reflected in the contemporary career advice provided to undergraduate economics students, where the need for computational skills is increasingly emphasised. According to a blog post (University of Sussex, 2024), economics graduates who possess advanced programming skills are in high demand across various industries, including technology, finance, healthcare, and e-commerce. These skills enable them to apply statistical techniques, leverage machine learning, and solve complex analytical problems. A blog post from LSE (Carrigan and Brooker, 2019) highlights the increasing demand for computer programming skills in the social sciences. Furthermore, there is evidence that the wage premium for liberal arts graduates with computer programming skills exceeds that of graduates equipped solely with data analysis skills – a traditional attribute of an economics graduate (Blumenstyk, 2016).

Central banks, such as the Bank of England, the European Central Bank, and the Bank of Japan, recruit economics graduates for roles involving economic forecasting and policy analysis using Dynamic Stochastic General Equilibrium (DSGE) models. International organisations like the International Monetary Fund, the World Bank, and the Organisation for Economic Co-operation and Development employ undergraduate economics graduates to assist in macroeconomic analysis and policy evaluation, where programming skills are essential. Private financial institutions – including Goldman Sachs, JP Morgan, and BlackRock – hire economics graduates to work on integration of macroeconomic forecasts into financial strategies, necessitating strong programming abilities.

Moreover, technology firms such as Google, Amazon, Meta, and Microsoft recruit economics graduates for positions involving market design, pricing algorithms, auctions, and platform economics – areas that demand proficiency in programming. E-commerce platforms like Amazon and Alibaba employ economics graduates to optimise pricing strategies, analyse consumer behaviour, and forecast demand through large-scale data analysis and machine learning. Competition authorities, including the Competition and Markets Authority in the UK and the Directorate-General for Competition of the European Commission, hire economics graduates to work on market dynamics, game-theoretic models of competition, and regulatory frameworks, where programming is instrumental for simulations, cost-benefit analyses, and empirical studies. Similarly, healthcare firms and pharmaceutical companies employ economics graduates to use microeconomic analysis for pricing models, policy impact assessments, and demand forecasting in response to changing market conditions or regulatory environments, often involving programming for analysing multimodal data from clinical trials or healthcare markets. These industry demands highlight the pressing need to re-evaluate and enhance the current economics curriculum.

The following two examples illustrate the real-world applications of programming in economics: NHS England's use of Discrete Event Simulation and the forecasts produced by the UK Office for Budget Responsibility during COVID-19.

NHS England utilised a Discrete Event Simulation (DES) model to explore the impact of changing workforce allocations on surgery waiting times (Harper et al., 2023). Developed to support capacity planning for elective orthopaedic surgeries, this model simulated various scenarios, such as adjusting the number of beds and operating theatres and improving productivity by reducing lengths of stay and delayed discharges. Using anonymised patient records from 2016 to 2019, the DES model demonstrated that increasing the number of beds and reducing lengths of stay could significantly lower bed utilisation and allow for additional theatre activity. This approach enabled NHS planners to identify the optimal balance of resources needed to meet new targets and reduce surgical backlogs.

Without programming, constructing such complex, dynamic models to simulate different allocation strategies would be nearly impossible. Simulation allows economists to test and refine policies before implementation, visualise the impacts of allocation decisions on key outcomes like waiting times and patient access, and handle complex data involving patient flows, hospital capacity, and workforce constraints.

During the COVID-19 pandemic in 2020, the UK Office for Budget Responsibility (OBR) had to produce forecasts amidst significant economic uncertainty.[2] Economists working at the OBR were tasked with updating models to reflect the pandemic's impact, including forecasting the effects of government interventions like furlough schemes, public health expenditure, and changes in taxation. These forecasts were essential for the UK government in planning its response to the crisis. Using DSGE models and other simulation tools, the OBR was able to test different fiscal policies to understand their impact on debt, unemployment, and economic recovery.

[1] https://www.economicsnetwork.ac.uk/projects/surveys/employers2019
[2] https://obr.uk/coronavirus-analysis/

Programming skills are crucial in this work, involving data analysis of large datasets, forecasting and simulating macroeconomic models to predict future conditions, and visualising the results for effective communication of the outcomes in various scenarios to the government and to the public.

These examples, while drawn from complex, real-world economic challenges, remain directly relevant to the skillset and career paths of economists with undergraduate degrees. Teams at both the NHS and the OBR are typically composed of economists and analysts with varying levels of academic and professional experience. The presence of individuals holding a bachelor's degree in economics, who often contribute to data collection, preliminary analysis, and the application of programming tools to routine model building and calibration, attests to the practical nature of these tasks. Even though the conceptual frameworks, such as discrete event simulations or the fiscal forecasts developed under high uncertainty, may appear advanced, the foundational coding skills and analytical techniques we advocate can be meaningfully employed at the entry level. As these organizations routinely hire economics graduates with bachelor's degree, the ability to understand, adapt, and implement basic computational solutions and interpret their results is not only beneficial but increasingly essential for their work. In this sense, the examples are not merely aspirational; they illustrate the kind of work environments and tasks where programming-enhanced economic training is directly applicable for graduates entering the workforce.

2.2. *Current approaches and their limitations*

Real-life economic applications rely increasingly on computer programmes in solving complex mathematical models, analysing large multimodal datasets, and simulating the projected outcomes of changes in economic environment. Yet, in the UK's undergraduate economics curriculum, programming skills are rarely taught in a systematic way. Students typically conduct data analysis using software packages like R, Stata, or EViews, which offer menu-driven interfaces that do not foster genuine coding proficiency. Although this approach suffices for basic econometric assignments, it does not equip students to implement or simulate more advanced structural economic models. As a result, economics graduates often lack the technical foundations needed to engage with modern research and professional work in economics.

Without a dedicated computing component, the teaching of core economics subjects, especially those that rely on computationally intensive models, can be rendered superficial. For example, in microeconomics courses students learn about optimisation and strategic interaction, but without numerical computational techniques the analysis is necessarily restricted to a set of 'toy models'. Similarly, in macroeconomics, students encounter models aiming to explain growth, business cycles, or policy impacts, but these models are, again, simplified to allow for pencil-and-paper solutions. While simple models are useful for introducing fundamental concepts and initial exploration, students may perceive such models as overly abstract and disconnected from the real world. The absence of robust coding practice leads to a gap between theoretical coursework and the empirical, data-driven realities of economic analysis.

