A Python-Based Undergraduate Course in Computational Macroeconomics

Brian C. Jenkins*

September 18, 2020

Abstract

I describe a new course that I taught at the University of California, Irvine in the winter quarters of 2019 and 2020. The course is a Python-based introduction to macroeconomic data analysis and modeling. Students develop basic familiarity with dynamic optimization and simulating linear dynamic models, basic stochastic processes, real business cycle models, and new Keynesian business cycle models. For many of my students this is their first experience with computer programming in any language. Students also gain familiarity with the popular Python libraries Numpy, Matplotlib, Pandas and made extensive use of the Jupyter Notebook. Feedback from students suggests that they found the course to be valuable, interesting, and enjoyable.

*Associate Teaching Professor, Department of Economics, University of California, Irvine, Phone: +1 (949) 824-0640, Fax: +1 (949) 824-2182, Mailing address: 3151 Social Science Plaza, Irvine, CA 92697-5100, Email: bcjenkin@uci.edu

1 Introduction

Computational methods are prominent in macroeconomic research. In fact, computational methods are often the only way to solve and simulate modern macroeconomic models because most simply don't admit pencil and paper solutions.[1] Computational methods are also often essential for economic data management, analysis, and visualization. Furthermore, many jobs in business and finance value computational proficiency at some level. Blumenstyk (2016) reports evidence that possessing computer programming skills raises the wages of newly graduated liberal arts majors by $14,000 per year. Data analysis and management skills produce a $12,000 annual wage premium.

In this article, I describe a new course that I taught at the University of California, Irvine (UCI) in the winter quarters of 2019 and 2020 under the name "Computational Macroeconomics". The course is a Python-based introduction to macroeconomic data analysis and modeling. Students practice downloading and managing macroeconomic data from internet sources, computing statistics, preparing data visualizations, simulating linear dynamic models, solving models of dynamic optimization, simulating real business cycle (RBC) and new Keynesian business cycle models, and verbally interpreting computed results. Students are also encouraged to think critically about the RBC modeling approach after reading and discussing papers by Prescott (1986) and Summers (1986). Along the way, students gain familiarity with the basics of computer programing and some popular Python libraries like Numpy, Matplotlib, Pandas, and the Jupyter Notebook environment. By the end of the course, students develop deeper macroeconomic intuition, gain experience analyzing and plotting data with Python, practice business cycle modeling, and build programming experience that will hopefully start them on a path of increasing computer proficiency.

The course is intended for senior economics majors and the prerequisites are intermediate macroeconomics and one quarter of econometrics. The course presumes no prior computer programming experience and in the first week, students get a brief introduction to programming in Python. For many, this is their first meaningful encounter with computer programming. Part my philosophy in designing the course is that programming is like cooking. Most people are not going to be professional chefs and so, reasonably, they learn how to cook as they go; acquiring the skills necessary to execute a desired recipe. Likewise, following a brief introduction to Python basics, additional programming techniques are introduced in each class on an as-needed basis. Students amass examples of working code that they can reference for future applications. Hopefully the course stimulates their interest in coding and

[1] The following books are excellent references on solving, simulating, and estimating macroeconomic models: Canova (2007), Stachurski (2009), DeJong and Dave (2012), and Ljungqvist and Sargent (2018).

inspires them to continue practicing on their own or in more formal settings.

Instructional materials for the course are available in the following GitHub repository:

https://github.com/letsgoexploring/computational-macroeconomics

The repository contains a set of Jupyter Notebooks for the course organized by class meeting. A Jupyter Notebook is a document that integrates computer code, in this case Python, with rich text including equations and links. One of the contributions of this paper is that I describe how to exploit the versatility of Jupyter Notebooks to teach better. The course GitHub repository also contains a sample syllabus, lecture slides, and a Python script for removing code from Jupyter Notebooks to make what are essentially electronic worksheets for class and homework assignments. I am happy to provide the course assignment files to other instructors upon request.

I am not the first to teach computational methods to undergraduates (see, for example Solis-Garcia (2018) and Neumuller, Rothschild and Weerapana (2018)). The value-added in this article is that I (1) describe an outline of the course that I teach, (2) describe how to teach it effectively with Python and the Jupyter Notebook, and (3) provide resources to other instructors who might want to teach a similar course.

2 Who Takes the Course?

Students in my winter 2019 and 2020 courses completed pre- and post-quarter surveys. Between the two courses, I had 90 students; 40 in 2019 and 50 in 2020. The surveys were *not* anonymous and were independent from the university's official course evaluation system. Completing the surveys was optional, but all 90 students completed both. The pre-quarter survey asked them to provide a little information about their backgrounds. 78 percent of the students were in their fourth year, 11 percent were third year students, 9 percent were in fifth-year students, and the remaining students were in their third year. All were in one of the three economics majors that UCI offers (Quantitative Economics, Business Economics, or Economics) and several were double majoring in Mathematics, Data Science, Education Sciences, Computer Science, Biological Sciences, Political Sciences, Biomedical Engineering, and/or Psychology.

The pre-quarter survey indicated that students had a strong desire to learn computing skills and that, in general, they entered the class with limited programming experience. More than half of the class (59 percent) had never taken a college-level computer programming course and, as shown in Figure 1, about 19 percent of students indicated that they were not even moderately familiar with *any* programming languages. Figure 2 shows students'

self-reported assessment of computer programming proficiency using the NIH Competencies Proficiency Scale.[2] 88 percent reported proficiency at novice or below. Given the students's advanced standing in their degree programs, it is unlikely that they would have obtained programing experience as undergraduates in other settings.