In response to these challenges, some UK undergraduate programmes have begun offering stand-alone computational economics courses at the advanced (honours) level. Institutions such as the London School of Economics, University College London, Univesities of Southampton, St Andrews, and Glasgow offer methods-based computational economics courses. However, evidence from instructors suggests that while these courses are valuable, they typically arrive too late in the curriculum to support earlier coursework, and they devote significant time to teaching basic programming skills that students could have acquired beforehand.[3]

One example of such a course is provided by Jenkins (2022), who describes a 20-hour module taught at a US university but comparable in scope and level to those found in the UK. Jenkins argues that computational methods are essential for solving and simulating modern economic models, many of which do not admit paper-and-pencil closed-form solutions. His course allocates six hours – nearly the one third of the total time – to teaching the basics of a programming language, a decision he acknowledges as a difficult trade-off between time spent on foundational programming skills and time available to engage with economic applications. We share this concern: dedicating one third of available time to learning a programming language limits the instructor's ability to cover a broad spectrum of topics, resulting in a rushed experience that may only skim the surface of economic applications.

Moreover, while Jenkins suggests that a 'programming is like cooking' approach, where students learn by doing and focus on immediate applications, is sufficient to expose students to the necessary programming skills, we argue that this approach, while practical, is insufficient for achieving deep and transferable learning outcomes. A short, methods-based course may enable students to code simple models, but it does not provide the comprehensive programming foundation necessary for more advanced, independent economic modelling and analysis. In addition, this approach risks omitting crucial elements of programming culture, such as code efficiency, debugging strategies, and best practices in software development, which are essential for applying computational methods across a wider range of economic models.

With limited instructional hours, it is also challenging for students to gain a solid grasp of both the computational techniques and the economic theories they are intended to illuminate. This problem is exacerbated when the course must also introduce entirely new economic topics, leaving insufficient time to explore the full potential of computational methods. The risk is that students will leave the course with a fragmented understanding of how to apply these methods in other contexts, particularly in areas not explicitly covered in the curriculum.

An alternative might be for economics students to take programming courses offered by the Computer Science (CS) department. However, this option is not practical within the UK educational system for several reasons. CS courses are designed with a focus on

[3] Two Glasgow-based authors who have taught such a stand-alone course share this view.

software development, computational theory, and programming languages that are often not directly applicable to economics. They may cover languages like Java or C+ +, which are less commonly used in economic computations compared to MATLAB, Python, or R. Furthermore, CS courses may emphasise topics like data structures, algorithms, and computer architecture, which, while valuable, do not align with the immediate needs of economics students.

Economics students require knowledge of numerical methods specific to their field, such as root-finding algorithms, numerical integration, optimisation techniques, and simulation of dynamic systems. CS courses typically do not cover these methods within the context of economic applications. As a result, students may not acquire the specific computational tools necessary for their future work or postgraduate studies. Furthermore, the apparent lack of relevance to economic applications may even demotivate students from putting effort into learning.

In addition, the UK educational system is characterised by highly structured degree programmes with limited flexibility for electives outside the prescribed curriculum. Economics students may find it impossible to enrol in CS courses due to prerequisites they have not completed, scheduling conflicts with mandatory economics courses, and capacity constraints within CS departments that prioritise their own students.

Taken together, these approaches of either relying on stand-alone advanced courses or referring students to CS courses do not effectively integrate programming into the core of the undergraduate economics curriculum. Lacking a solid coding foundation, the students cannot fully exploit computational techniques in their microeconomic and macroeconomic studies and may end up ill-prepared for contemporary research, data-intensive policy roles, or postgraduate training. This underscores the need for earlier and more systematic integration of programming into the economics curriculum.

2.3. Related literature

Economics education research has long recognized the need for more systematic programming skills in economics (Adams and Kroch, 1989; Day, 1987; Motahar, 1994). Some authors argue that specialised advanced-level courses like 'Computational Macroeconomics' (Jenkins, 2022) or, for example, macroeconomics courses incorporating several lectures on programming to assist the exposition of DSGE models (Solis-Garcia, 2021) could improve student understanding and reduce the skills gap. However, we are not aware of any existing proposals of integrating programming in the undergraduate economics course, or, indeed, into other quantitative social science or humanities curricula, even though the importance of adapting programming to each field's needs has been acknowledged elsewhere. For instance, Charlton and Birkett (1992) document an attempt to teach elementary programming to accountancy students and concludes that, to be successful, a programming course must be relevant for both the vocational needs and for the main subject. Similarly, Jeffcote (1997) uses a case study of an interdisciplinary *Information Technology and Society* undergraduate programme to argue that IT competence, as a transferable skill, improves employability prospects of students whose first choice was a social science or humanity subject.

There is, on the other hand, a literature on integration of programming in the undergraduate teaching of (non-computer) engineering. The need in systematic teaching of programming skills to engineering students emerged decades ago, and integration of programming in the undergraduate engineering curricula has become widely spread, although is still far from universally accepted. An example of a set of curricula for electrical, mechanical, and manufacturing engineering courses with integrated programming is given in Dunne et al. (2005). The authors outline similarities and differences in programming components across engineering disciplines and emphasise the need to adapt the choice of teaching and assessment to the programming skills demanded in specific industries.

Integration of programming in the undergraduate Electrical (and Electronic) Engineering curriculum was advocated in Baldwin et al. (1979), notably, by an ad hoc group of managers from five larger employers of electrical engineers at that time.[4] Among more recent work on pedagogical approaches to integrated programming in this discipline, Pejcinovic and Wong (2017) describe successful use of hands-on projects combining problem-solving, programming, and hardware interfacing in the redesigned curriculum, and its advantages over the earlier structure which included topics unrelated to electrical engineering. Integrated approach and its benefits for students' motivation and engagement, along with the students' feedback, are described in Notaros et al. (2019) and Grindei et al. (2023).

In a similar vein, dos Santos et al. (2018) note that in many Chemical Engineering programmes programming is taught at the beginning, with examples often not related to the main subject. They argue for introducing an elective programming course for the final year students, with applications specific to chemical engineering, and share their experience and learning outcomes based on students' survey. Arjmandi et al. (2023) note the challenges of integrating programming in the undergraduate engineering curriculum and describe their experience of pedagogical approach to introducing weekly programming workshops complementing the lectures in one of the core second-year courses in an undergraduate Chemical and Material Engineering programme.

In the undergraduate Civil Engineering teaching, Bowen (2004) argues for the use of programming in first-year structural design group project, in order to motivate and retain students, as the U.S. system allow them to switch engineering major. da Silveira Monteiro and da Silva Pitangueira (2018) describe how programming and numerical methods can be integrated in undergraduate Civil Engineering course using a research-led teaching platform shared with other engineering courses.

While the principles are similar across different engineering subjects and sciences, there are variations in pedagogical approaches. In the undergraduate Physics education, for example, Gould and Tobochnik (2001) is among early proposals for integration of

[4] Bell Laboratories, General Electric Company, IBM, Magnavox, and Westinghouse Electric Corporation.

programming. Chonacky and Winch (2008) and, more recently, Caballero and Merner (2018), give an overview of the curriculum reform, with examples of practices in structure and pedagogical approaches, as well as perceived barriers and difficulties. In a case study of integrated programming for Chemistry and Biochemistry, Biochemistry, Fuchs et al. (2024) outline the challenges faced by the students and provide recommendations on pedagogical approach for improvement of learning outcomes.

In this paper, we contribute to the education literature by proposing a systematic approach to developing programming skills among undergraduate economics students, drawing on lessons from science and engineering courses with integrated programming. Based on the UK educational system, with perspectives from authors in Scotland and England, we advocate for designing and integrating a sequence of programming courses into the curriculum similarly to data analysis courses. This involves starting with an introductory programming course, followed by an intermediate-level course, as described in the rest of the paper.

3. Proposed integrated programming curriculum

Integrating programming into the curriculum offers several advantages. First, it allows for the early development of computational skills, enabling students to engage more deeply with economic models throughout their studies. Second, programming education encourages active learning techniques such as hands-on coding exercises, pair programming, and project-based learning. These methods enhance problem-solving skills and promote a deeper understanding of economic concepts. Last, but not least, economics graduates with programming proficiency are more competitive in the job market, as employers increasingly value candidates who can apply computational methods to economic analysis.

We propose a systematic integration of programming courses into the undergraduate economics curriculum through a two-course sequence: *Programming Principles for Economists* at the introductory level, focusing on basic programming concepts, programming logic, and an introduction to numerical methods relevant to economics, followed by *Applied Programming for Economists*, an intermediate-level course that applies programming skills to complex economic models and real-world economic problems. This section focusses on describing the content and structure of these proposed courses, demonstrating how they are designed to achieve the outlined advantages. Additionally, we examine the broader curricular changes and trade-offs involved in implementing this two-course sequence toward the end of the section.

3.1. Programming principles for economists

The introductory programming course is designed to provide economics students with essential programming skills tailored to their field. It should be mandatory and taken during the first year, or one of the pre-honours years, similar to how mathematics and introductory statistics are integrated into the curriculum. The course will consist of three key components: Programming Logic and Computer Operations, Language-Specific Basics of Programming, and Introduction to Numerical Algorithms and Their Implementation. The integration of these methods with economic theory ensures that students gain relevant, practical skills that they can apply directly to their studies and future careers in economics.

3.1.1. Course design

3.1.1.1. Programming logic and computer operations. The first component focuses on programming logic and a fundamental understanding of computer operations. This section aims to develop students' understanding of the importance of structured code and the basic principles of how computers execute instructions. In many stand-alone computational economics courses, this part is often neglected due to time constraints. However, familiarity with programming logic makes subsequent learning more efficient and reduces the number of errors students are likely to make – an essential consideration since economics students are often occasional programmers who are more prone to making mistakes in the code.

For instance, understanding how the system processes instructions and manages memory can significantly ease the debugging process, turning what is often a frustrating experience into a more manageable and even enjoyable task. This skill is crucial, as effective debugging is uncommon even among doctoral students who are exposed to programming primarily through their research rather than formal training. By including a strong emphasis on programming logic, students will be better equipped to use programming as a tool to enhance their understanding of economic models.

This part of the course may not require the use of a specific programming language and can instead rely on pseudocode and flowchart symbols to teach the principles. This approach allows students to grasp the core concepts without being overwhelmed by syntax or language-specific details. It is recommended that this component take up to six hours of the total course time but that these hours be distributed throughout the course to reinforce relevant concepts as they arise. Teaching can be based on chapters of Farrell (2012), though instructors may need to adapt certain chapters to better fit the course's specific focus.

3.1.1.2. Language-specific basics of programming. The second component covers the basics of a specific programming language, where the choice of the language and, subsequently, the choice of the textbook and other materials, will depend on instructor's preferences. Topics can include variables and data types, mathematical operations, functions and scripts, control flow commands, and basic input/output operations. This part of the course is conventional and aligns with the content typically covered in computational economics courses. Based on our own experiences and the structure of similar courses, six hours is a reasonable amount of time to dedicate to this section, allowing for the introduction of some small economics-related examples to illustrate the concepts in a practical context.

3.1.1.3. Introduction to numerical algorithms and their implementation. The third component introduces students to key numerical algorithms and their implementation in the chosen programming language. For example, a careful discussion of root-finding methods will enable students to understand the differences between algorithms such as Newton's method and the bisection method, and learn when and how to simplify multivariate problems, visualise function behaviour, assess the existence or multiplicity of roots, and choose the appropriate root-finding method. Additional topics could include spectral decomposition of matrices, numerical differentiation and integration, polynomial fitting, and the use of symbolic operators.

In stand-alone computational economics courses, the focus on specific examples often leaves little time to discuss the broader applicability of chosen numerical algorithms, such as the importance of solver options and their impact not only on the speed and accuracy of results but also on the ability to obtain a solution at all. This section is vital because it teaches students how to make algorithmic decisions based on the nature of the problem they are solving, a skill that is often underdeveloped in short, methods-focused courses. By integrating this content into an introductory course and allocating six to eight hours to this section, students will have sufficient time to engage with practical examples drawn from economics, leading to a more robust understanding of how to apply these techniques across a wide range of economic problems. A possible textbook for this section is Lindfield and Penny (2018), although the instructor may need to select specific algorithms most relevant to the course's focus.

3.1.2. Course structure and assessment

The course should include regular homework assignments that can be automatically graded using tools,[5] allowing for timely feedback in large classes. Computer lab sessions should be organised as hands-on exercises where students work on new problems based on the material covered in previous lectures. Instructors may choose to have students submit their work for assessment at the end of each session, similar to the approach discussed by Jenkins (2022), depending on the feasibility and resources available.