3 Why a Course in Computational Macroeconomics?

I have four primary learning objectives for students in my Computational Macroeconomics course. The first is for them to learn to use the Python programming language to simulate quantitative dynamic macroeconomic models that are foundational for cutting-edge research and policy analysis. These models are often covered rapidly in first-year graduate courses, but in my course, I develop them at a pace that the undergraduates have time to build intuition. I also carefully avoid certain advanced topics like Bellman equations and the details of Taylor approximation methods. Exercises emphasize visualizing data, simulating models, and providing intuitive explanations of the underlying economic mechanisms. Students leave the course with meaningful exposure to advanced macroeconomic modeling.

The second objective of the course is to practice data analysis and visualization. We work with macroeconomic data from FRED and data sets that I've prepared specifically for them. We make ample use of OLS estimation and summary statistics. Students are frequently asked to construct plots of historical data or data that they have simulated from models. I strongly emphasize the importance of creating high quality, easy-to-read visualizations. Students practice explaining the intuition behind the figures that they produce to me and to the class.

Third, I want students to develop stronger macroeconomic intuition. They do this by managing and analyzing macroeconomic data, by working with models and simulating them, and by reading articles and discussing debates. We discuss the Lucas (1976) critique and we read a debate between Edward Prescott and Lawrence Summers about the merits of RBC modeling. We explore a new Keynesian model and I relate it back to the IS-MP-AS model that most of them will have learned in intermediate macroeconomics.

The final objective of the course is to teach students Python programming skills, specifically, and computer-proficiency more generally. The course presumes no prior coding experience and students learn to code like many economists do: by acquiring skills as needed to solve specific problems. I share with the class how programming is fundamental to my research and administrative productivity. I hope that the experience develops the students's confidence, inspires them to continue developing programming skills, and ultimately makes

[2] https://hr.nih.gov/working-nih/competencies/competencies-proficiency-scale

them more productive and marketable when they graduate from UCI.

The Journal of Economic Education (JEE) recently published a collection of articles on whether dynamic stochastic general equilibrium (DSGE) models should be taught in undergraduate economics courses.[3] Solis-Garcia (2018) makes the case in favor, arguing that a DSGE-based macroeconomics course brings capable undergraduate students to the macroeconomics research frontier, provides them with marketable programming experience, and prepares them for the content that they will find in PhD programs. Strongly against the proposition, Setterfield (2018) argues that DSGE methods should not be taught to undergraduates. According to Setterfield, "the purpose of undergraduate macro is less about getting the theory exactly right than it is about basic concepts, terminology, and how policy affects the economy." The purpose is undermined, he argues, when students are taught business cycle models with no or limited role for macroeconomic policy.

Taking the middle road in the symposium, Neumuller et al. (2018) argue that since DSGE modeling is part of mainstream macroeconomic research, undergraduates should be introduced to DSGE models, but in a way that builds a "conceptual bridge" between the macroeconomics research frontier and the more intuitive, Keynesian-inspired models of the standard intermediate macroeconomics curriculum. For them, the ideal bridge makes clear to students *why* DSGE modeling has value and what are the costs and benefits of alternative approaches to macroeconomic modeling. Crucially, they emphasize that it would be inappropriate to send students a message that what they had already learned about macroeconomics is wrong.

I respect Setterfield's arguments against and the concerns about teaching DSGE methods to undergraduates. But I also agree with the justifications put forth by Neumuller et al. It can worthwhile if done in a way that builds students's intuition for macroeconomics. That is, if the DSGE models are only part of a set of broader set of course topics designed to promote deeper understanding of macroeconomics, then the pedagogical benefits outweigh the drawbacks that Setterfield identifies.

My course is a complement, not a substitute, for a strong, traditional intermediate macroeconomics foundation. It's an opportunity for students to try something new, get a different perspective on some of the ideas that they've been learning, and hopefully to get excited about programming, data analysis, and macroeconomic modeling.

[3] "The Macro Pedagogy Debate: Teaching DSGE to Undergraduates Symposium," Issue 3, 2018. Articles in the symposium include: Colander (2018), Solis-Garcia (2018), Setterfield (2018), and Neumuller et al. (2018)

4 Why Python?

I could, with similar effect, teach this course using a different program. Matlab and Octave are obvious alternatives. However, Python offers several advantages. First, Python is free to obtain so students won't have to worry about buying software licenses in order to apply what they learn in the course. Second, Python is becoming more popular within economics and many other fields.[4] Third, Python is versatile. There is an abundance of high-quality third-party libraries that can be easily and freely installed from the Python Package Index (PyPI) repository that add functionality to a basic Python installation. So by the end of the course, students will have experience with a program that is free, versatile, and increasingly used in economics, finance, and many, many other applications.

Dynare is popular and widely used software for simulating DSGE models that runs on top of Matlab or Octave. Some readers will wonder why I didn't structure the course so that I could use make use of it. An obvious reason is my preference for Python over Matlab. Another reason is that Dynare uses a Dynare-specific syntax and so even if I were to use Matlab, I'd still avoid teaching Dynare in this course. Dynare is convenient because the user enters model equilibrium conditions in a symbolic format and then Dynare parses the equations. But since the user isn't programing in the fundamental language (e.g., Matlab), they aren't practicing the fundamental language. I prefer to have the course based entirely on one language. I wrote a Python package called linearsolve specifically for this course that allows users to solve and simulate DSGE models using only Python code. See Section 6 for details about linearsolve.