Assessment for the introductory course can reflect the potentially large class sizes and the need for a scalable, objective evaluation of basic programming proficiency. For a large cohort, an in-course (mid-term) test, possibly structured as an automatically graded quiz or a series of coding tasks, can assess students' ability to implement fundamental programming concepts and apply them to simple economic problems. While project-based assessment is often beneficial in programming courses, at the introductory stage it may be more practical to employ formats similar to those used in undergraduate computer science programmes, where basic coding exercises and algorithmic questions can be efficiently auto-graded. This approach allows for consistent feedback and reliable measurement of core skills without requiring an open-ended, research-style project.

A more substantial written exam at the end of the course could then include short essay-type questions on programming logic, good coding practices, and their relevance to specific economic applications. If not all students plan to continue to more advanced economics modules, this structure ensures that all learners acquire the essential computational foundations needed for further study or immediate entry into the job market.

This is, of course, just one possible option for assessment. An instructor can use different assessment forms and structures, adapted accordingly to the availability of resources, the needs of the department, and the students profile.[6]

3.2. Applied programming for economists

Building on the introductory course, the intermediate-level programming course aims to deepen students' programming skills and apply them to complex economic models. This intermediate-level course should be compulsory for students enrolled in the single-honours Economics degree, whereas for students in related programmes (e.g., Business Economics) or joint degrees (e.g., Economics and Management), the course could be offered as an elective. For the latter, this approach will allow flexibility in choosing other electives, while still encouraging the development of valuable computational competencies.

3.2.1. Course design

While this course may share some similarities with stand-alone computational economics courses, there are important differences, particularly in the choice and depth of topics. Our approach is informed by the Glasgow-based authors' experience with the 'Economic Analysis with MATLAB' course, taught at the University of Glasgow.[7]

The course builds on the introductory programming course, focusing on a range of economic applications that are both engaging and relevant to students interested in various areas of macroeconomics and microeconomics. Since the basics of the programming language should already be covered in the introductory course, this intermediate course can begin immediately with applying these skills to real economic problems.

Given the structure of the UK educational calendar, instructors will have about 20 h to cover approximately five topics in depth. Each topic should be largely self-contained, relying on material from introductory courses and general economic intuition, so that

[5] Examples are MATLAB Grader and nbgrader for Python.

[6] One consideration in choosing the form of assessment is the use of AI and issues of academic integrity. As pointed out by Liu (2023), current capabilities of ChatGPT and similar tools are limited in debugging and data-driven numerical calculations. To that extent, typical applications in the proposed courses are less vulnerable to the abuse. At the same time, an efficient appropriate use of AI can be demonstrated in class, as the means of expediting simple coding tasks.

[7] All examples in this section were used by the Glasgow-based authors in teaching 'Economic Analysis with MATLAB', ECON4101, at the University of Glasgow, between 2021 and 2024, unless stated otherwise.

students can follow along even if they haven't covered the specific economic theory elsewhere. Importantly, each topic should have a strong computational element, where programming is essential for understanding and visualising the results.

3.2.2. Proposed topics

We propose the following topics based on some of the existing courses and other ideas.

3.2.2.1. Solving and simulating linear dynamic systems. A Repeat Customer Model (based on an example presented in Karst, 2013) works well in this topic. This model helps students understand how eigenvalues and eigenvectors explain the dynamics of a non-stationary dynamic system. Homework might include modifying the model to incorporate a loyalty scheme, adding complexity to the dynamics. Alternatively, a dynamic IS-LM model with Adaptive Expectations could be used, potentially linking with standard macroeconomics courses.

3.2.2.2. Solving and Simulating Non-Linear Dynamic Systems. The SIR Model in Epidemiology (a discrete time version of, e.g., Hethcote, 2000) is particularly relevant given the interest in the economic and public health effects of the COVID-19 pandemic. This model serves well as an example of a non-linear system, as we gradually introduce features like vaccination and quarantine, illustrating the complex dynamics of such models. We discuss potential extensions to assess the welfare cost of epidemics. Other potential examples – which can appear either in the course or in homework – include environmental economics, evolutionary games, and competition models, all leading to the Lotka-Volterra equations. These systems often have non-trivial steady states, allowing to apply root-finding techniques discussed in the Programming Principles course.

3.2.2.3. Markov chains. Markov chains are useful in many macroeconomic models, though they are rarely covered in standard courses. We focus on transition matrices and absorbing states, with examples like demographic models and labour market models (an excellent exposition and many examples that can be used in homework can be found in Bradley and Meeks, 1986). This topic also ties back to the importance of eigenvalues in understanding economic dynamics. Additionally, the demographic example discussed in Bradley and Meeks (1986) works well to build data visualisation skills.

3.2.2.4. Portfolio choice. This topic introduces the basics of risk and return, guiding students through the principles of constructing an efficient portfolio (an accessible and self-contained treatment can be found in Bodie et al., 2024). It is an appealing introduction to financial theory for economics students, providing a practical application of computational techniques and elementary probability theory. An assignment can discuss the differences between different types of portfolios.

3.2.2.5. Social networks. Social networks offer intuitive concepts and real-life examples that students find engaging. They also provide a rich ground to practise programming skills. For example, when computing betweenness centrality, we work with graphs. Instead of using a 'black box' of a standard software package, such as the NodeXL plug-in in Excel, students can develop logical algorithms to compute the number of paths with certain characteristics and code this efficiently.

Social network applications are diverse and always engaging. We have studied Florentine marriages (Padgett and Ansell, 1993), the power of judges (an example from Bradley and Meeks, 1986), and internet page indexing (Jackson, 2008), highlighting the importance of eigenvalues in understanding economic outcomes. A natural extension is to study simple examples of games on a network. We used homework examples based on network representations of criminal gangs found in the literature.

3.2.2.6. Rational expectations in macroeconomics. While not currently included in our course, this topic could easily be added. It involves explaining and coding the Blanchard-Kahn theorem (Blanchard and Kahn, 1980), applying it to simple monetary policy models.

3.2.2.7. Real business cycle model. Once students understand Rational Expectations, they can explore the Real Business Cycle model, possibly using symbolic packages for linearisation. This would allow students to simulate and analyse this model within the course's timeframe (see Solis-Garcia, 2021, for a similar approach).

An instructor may choose to cover fewer topics but study them in greater depth. However, our own experience suggests that students value the diversity of topics, the exposure to different fields, and the clear understanding of how skills developed through a particular model are transferable to a much wider range of models.

3.2.3. Course structure and assessment

The course will include regular homework assignments. Unlike the introductory course, these assignments might be too complex for automated grading due to the variety of possible solutions. Instead, we can use online learning platforms for submissions, with deadlines set early enough to allow common mistakes to be discussed in weekly computer lab sessions. These labs should be hands-on, with students working on new problems based on lecture material.

Given the likely large class sizes, assessments need to remain both rigorous and varied, allowing students to demonstrate their strengths in multiple ways. A mid-term assessment (in-course exam), potentially using multiple-choice questions, can evaluate the understanding of key concepts and programming skills at a more advanced level than the introductory course. For the final assessment, we propose a substantial group project. Alternative methods, such as individual take-home exams or multiple smaller assignments, can also be used. Issues to consider are the potential risk of overloading students during a busy teaching period and the need to address the

depth and realistic complexity of the tasks appropriate for advanced skills.

A final, project-based assessment carried out in the post-teaching period affords students the time and flexibility to engage meaningfully with a complex economic problem. Project-based and team-based learning at this stage aligns well with best practices observed in STEM disciplines, where such approaches help students develop not only technical expertise but also transferable skills like teamwork, communication, and project management. The project could involve, for example, welfare analysis of strategic trade, calibration and simulation of an endogenous growth model, effect of R& D subsidies on firms, or a comparison of different modeling approaches for the same economic application (such as SIR models, Markov chains, and social networks in the cost-benefit analysis of public health policies during an epidemic). This method ensures a more authentic learning experience, better mirrors the collaborative nature of professional economic research, and serves as a natural complement to earlier assessments that focus on core theoretical and computational competencies.

3.3. Teaching methods for programming courses

To maximise the effectiveness of both the introductory and intermediate-level programming courses, we employ teaching methods grounded in constructivist learning theory, situated learning theory, and cognitive flexibility theory. These pedagogical approaches are designed to enhance learning outcomes by fostering active engagement, contextual understanding, and adaptable problem-solving skills.

Our teaching methods emphasise active learning through hands-on exercises, consistent with constructivist learning theory, which posits that learners construct knowledge through experiences and reflections (Piaget, 1996). In both courses, students engage in coding exercises during lab sessions, directly applying programming concepts to economic problems. This experiential approach allows students to build their own knowledge structures by actively exploring and solving problems, leading to a deeper understanding of both programming and economics.

We also incorporate contextual learning, drawing from situated learning theory, which suggests that knowledge is best acquired in context (Lave and Wenger, 1991). By integrating programming tasks with real-world economic scenarios, we help students see the relevance of programming to their field, enhancing motivation and retention. For instance, in the introductory course, students might use programming to solve basic economic equations or visualise data, while in the intermediate course, they apply programming skills to more complex models such as simulating the dynamics of an epidemic or optimising investment portfolios.

To develop cognitive flexibility, we encourage students to approach problems from multiple perspectives and adapt their strategies accordingly, aligning with cognitive flexibility theory (Spiro et al., 1991). By exposing students to a range of numerical methods and encouraging them to make algorithmic decisions based on the characteristics of the problems they are solving, we foster adaptability and advanced problem-solving skills. This prepares them to navigate complex economic models and select appropriate computational techniques as needed.

Collaborative learning and peer instruction are integral to our teaching methods, inspired by Vygotsky (1978) social constructivism, which emphasises the social nature of learning. We utilise strategies such as pair programming and group projects, allowing students to articulate their understanding, learn from different perspectives, and develop communication skills. In both courses, students work together on assignments and projects, reinforcing concepts and correcting misconceptions through peer interaction.

Reflective practice is also incorporated into the courses, encouraging students to consider how their programming choices impact the outcomes of economic models. Through discussion sessions and group processing after completing the projects, students internalise what they have learned and understand its broader implications.[8]

By employing these teaching methods across both courses, we anticipate several enhanced learning outcomes. Students are expected to develop a deeper conceptual understanding of programming and economics, increased engagement and motivation, improved problem-solving abilities, and better preparedness for advanced studies and careers requiring computational economics expertise.

Integrating these pedagogical strategies not only aligns with established learning theories but also creates a cohesive learning environment that supports the progression from foundational knowledge to complex application. By fostering active engagement and contextual understanding, we aim to equip students with the skills necessary to navigate the increasingly rich computational landscape of modern economics.

3.4. Integration with the economics curriculum

Integrating the proposed programming courses into the existing economics curriculum requires careful coordination with other foundational subjects. The introductory Programming Principles for Economists (PPE) course should ideally run in parallel with standard calculus and introductory econometrics modules. Because the concept of a root is introduced early in mathematics instruction, students can readily transfer this understanding to computational root-finding techniques covered in PPE. Similarly, because eigenvalues and eigenvectors typically appear later in the mathematics sequence, so aligning their introduction with PPE lectures ensures that programming builds naturally on these mathematical tools.

The subsequent Applied Programming for Economists (APE) course, offered at the honours level, can then be coordinated with core

[8] See Johnson and Johnson (1999) on educational benefits of group processing.

macroeconomics and microeconomics modules. Instead of duplicating material, APE focuses on computational methods that deepen students' understanding of theoretical models taught elsewhere. As discussed in Section 3.4, integrating the timing of APE with key topics in macro and micro courses allows students to apply their coding skills precisely when they encounter challenging models, making those models more accessible and meaningful.