There are many options for installing a Python environment for scientific computing. I use Anaconda and recommend that my students do too.[5] Anaconda is a Python package manager, a Python data science distribution, and a collection of over 1,500 open source Python packages including ones that I use in this course: Matplotlib, NumPy, SciPy, Pandas, and StatsModels. Anaconda is fast and easy to install and its package manager ensures that all installed Python package versions are compatible.[6]

Just like there are many ways to get a Python installation, there are even more options for choosing a development environment. In addition to Python and a base set of Python packages, an Anaconda installation also includes a few other programs including the Jupyter Notebook. The Jupyter Notebook is a web-browser environment that allows for a mixture

[4] See, for example the economics-oriented Python resources and examples at QuantEcon: https://quantecon.org/.

[5] https://www.anaconda.com/products/individual

[6] For instructions for installing and updating an Anaconda distribution, see the short walkthrough that I made for my YouTube channel: https://youtu.be/tzAB5swLxx0

of Python and other cells. Notebooks can be exported to HTML and other portable formats for easy sharing. From a pedagogical perspective, the Jupyter Notebook is a fantastic tool that makes it easy to teach. I use the Notebook to teach in class. I prepare a complete Notebook with instructions and code and then use a script that removes all of the code, but not the comments, from a Notebook. I share the blank Notebook with the class and we work through the coding exercise together. I use a similar approach for constructing homework assignments. Students complete blank homework Notebooks, convert the completed Notebooks to html, and upload to the course Canvas page for easy grading. I elaborate on how I use Jupyter Notebooks in the next section.

5 Course Overview

I describe the structure of the course including topics, class meeting format, and the nature of the course assignments. I discuss which particular aspects of the course seem to work well and where I think improvements could be made. The course design reflects my particular interests and the constraints of teaching on a quarter system. I would expect that other instructors would choose to omit some of what I cover and emphasize other things. I discuss alternative topic ideas that would adequately introduce students to advanced macroeconomic concepts and provide them with coding experience too. I also suggest how the course might be expanded into a semester-long course.

5.1 Course Structure

UCI is on the quarter system and so my course lasts for 11 weeks: 10 weeks of regular class instruction and a final exam in the 11^{th} week. During the first 10 weeks, each week consisted of two 80 minute lecture sessions and one 50 minute "discussion section" during which students typically worked on problems for credit. For Computational Macroeconomics, I developed a hands-on approach to instruction that emphasizes continual learning-by-doing. All meetings were held in a computer lab.

5.1.1 Lecture Format

A typical lecture is based on a prepared Jupyter Notebook. A Jupyter Notebook is a document that integrates computer code, in this case Python, with rich text including equations and links. The Notebook is a wonderful instructional tool because it allows me to write notes and instructions in HTML that students can read in advance and then we complete

the Notebook together in class. See Figure 3 for an example of a blank and complete Notebook. On the left is the blank Notebook that is distributed in advance and on the right is the completed Notebook with code and results.

Making a blank notebook would be tedious and so I have written a script that parses completed Notebooks and removes code, images, and results and leaves HTML, code comments, and code that has been marked with appropriate comment text to be left alone.[7] Therefore all I have to do to prepare a Notebook for class is construct the complete Notebook and run the script and then I have a blank Notebook that I can share with students before class and a completed one that I can share with them after.

In general, a lecture starts with a short description of an economic problem to be solved followed by me walking the students through coding examples related to the problem. Typically, a couple of times per class, I give the students a short problem to work themselves. These take about 10 minutes each. I move around the room helping students with their errors and encourage students with coding experience to help their neighbors. Students seem to really like that lectures are held in computer labs because they get immediate help.

5.1.2 Discussion Section Format

In addition to the two 80 minute lecture sessions per week, I also lead a 50 minute "discussion section" during which students work on one or more programming problems to be submitted by the end of the class for credit. The problems are based on the new material covered during the previous lecture. I encourage the students to work together but each has to submit their own work. I have not made the discussion assignments available on the public GitHub repository. I would be happy to share them with other instructors. Please contact me directly to obtain them.

Week 8 is a little different. Following a section on simulating RBC models, students are asked to read Edward Prescott's (1986) and Lawrence Summers' (1986) articles in the *Federal Reserve Bank of Minneapolis Quarterly Review*. In the discussion section that week, we discuss some of Summers' criticisms of Prescott's RBC modeling approach. Students submit answers to questions designed to help them think through the issues in the papers. The goal is to help the students think a little bit more carefully about the assumptions underlying the RBC models.

[7] File name is "make_blank_notebooks.py"; available on the course GitHub page.

5.1.3 Homework

I give weekly graded homework assignments. Homework assignments are distributed as Jupyter Notebooks that contain instructions, but no code. As mentioned, a nice feature of the Notebook is that they can be exported to HTML and viewed as a web document. UCI uses Canvas as our learning management system (LMS) and Canvas allows students to upload assignments as HTML files so it's easy to manage, review, and grade submissions. Homework assignments are available to other instructors on request.

5.1.4 Final Projects

At the end of the quarter, students are assigned to groups of five. Each group is given a variation of an RBC model and a set of instructions about what they are to do with the model. For example, model variants include an RBC mode with capital adjustment costs and an RBC model with stochastic government consumption. In all cases, the groups have to derive a new set of equilibrium conditions. In some cases, they are asked to download data from FRED to calibrate parameters that are novel to their respective models. In other cases, they are asked to compute several counterfactual impulse response simulations with different values of a key parameter. In all cases, the groups produce a 10 to 15 minute presentation of their results to be delivered during the last week of class. Groups submit their presentation slides and Jupyter Notebooks with code used to generate results. Final project assignment instructions are available to other instructors on request.

In the final project, students bring together many of the skills that they learn during the quarter and apply them to produce something meaningful. The purpose of the presentation component is to help students see programming as a means to an end. They use Python to simulate their respective models and to produce visualizations that they include in their presentations. A lot of students seem to have anxiety about the final project because they are generally averse to giving presentations and, understandably, they are not confident working with RBC models. But my students have risen to the challenge and I have been impressed by their success in completing the final project assignments and by the quality of their presentations.