For instance, macroeconomic models in widely used textbooks such as like Carlin and Soskice (2015) often differentiate short-run, medium-run, and long-run dynamics, yet students can struggle to connect these theoretical distinctions with real-world economic processes. In an intermediate-level module, they might be asked to code the IS-LM-PC model and experiment with different policy scenarios, thereby clarifying how short-run and medium-run effects unfold. Later, in a more advanced module, they could apply similar computational techniques to the Solow growth model, observing how parameter changes shape long-run outcomes and gaining a deeper grasp of macroeconomic growth theory and policy. By studying such models in parallel with a programming course, students come to see them not as purely theoretical constructs but as tools they can use to explore real policy questions.[9]

Likewise, microeconomics offers numerous opportunities to benefit from programming. One especially relevant area involves game-theoretic models of strategic interaction, which are often used in microeconomics to analyse competition, auctions, or bargaining. For example, when studying Cournot equilibrium, students might be asked to wright a code that will produce characterisation of the equilibrium, visualise strategies, and identify the equilibrium point. By investigating how changes in the parameter values or in the modes of interaction, such as shifting from a Cournot duopoly to a Stackelberg duopoly, affects equilibrium quantities, prices, and welfare, students gain a deeper intuition for how timing and first-mover advantages shape market outcomes, with implications that extend to broader policy questions on market power and competitive dynamics. Moreover, once students can write basic code, instructors can introduce algorithms for identifying dominated strategies or computing mixed-strategy equilibria in $n \times 2$ games conceptually, then assign a coding exercise for practical exploration. Rather than guiding the class step-by-step through just one pen-and-paper example, the instructor can explain the algorithm conceptually and then assign a coding exercise. Students will have an opportunity to explore a range of scenarios by varying parameters, modifying payoffs, or experimenting with additional strategies, and thus gain a more thorough understanding of the underlying logic. This approach frees class time for higher-level discussions of theoretical insights or policy implications, making the application of game-theoretic concepts both more rigorous and more engaging.[10]

By incorporating these programming-based activities into both macro and micro modules, the curriculum ensures that computational skills are reinforced at each step, rather than remaining isolated within stand-alone courses. Students repeatedly see how coding deepens their understanding of key economic frameworks, making it clear that programming is not merely an extra skill but a tool integral to modern economic inquiry and policy analysis.

One outcome of this repeated exposure is a broader range of potential dissertation topics – an important development, given that dissertations are a central component in the vast majority of the undergraduate economics programmes in the UK. A typical dissertation requires formulating a research question, reviewing relevant literature, and conducting quantitative analysis. Students lacking programming skills often default to simple regression-based methods, leaving much of modern economics research – particularly involving dynamic or policy-oriented models – beyond their reach. By mastering a more powerful programming environment, undergraduates gain the ability to tackle a variety of contemporary issues, including macroeconomic policy analysis with DSGE models (Junior et al., 2022), life-cycle saving decisions (Findley, 2014), corporate strategy (Pezzino, 2016) and the links between inequality and growth (Hanlon, 2013). By linking core mathematical and economic concepts with computer programming, students can handle a broader set of research questions, producing richer and more original work in their undergraduate dissertations.

3.5. *Trade-offs and practical considerations*

Implementing new mandatory programming courses requires careful consideration of curriculum adjustments, resource allocation, and support for diverse student backgrounds.

Introducing new mandatory courses into an already packed undergraduate curriculum requires careful consideration of trade-offs. One practical challenge is determining which existing courses or content areas might be adjusted or reduced to make room for the introductory programming course. Department might consider streamlining the components of their existing portfolios, including the balance of mandatory and elective modules. An example of building in these two proposed courses, based on the 2024–25 curriculum for BSc Economics at Brunel University London (UCAS L100 3-year undergraduate programme) and MA(Hons) Economics at the University of Glasgow (UCAS L150 4-year undergraduate programme), is shown in the Appendix.[11]

An important consideration is the allocation of resources, including staffing and computational facilities. Providing adequate support for students learning programming – especially those who may have only little or no prior experience – requires investment in teaching assistants, computer labs, and access to software. Another challenge could be limited expertise in programming among

[9] Examples of additional macroeconomic topics where computational methods significantly improve learning include Strulik (2004), Dalton et al. (2012) and Bongers et al. (2020).

[10] Illustrations of coding-based microeconomic topics can be found in Kochanski (2012), Cobb and Sen (2014), Gorry and Gilbert (2015), and Kuroki (2021).

[11] In Scottish universities, an honours-level undergraduate degree awarded upon completion of a standard 4-year undergraduate programme in certain subjects, including economics, is traditionally called MA rather than BA or BSc. Thus, the MA(Hons) Economics programme at Glasgow is an undergraduate degree, comparable to a BA/BSc Economics elsewhere.

economics instructors. In many instances, however, academic economists have at least some relevant experience acquired as 'learning by doing' in their own research. Specifically tailored training for the instructors assigned to teach these courses can be provided if necessary using internal resources. The level of specialisation in teaching these courses will be similar to that for teaching mathematical methods.

It is also important to consider the differences in the backgrounds of students. Not all students may have the same level of comfort with technical subjects. Therefore, the programming courses should be designed with accesibility and inclusivity in mind, with support structures in place to assist students who may struggle initially. This could include supplementary tutoring sessions, online resources, and collaborative learning opportunities.

4. Conclusion

The increasing reliance on quantitative methods and computational techniques in economics has made programming proficiency indispensable for students aiming to effectively analyse complex economic phenomena and contribute to research and policy-making. Including programming in economics education will enable students to use numerical solution methods, sophisticated analysis of multimodal data, and scenario simulations. This hands-on approach allows them to solve and test complex economic models in active ways that traditional teaching methods that rely on 'toy models' cannot offer. A deeper, project-based involvement with coding will also help students improve their critical thinking and problem-solving abilities and to better their understanding of theoretical concepts and how they apply in real life.

Integrating programming into the undergraduate economics curriculum is not merely an enhancement but a crucial step in modernising economics education to meet the evolving demands of employers of economics graduates in the data-driven world. Employers across various sectors – including central banks, international organisations, financial institutions, technology firms, insurance and healthcare companies – increasingly seek candidates who can apply computational methods to economic analysis, forecasting, and data-driven decision-making. Equipped with these sought-after skills, the students enhance their competitiveness in the job market for the roles in the industry and public sector that require analytical rigour and technological adeptness.

Programming proficiency also expands students' research capabilities. With the computational tools to undertake sophisticated analyses, students can explore a wider range of topics in their undergraduate dissertations and other research projects. They will be better prepared to use advanced analytical methods, offer original ideas in economics, and explore today's economic problems that require advanced computer skills. This not only enriches their academic experience but also lays a strong foundation for those who pursue postgraduate studies or research-oriented careers.

By systematically incorporating tailored programming courses – starting with an introductory course that focuses on programming logic, computational languages, and basic numerical methods, followed by an intermediate-level course that applies these skills to real-world economic problems – we as educators will address the existing significant technical skills gap. This integration not only improves employability and bridges the divide between undergraduate and postgraduate studies but also enriches the overall educational experience for economics students.

Authors' contribution

Conceptualization: All three authors jointly formulated the research goals and aims.