5.2 Course Topics

Table 1 contains a list of the topics that I cover in each lecture. The 10-week course is roughly divided into three three-week blocks of instruction and a final week of group project presentations in the final week. In the first three-week block, students learn the basics of

Python programming. In the second, they learn how to simulate dynamic models. In the third, they analyze business cycle data and simulate business cycle models.

The first three-week block is dedicated to building basic Python proficiency and introducing specific programing techniques and Python functionality that students will use in the remainder of the course. In the first four classes, students learn about Python variable types, built-in functions, how to interpret error messages, how to work with Numpy arrays and how to make simple graphs with Matplotlib. I assign the chapter "Introduction to Programming" from Stachurski's (2009) book to be read the first week because it is an excellent, concise introduction to programming in general and Python in particular. I also direct students to books available electronically from my campus's library for additional resources.

In the third week, students are introduced to Pandas and Statmodels. Pandas is a powerful tool for managing data and Statsmodels provides statistics functionality. In Class 5, students work with cross country money growth and inflation data for over 170 countries. They use Pandas to import the data, sort the data, and compute and interpret summary statistics. Then they replicate Chart 1 from McCandless and Weber (1995).

In Class 6, I introduce the class to Statsmodels. In this class, students work with data from the Penn World Tables. To motivate the class, I show them Figure 4 from Jones and Romer (2010) depicting the positive and striking correlation between GDP per capita and total factor productivity (TFP) across countries. Students are given data on GDP per capita, human capital per capita, and physical capital per capita for a large sample of countries. They use the data to compute TFP for each country. Then, they use Statsmodels to estimate a linear regression model of log TFP on log GDP. We discuss the results in class.

In the second three-week block, we begin developing theoretical tools for model simulation. Topics include simulating linear difference equations (Class 7) and simulating nonlinear difference equations (Class 8). I use the Solow model as an application here because (1) students have already encountered it in their intermediate theory courses and (2) it provides a nice introduction into RBC modeling.

In Class 9, I teach the students how to measure business cycle fluctuations. I give them a data set with aggregate US data on GDP, consumption, investment, and trends of each statistic computed using a HP filter. Students use the data to compute the business cycle components of each quantity in the data; i.e., log deviations of each quantity from trend. They make plots of the computed series and compute and interpret summary statistics of the business cycle data including standard deviations of each series and correlations. They learn that the goal is to develop a model to explain the patterns in the data.

Next, we consider white noise and AR(1) processes in Class 10. Generating random numbers is easy with Numpy and we simulate the processes. The students apply this to

estimating an AR(1) model of the business cycle component of TFP for the US. They find that an AR(1) model fits the data well and that the coefficient on lagged TFP is estimated to be around 0.7 so that TFP fluctuations are evidently highly persistent.

In Class 11, I introduce students to dynamic optimization by studying a cake-eating problem. I present a two-period model and solve for the optimal consumption plan. We discuss the intuition and then I ask them to solve a three-period problem on their own.

Finally, in Class 12, I show the students how to use the Python module linearsolve. linearsolve is a Python module for computing linear approximations to and simulations of DSGE models. I wrote linearsolve specifically for this course and I describe it more detail in Section 6. Applications include using linearsolve to simulate an AR(1) process and to compute a stochastic simulation of the Solow growth model. This concludes the tool-building part of the course.

Having worked with the cake-eating model in Class 11 and presenting an application of the problem in a discussion section, in Class 13 we begin to explore RBC modeling. I show them slides containing results from a simulated stochastic Solow growth model. In Class 9, they learned about the properties of business cycle data. In this class, I show them that (1) the stochastic Solow model does a "reasonably" good job of explaining output and cosumption fluctuations in the US and that (2) the stochastic Solow model substantially under-predicts the volatility of investment. Also, I show them that the Solow model predicts perfect correlation of consumption and investment with output.

Next, I present an RBC model of a centralized economy that is basically that of Brock and Mirman (1972). I show them how to set up the problem and to derive the household's first-order conditions. To me, the important parts of this exercise are (1) writing the problem and understanding the economic decision, (2) deriving the first-order conditions, collecting all equilibrium conditions and verifying that the number of equations matches the number of endogenous variables, and (4) identifying which variables are endogenous and which are exogenous. Other instructors (e.g., see Solis-Garcia (2018)) want students to learn solution techniques like value function iteration, but in my experience teaching solution techniques to undergraduates is costly. It takes too long to teach them competence and does not give them a generalizable skill. The linearsolve Python module requires students to write Python code based on the equations of the RBC model and does the advanced and time-saving work of model approximation and solution.

The result of Class 13 is for students to generate on their computer screens a set of impulse response plots following technology shock in the RBC model. We analyze the generated figure. I relate the image to what they know about the tradeoff between consumption and saving. Since they have already practiced simulating AR(1) processes, so they know where

the TFP part of the simulation comes from. We interpret the rest of the impulse response plots by appealing to the model's equilibrium conditions and intuition. We start by asking what happens to the endogenous variables in the period of the TFP shock. More TFP leads to more output making more resources available for consumption and investment. Why does consumption continue to rise after period 5? Because the TFP shock causes the capital stock to grow. Why is output still above steady state in period 25 but TFP has essentially returned to zero? Because the economy accumulated capital. This is one of the key pieces of intuition that I want them to see: the household in the RBC model uses capital accumulation to store the temporary benefits of a productivity increase.

In the discussion section following Class 14, students use linearsolve to generate a stochastic simulation of the RBC model without labor. They compute summary statistics and find that the RBC model without labor does a little better than the stochastic Solow model in that it generates a significantly higher variance of investment and a lower correlation of investment with GDP. However, the model still drastically under-predicts consumption and GDP volatility and the predicted correlations of the endogenous variables could be improved.

In Classes 14 and 15, we are in a position to start replicating the results of Prescott (1986) including simulated data in Figure 3 and the correlations in Table 2. Prescott's model is basically what we have already been working with except it allows for an endogenous labor choice. By the end of the class, we compute and interpret impulse responses from the model. In Class 15, we examine how changing the autocorrelation of the TFP process changes the persistence of a shock to TFP and the shapes of the impulse responses of other variables too. The goal is to emphasize one of the benefits of computational methods: it's easy to do counterfactual simulations to see how parameterizations affect results.

We conclude Class 15 by comparing statistics computed from the simulated RBC model with statistics from US business cycle data. Students see that, relative to the RBC model without labor, Prescott's RBC model produces GDP and investment volatility that is closer to that observed in the data. I invite the students to be critical and to observe that there is still room for improvement. Predicted correlations of variables with GDP are still too low and so is consumption volatility. But the model seems to do well considering it's relative simplicity.

In the discussion section following Class 15, we wrap-up the RBC section with a discussion of the arguments in Summers (1986) critique of Prescott's RBC modeling approach. Summers offers a nice, concise critique of the RBC approach by arguing that the chosen parameter values for Prescott's simulation are suspect ("big loose tent flapping in the wind"), that Prescott doesn't provide support for why we should believe that technology has fluctuated as substantially as Prescott suggests, that Prescott doesn't explain whether his model

can predict price movements (e.g., wages or interest rates), and finally that Prescott's model completely disregards the substantial market failures that accompany business cycle downturns. I take these criticisms seriously and I encourage the students to do so also. I use these criticisms to motivate the new Keynesian modeling perspective as a response to criticisms of the RBC framework by Summers and others.

The final three class sessions are devoted to developing a new Keynesian business cycle model. I motivate the new Keynesian IS curve by presenting a simple intertemporal optimization model of saving in an endowment economy. The Euler equation in the model is the IS equation. I show that the IS equation in the new Keynesian model is analogous to the Euler equation in the RBC models that we've already considered. Hopefully they see that even though we have moved beyond the RBC framework, we did not start over from scratch.

To close the model, I provide a new Keynesian Phillips curve (NKPC) and a Taylor (1993)-type monetary policy rule. I discuss the intuition behind the NKPC but I do not derive the equation for the class. I do, however, provide them with optional notes on its derivation for any interested students to read.

My rationale for including the new Keynesian model in the course is twofold. First, after discussing with the class some of the drawbacks of the RBC modeling approach, it's appropriate to present students with an alternative perspective addressing at least some of Summers's comments to show students that we don't have to throw out all elements of the RBC modeling approach. Second, many popular intermediate macroeconomics and money and banking textbooks include a version of the new Keynesian-flavored IS-MP-AD model (Examples include Mishkin (2019),Mankiw (2019), and Jones (2020)). This is a nice time to introduce a quantitative model that is related to what the students would have seen in intermediate macroeconomics or money and banking.[8]

As with the RBC simulation, I emphasize interpreting equilibrium conditions, constructing visualizations of impulse responses to shocks, and visualizations of stochastic simulations and this occupies Classes 17 and 18. To wrap-up instruction, I use Class 18 to describe how many contemporary models combine elements of RBC and new Keynesian modeling and therefore fall under the what Goodfriend and King (1997) call the *New Neoclassical Synthesis* (NNS). I describe the equilibrium conditions in a NNS model that combines Prescott's RBC model with a new Keynesian Phillips curve (NKPC) arising from monopolistically competitive firms facing sticky prices. I give them the code to simulate the model and we analyze the impulse responses generated by the model. The point is to demonstrate a larger-scale

[8]Note that Mankiw's intermediate macroeconomics textbook has a nice quantitative model that students can simulate with spreadsheet software. To make the model tractable, he assumes a static IS relationship between the real rate and output and that agents have adaptive expectations.

model that integrates the appealing features of the two models that they learned during the quarter and to given them a sense of how macroeconomic models evolve.

5.3 Alternative Topic Ideas

The course content reflects my interests and the amount of content is limited by the length of the 10-week quarter. Here are some thoughts on what I might include if I had the additional five weeks of a semester. First, it would be nice to include an endogenous growth model for the class to simulate. Jones (2020) presents a nice computable endogenous growth model in his intermediate textbook. Second, in the RBC section of the course, we could take time to cover some extensions to the RBC model including things like capital adjustment costs or shocks to government purchases. As it is, these topics are only included among final project topics, but it would be nice to have class time to teach and discuss them. Third, in the new Keynesian section of the course, some additional time would give us time to review the evidence for NKPC and monetary policy rules. I would cover Taylor's (1993) paper and give students an exercise to replicate Figure 1 from his paper. Fourth, I would cover discretionary monetary policy as described in Clarida, Galí and Gertler (1999).

Of course some instructors may prefer to go more in depth on any of the topics that I have proposed so far others may wish to include entirely different topics and methods. For example, I've already mentioned that Solis-Garcia teaches value function iteration in his course at Macalester College. Another opportunity is the Diamond-Mortensen-Pissarides (DMP) of unemployment. Bhattacharya, Jackson and Jenkins (2018) describe how to teach the DMP model to undergraduates. They provide a model that can be computed as easily as the Solow model and they provide data that can be used in empirical exercises on the website for their paper.[9] Additionally, the course could be modified to include forecasting and VAR methods; both of which could be taught with the Statsmodels Python module.