Methodology: All three authors collaborated on the design of methodology, including models and pedagogical approaches.

Investigation: All three authors were involved in researching existing literature, collecting relevant data, and reviewing current practices in economics education.

Resources: All three authors identified and provided necessary materials, references, and computing resources for the study.

Writing – Original Draft: All three authors drafted the manuscript text, incorporating ideas and feedback iteratively.

Writing – Review and Editing: All three authors critically reviewed, commented on, and revised all sections of the manuscript at multiple stages.

Visualization: All three authors collaborated on figures, tables, and other illustrative materials to present the curriculum design and supporting arguments.

All authors contributed equally and approved the final version of the manuscript.

CRediT authorship contribution statement

Nigar Hashimzade: Conceptualization, Methodology, Investigation, Resources, Writing - Original Draft, Writing - Review and Editing, Visualization **Oleg Kirsanov:** Conceptualization, Methodology, Investigation, Resources, Writing - Original Draft, Writing - Review and Editing, Visualization **Tatiana Kirsanova:** Conceptualization, Methodology, Investigation, Resources, Writing - Original Draft, Writing - Review and Editing, Visualization

Declaration of Competing Interest

None.

Appendix A. Appendix

This Appendix shows two examples of building in these two proposed courses, based on the 2024–25 curriculum for BSc Economics at Brunel University London (UCAS L100-type 3-year undergraduate programme) and MA(Hons) Economics at the University of Glasgow (UCAS L150 4-year undergraduate programme).

We use PPE to denote Programming Principles for Economists and APE to denote Applied Programming for Economists course.

Table A1
An Example Based on BSc Economics Curriculum at Brunel University London.

Current	Credits	Proposed	Credits
Year 1: Level 4			
Mathematics for Economics and Finance	15	No changes	15
Microeconomic Principles	15	No changes	15
Macroeconomic Principles	15	No changes	15
Financial Markets	15	No changes	15
Introduction to Financial Accounting	30	Intro. to Fin. Acc.*	15
Statistics and Data analysis	30	No changes	30
		PPE	15
Year 2: Level 5			
Microeconomics Principles II	30	Micro. Principles II	15
Macroeconomics Principles II	15	No changes	15
Mathematical Economics	15	No changes	15
Money and Banking	15	No changes	15
Introduction to Econometrics	30	No changes	30
Monetary Economics	15	No changes	15
		APE	15
Year 3: Level 6			
Game Theory	15	No changes	15
Further Econometrics	15	No changes	15
Advanced Macroeconomics	15	No changes	15
Panel Data	15	No changes	15
2 optional modules	30	No changes	30
2 optional modules or Applied Research Project	30	No changes	30

* Note: Additional considerations may arise if the curriculum must comply with professional accreditation requirements (for example, from accounting professional bodies).

Table A2
An Example Based on MA(Hons) Economics at the University of Glasgow.

Current	Credits	Proposed	Credits
Year 1: Pre-Honours			
Economics Level 1 Courses	40	No changes	40
Other Level 1 Courses	80	No changes	80
Year 2: Pre-Honours			
Economics Level 2 Courses	40	No changes	40
Other Level 2 Courses	40	No changes	40
Intro Mathematics & Intro Statistics	20	No changes	20
Other Level 1 Courses	20	**PPE**	20
Years 3: Junior Honours			
Microeconomic Analysis	15	No changes	15
Macroeconomic Analysis	15	No changes	15
Econometrics 1	15	No changes	15
Econometrics 2	15	No changes	15
4 Optional Courses	60	3 Optional Courses	45
		APE	15
Year 4: Senior Honours			
Dissertation	30	No changes	30
6 optional courses	30	No changes	30

References

Adams, F.G., Kroch, E., 1989. The computer in the teaching of macroeconomics. J. Econ. Educ. 20 (3), 269–280.
Arjmandi, M., Woo, M.W., Mankelow, C., Loho, T., Shahbaz, K., Auckaili, A., Thambyah, A., 2023. Embedding computer programming into a chemical engineering course: the impact on experiential learning. Educ. Chem. Eng. 43, 50–57.