Finally, in many cases it may be worthwhile to require a programming prerequisite. At UCI, enrollment constraints in even introductory computer science courses makes it infeasible for me to require that my students take a programming course in advance. But at schools where it's an option, adding the prerequisite would certainly free up time on the course schedule.

[9] https://www.briancjenkins.com/dmp-model/

6 linearsolve

I developed new software specifically for this course. Solving dynamic stochastic macroeconomic models requires advanced computational techniques and so I wrote a Python module called linearsolve, available to any Python user from the Python Package Index, that computes linear approximations to DSGE, computes the solutions to the approximated models, and produces customizable simulations.[10] Pedagogically, the program is useful because it requires that students properly enter equilibrium conditions as Python code so they practice economics and Python at the same time.

Figure 4 depicts of an example of how to use the code to compute impulse responses to a technology shock in an RBC model with log utility and no labor. For more examples of how to use linearsolve, please see the documentation:

https://www.briancjenkins.com/linearsolve/docs/build/html/index.html

7 Conclusion

Designing and teaching my Computational Macroeconomics course has been a wonderful experience. I am convinced that the content is valuable and that it has been well-received by my students. Indeed, one student who took the course in winter 2019 described the course as a "game-changer" for her in an interview with an online campus publication (Byrd 2019). This student's sentiment was echoed by others in the (non-confidential) post-quarter survey that I asked my students to complete. Given my students's enthusiasm for the course, I would like to see computational methods built in to more courses in a deliberate and unified way. While computational methods are not, strictly speaking, economics, they are nonetheless important to economics. And to the extent that many economists are practitioners, we can provide great value to our students at low cost by giving them skills that will transfer over to other disciplines and activities while also enhancing their learning of economics.

[10] Full documentation available here: https://www.briancjenkins.com/linearsolve/docs/build/html/

References

Bhattacharya, Arghya, Paul Jackson, and Brian C. Jenkins, "Revisiting unemployment in intermediate macroeconomics: A new approach for teaching Diamond-Mortensen-Pissarides," *The Journal of Economic Education*, 2018, *49* (1), 22–37.

Blumenstyk, Goldie, "Liberal-Arts Majors Have Plenty of Job Prospects, if They Have Some Specific Skills, Too," *The Chronicle of Higher Education*, Jun 2016. (accessed: September 16, 2020).

Brock, William A and Leonard J Mirman, "Optimal economic growth and uncertainty: The discounted case," *Journal of Economic Theory*, 1972, *4* (3), 479 – 513.

Byrd, Christine, "Pantsuit generation," https://www.socsci.uci.edu/newsevents/news/2019/2019-05-28-pantsuit-gen.php May 2019. (accessed: September 16, 2020).

Canova, Fabio, *Methods for Applied Macroeconomic Research*, Princeton, NJ: Princeton University Press, 2007.

Clarida, Richard, Jordi Galí, and Mark Gertler, "The Science of Monetary Policy: A New Keynesian Perspective," *Journal of Economic Literature*, 1999, *37* (4), 1661–1707.

Colander, David, "Teaching DSGE to undergraduates symposium: Introduction," *The Journal of Economic Education*, 2018, *49* (3), 224–225.

DeJong, David N. and Chetan Dave, *Structural Macroeconometrics*, 2nd ed., Princeton, NJ: Princeton University Press, 2012.

Goodfriend, Marvin and Robert King, "The New Neoclassical Synthesis and the Role of Monetary Policy," *NBER Macroeconomics Annual*, January 1997, pp. 231–283.

Jones, Charles I., *Macroeconomics*, 5th ed., New York: W.W. Norton & Company, 2020.

___ **and Paul M. Romer**, "The New Kaldor Facts: Ideas, Institutions, Population, and Human Capital," *American Economic Journal: Macroeconomics*, January 2010, *2* (1), 224–45.

Ljungqvist, Lars and Thomas J. Sargent, *Recursive Macroeconomic Theory*, 4th ed., Campridge, MA: The MIT Press, 2018.

Lucas, Robert E., "Econometric policy evaluation: A critique," *Carnegie-Rochester Conference Series on Public Policy*, 1976, *1*, 19 – 46.

Mankiw, N. Gregory, *Macroeconomics*, 10th ed., Worth Publishers, 2019.

McCandless, George T. and Warren E. Weber, "Some monetary facts," *Federal Reserve Bank of Minneapolis Quarterly Review*, Summer 1995, pp. 2–11.

Mishkin, Frederic S., *The Economics of Money, Banking, and Financial Markets*, 12th ed., New York: Pearson, 2019.

Neumuller, Seth, Casey Rothschild, and Akila Weerapana, "Bridging the gap between undergraduate and graduate macroeconomics," *The Journal of Economic Education*, 2018, *49* (3), 242–251.

Prescott, Edward C., "Theory ahead of business cycle measurement," *Federal Reserve Bank of Minneapolis Quarterly Review*, Fall 1986, pp. 9–22.

Setterfield, Mark, "Maybe you can, but perhaps you shouldn't! saving undergraduate macroeconomics from DSGE modeling," *The Journal of Economic Education*, 2018, *49* (3), 237–241.

Solis-Garcia, Mario, "Yes we can! Teaching DSGE models to undergraduate students," *The Journal of Economic Education*, 2018, *49* (3), 226–236.

Stachurski, John, *Economic Dynamics*, Cambridge, Massachusetts: MIT Press, 2009.

Summers, Lawrence H., "Some skeptical observations on real business cycle theory," *Federal Reserve Bank of Minneapolis Quarterly Review*, Fall 1986, pp. 23–27.

Taylor, John B., "Discretion Versus Policy Rules in Practice," *Carnegie-Rochester Conference Series on Public Policy*, 1993, *39*, 195–214.

8 Tables

Table 1: **Outline of topics**. List of topics covered in each lecture.

Class #	Topic(s)
1	Introduction to course and the Jupyter Notebook. Short coding examples.
2	Python basics. Built-in functions, error messages, built-in object types, math.
3	Numpy module. Math, arrays, and random number generation.
4	Plotting with the Matplotlib module.
5	Data management with the Pandas module.
6	Statistics with Statsmodels module.
7	Simulate linear first-order and linear higher-order difference equations.
8	Simulate nonlinear first-order difference equations and systems of difference equations. Solow growth model.
9	Introduction to business cycle data
10	White noise and AR(1) processes.
11	Introduction to Dynamic Optimization: A Two-Period Cake-Eating Problem
12	The linearsolve module
13	Introduction to Real Business Cycle Modeling
14	Prescott's (1983) Real Business Cycle Model: Impulse responses
15	Prescott's (1983) Real Business Cycle Model II: Stochastic simulation
16	Introduction to New-Keynesian Business Cycle Modeling
17	New-Keynesian Business Cycle Modeling: Impulse responses
18	New-Keynesian Business Cycle Modeling: Stochastic simulation
19	Group project presentations.
20	Group project presentations.

9 Figures

Figure 1: **Programming languages for which students had prior experience.** Percentage of students responding that they are at least "moderately familiar" with each language. "Other" includes Microsoft Visual Basic, Mathematica, and EViews.

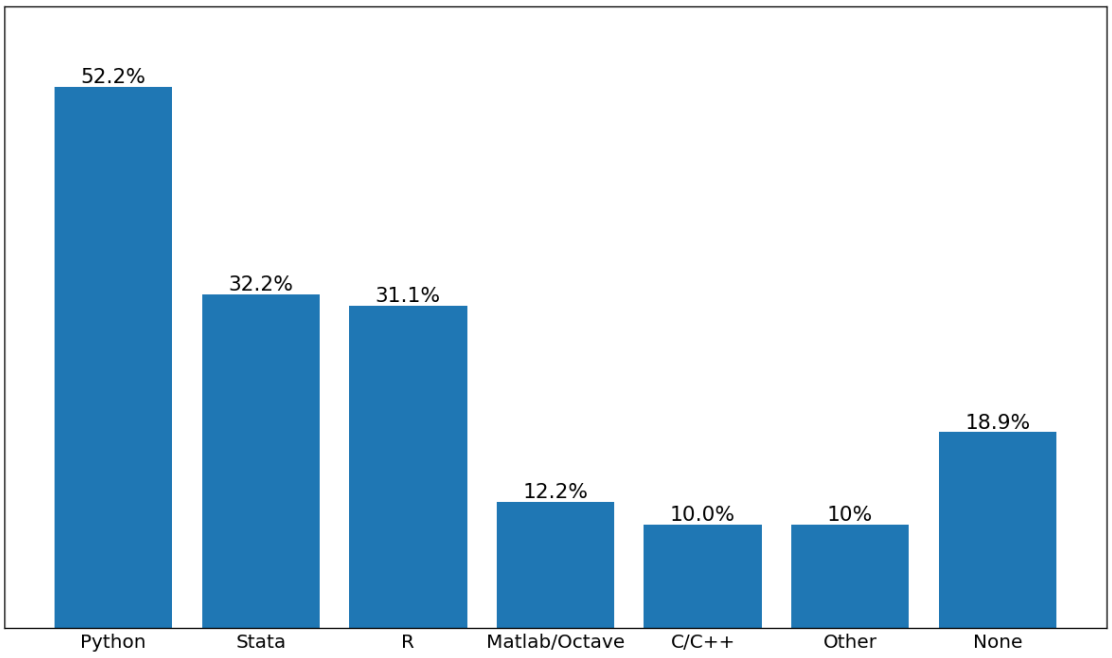

Figure 2: **Student pre-course programming proficiency.** Student self-assessment of computer programming proficiency. Options were (from least to most): No experience, fundamental awareness, novice, intermediate, advanced, expert. 90 percent reported proficiency at novice or below.