Baldwin, C.J., Cahn, C.R., Forman, J.W., Lehmann, H., Wischmeyer, C.R., 1979. A model undergraduate electrical engineering curriculum. IEEE Trans. Educ. 22 (2), 63–68.
Blanchard, O.J., Kahn, C.M., 1980. The solution of linear difference models under rational expectations. Econometruca 48 (5), 1305–1311.
Blumenstyk, G., 2016.Liberal-Arts Majors Have Plenty of Job Prospects, if They Have Some Specific Skills, Too. The Chronicle of Higher Education, June 9.⟨https://www.chronicle.com/article/liberal-arts-majors-have-plenty-of-job-prospects-iftheyhave-some-specific-skills-too/⟩.(accessed August 20, 2024).
Bodie, Z., Kane, A., Marcus, A.J., 2024. Essentials of Investments. McGraw Hill.
Bongers, A., Gomez, T., Torres, J.L., 2020. Teaching dynamic general equilibrium macroeconomics to undergraduates using a spreadsheet. Int. Rev. Econ. Educ. 35, 100197.
Bowen, J., 2004.Motivating civil engineering students to learn computer programming with a structural design project.In: 2004 Annual Conference, 9-931.
Bradley, I., Meeks, R.L., 1986. Matrices and Society. Princeton University Press.
Caballero, M.D., Merner, L., 2018. Prevalence and nature of computational instruction in undergraduate physics programs across the united states. Phys. Rev. Phys. Educ. Res. 14 (2), 020129.
Carlin, W., Soskice, D., 2015. Macroeconomics: Institutions, Instability, and the Financial System. Oxford University Press.
Carrigan, M., Brooker, P., 2019.Coding for and as social science.⟨https://blogs.lse.ac.uk/impactofsocialsciences/2019/07/12/coding-for-and-as-social-science⟩ (accessed September 14, 2024).
Charlton, J., Birkett, P., 1992. Should undergraduates studying arts and social sciences be taught computer programming? some evidence. Vocat. Asp. Educ. 44 (1), 107–120.
Chonacky, N., Winch, D., 2008. Integrating computation into the undergraduate curriculum: A vision and guidelines for future developments. Am. J. Phys. 76 (4), 327–333.
Cobb, B.R., Sen, T., 2014. Finding mixed strategy Nash equilibria with decision trees. Int. Rev. Econ. Educ. 15, 43–50.
daSilveiraMonteiro, H.A., daSilvaPitangueira, R.L., 2018. An overview of the numerical modeling and computer programming disciplines of an undergraduate civil engineering course and the insane project experience. Mecánica Computacional 36 (22), 1039–1048.
Dalton, T.R., Coats, R.M., Luccasen III, R.A., 2012. Exploring easter island economics with excel. Int. Rev. Econ. Educ. 11 (2), 102–114.
Day, E., 1987. A note on simulation models in the economics classroom. J. Econ. Educ. 18 (3), 351–356.
dosSantos, M.T., Vianna Jr, A.S., Le Roux, G.A., 2018. Programming skills in the industry 4.0: are chemical engineering students able to face new problems? Educ. Chem. Eng. 22, 69–76.
Dunne, B., Blauch, A., Sterian, A., 2005.The case for computer programming instruction for all engineering disciplines. In: 2005 ASEE Annual Conference, 10-1265.
Farrell, J., 2012. Just Enough Programming Logic and Design. Course Technology.
Findley, T.S., 2014. Using MS excel to solve and simulate the life-cycle/permanent-income model of consumption and saving. Int. Rev. Econ. Educ. 16, 129–146.
Fuchs, W., McDonald, A.R., Gautam, A., Kazerouni, A.M., 2024. Recommendations for improving end-user programming education: a case study with undergraduate chemistry students. J. Chem. Educ. 101 (8), 3085–3096.
Gorry, D., Gilbert, J., 2015. Numerical simulations of competition in quantities. Int. Rev. Econ. Educ. 18, 49–61.
Gould, H., Tobochnik, J., 2001. Integrating computation into the physics curriculum. International Conference on Computational Science. Springer, pp. 1031–1040.
Grindei, L., Constantinescu, C., Bojita, A., Holonec, R., Rapolti, L., 2023.Project-oriented approach in teaching programming to the first year undergraduate students in electrical engineering. In: 2023 10th International Conference on Modern Power Systems (MPS), 1-4.
Hanlon, M., 2013. Inequality and growth: understanding the link through a simulation. Int. Rev. Econ. Educ. 13, 44–49.
Harper, A., Monks, T., Wilson, R., Redaniel, M.T., Eyles, E., Jones, T., Penfold, C., Elliott, A., Keen, T., Pitt, M., Blom, A., Whitehouse, M.R., Judge, A., 2023. Development and application of simulation modelling for orthopaedic elective resource planning in England. BMJ Open 13 (12), 1–7.
Hethcote, H.W., 2000. The mathematics of infectious diseases. SIAM Rev. 42 (4), 599–653.
Jackson, M.O., 2008. Social and Economic Networks. Princeton University Press.
Jeffcote, R., 1997. Vocational computing skills and social science students–do they mix? some evidence from an interdisciplinary degree programme. J. Vocat. Educ. Train. 49 (2), 253–265.
Jenkins, B.C., 2022. A Python-based undergraduate course in computational macroeconomics. J. Econ. Educ. 53 (2), 126–140.
Johnson, D.W., Johnson, R.T., 1999. Learning Together and Alone: Cooperative. Competitive, and Individualistic Learning. Allyn and Bacon.
Junior, C.J.C., Garcia-Cintado, A.C., Junior, K.M., 2022. A modern approach to monetary and fiscal policy. Int. Rev. Econ. Educ. 39, 100232.
Karst, N., 2013. Applied Linear Algebra for Business, Economics and Finance. Mimeo, Babson College.
Kochanski, T., 2012. Toward teaching markets as complex systems: a web based simulation assignment implemented in Netlogo. Int. Rev. Econ. Educ. 11 (2), 102–114.
Kuroki, M., 2021. Using Python and Google Colab to teach undergraduate microeconomic theory. Int. Rev. Econ. Educ. 38, 100225.
Lave, J., Wenger, E., 1991. Situated learning: Legitimate peripheral participation. Cambridge University Press.
Lindfield, G., Penny, J., 2018. Numerical Methods: Using MATLAB. Academic Press.
Liu, Y., 2023.Leveraging the power of ai in undergraduate computer science In 2023 IEEE Frontiers in Education Conference (FIE), 1-5.
Motahar, E., 1994. Teaching modeling and simulation in economics: a pleasant surprise. J. Econ. Educ. 25 (4), 335–342.
Neumuller, S., Rothschild, C., Weerapana, A., 2018. Bridging the gap between undergraduate and graduate macroeconomics. J. Econ. Educ. 49 (3), 242–251.
Notaros, B.M., McCullough, R., Manić, S.B., Maciejewski, A.A., 2019. Computer-assisted learning of electromagnetics through matlab programming of electromagnetic fields in the creativity thread of an integrated approach to electrical engineering education. Comput. Appl. Eng. Educ. 27 (2), 271–287.
Padgett, J.F., Ansell, C., 1993. Robust action and the rise of medici, 1400-1434. Am. J. Sociol. 98 (6), 1259–1319.
Pejcinovic, B., Wong, P., 2017.Evolution of an introductory electrical engineering and programming course.In: 2017 ASEE Annual Conference & Exposition, Number, 10.18260/1-2–28312, Columbus, Ohio. ASEE Conferences.⟨https://peer.asee.org/28312⟩.
Pezzino, M., 2016. Understanding strategic competition using numerical simulations and dynamic diagrams in Mathematica. Int. Rev. Econ. Educ. 22, 34–47.
Piaget, J., 1996.Foundations of Constructivism.Longman Publishers USA.
Solis-Garcia, M., 2021. Yes we can! Teaching DSGE models to undergraduate students. J. Econ. Educ. 49 (3), 226–236.
Spiro, R.J., Feltovich, P.J., Feltovich, P.L., Jacobson, M.J., Coulson, R.L., 1991. Cognitive flexibility, constructivism, and hypertext: random access instruction for advanced knowledge acquisition in Ill-structured domains. Educ. Technol. 31 (5), 24–33.
Strulik, H., 2004. Solving rational expectations models using excel. J. Econ. Educ. 35 (3), 269–283.
University of Sussex, 2024.Career Options with Economics: Top 7 Careers in Economics.⟨https://isc.sussex.ac.uk/blog/career-in-economics⟩.(accessed September 14, 2024).
Vygotsky, L.S., 1978. Mind in Society: The Development of Higher Psychological Processes. Harvard University Press.