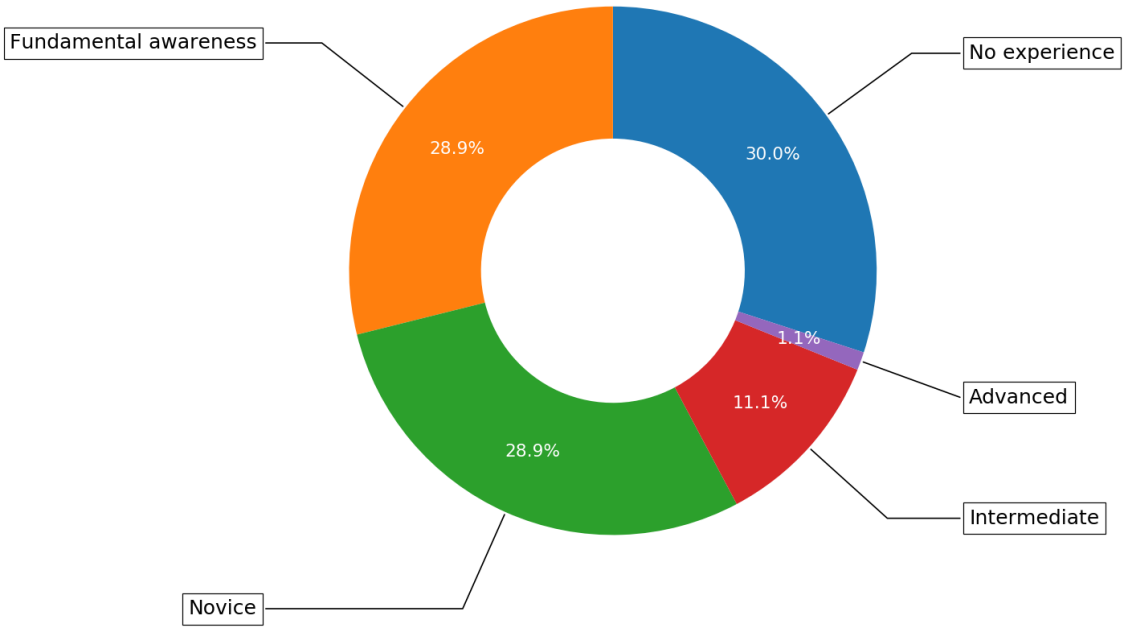

Figure 3: **Example of a Jupyter Notebook.** On the left is a blank Notebook that I provided students in advance of lecture. On the right is the Notebook with code and output that was completed in class.

The AR(1) Process

A random variable y_t is an *autoregressive process of order 1* or AR(1) process if it can be written in the following form:

$$y_t = \rho y_{t-1} + \epsilon_t, \tag{2}$$

where ρ is a constant and $\epsilon \sim \text{WN}(0, \sigma^2)$. The AR(1) process is the stochastic analog of the first-order difference equation where the random variable ϵ_t replaces the exogenous variable w_t.

Example

Simulate an AR(1) process for 101 periods ($t = 0, \ldots, 100$) using the following parameter values:

$$\rho = 0.5 \tag{3}$$
$$\sigma = 1 \tag{4}$$
$$y_0 = 0 \tag{5}$$

Plot the simulated values for y.

```
In [ ]: # Set the seed for the random number generator to 126

        # Initialize values for T, y0, rho, and sigma

        # Initialize an array of zeros for y

        # Set the first value of y equal to y0

        # Create a variable called 'epsilon' equal to an array containing T draws f

        # Iterate over t in range(T-1) to compute y

        # Plot y
```

The AR(1) process wtih $\rho = 0.5$ seems to fluctuate around the value $y = 0$ and so the process appears to be stable

Example

Simulate an AR(1) process for 101 periods ($t = 0, \ldots, 100$) using the following parameter values:

$$\rho = 1.5 \tag{1}$$
$$\sigma = 1 \tag{2}$$
$$y_0 = 0 \tag{3}$$

Plot the simulated values for y.

```
In [ ]: # Set the seed for the random number generator to 126

        # Initialize values for T, y0, rho, and sigma
```

The AR(1) Process

A random variable y_t is an *autoregressive process of order 1* or AR(1) process if it can be written in the following form:

$$y_t = \rho y_{t-1} + \epsilon_t, \tag{2}$$

where ρ is a constant and $\epsilon \sim \text{WN}(0, \sigma^2)$. The AR(1) process is the stochastic analog of the first-order difference equation where the random variable ϵ_t replaces the exogenous variable w_t.

Example

Simulate an AR(1) process for 101 periods ($t = 0, \ldots, 100$) using the following parameter values:

$$\rho = 0.5 \tag{3}$$
$$\sigma = 1 \tag{4}$$
$$y_0 = 0 \tag{5}$$

Plot the simulated values for y.

```
In [3]: # Set the seed for the random number generator to 126
        np.random.seed(126)

        # Initialize values for T, y0, rho, and sigma
        T = 101
        y0 = 0
        rho = 0.5
        sigma = 1

        # Initialize an array of zeros for y
        y = np.zeros(T)

        # Set the first value of y equal to y0
        y[0] = y0

        # Create a variable called 'epsilon' equal to an array containing T draws f
        eps= np.random.normal(loc=0,scale=sigma,size=T)

        # Iterate over t in range(T-1) to compute y
        for t in range(T-1):
            y[t+1] = rho*y[t] + eps[t+1]

        # Plot y
        plt.plot(y,lw=2)
        plt.grid(linestyle=':')
```


The AR(1) process wtih $\rho = 0.5$ seems to fluctuate around the value $y = 0$ and so the process appears to be stable

Figure 4: **The linearsolve Python module.** Example of how to approximate, solve, and simulate an RBC model with linearsolve.

In [1]:
```
# Import linearsolve, numpy, pandas, and matplotlib.pyplot
import linearsolve as ls
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# Create a pandas series with parameter values
parameters = pd.Series()
parameters['alpha'] = .35
parameters['beta'] = 0.99
parameters['delta'] = 0.025
parameters['rhoa'] = .9

# Define a function that evaluates the equilibrium conditions of the RBC model
def equilibrium_equations(variables_forward,variables_current,parameters):

    # Parameters
    p = parameters

    # Variables
    fwd = variables_forward
    cur = variables_current

    # Stack equilibrium conditions into a numpy array
    return np.array([
            p.beta/fwd.c*(p.alpha*cur.a*fwd.k**(p.alpha-1)+1-p.delta) - 1/cur.c,
            cur.c + fwd.k - (1-p.delta)*cur.k - cur.a*cur.k**p.alpha,
            p.rhoa*np.log(cur.a) - np.log(fwd.a)
        ])

# Initialize the model
model = ls.model(equations = equilibrium_equations,
                 n_states=2,
                 var_names=['a','k','c'],
                 shock_names=['e_a','e_k'],
                 parameters = parameters)

# Use linearsolve to compute the steady state numerically
model.compute_ss([1,1,1])

# Find the log-linear approximation around the non-stochastic steady state and solve
model.approximate_and_solve()

# Compute impulse responses and plot
model.impulse(T=41,t0=5,shocks=None)

fig = plt.figure(figsize=(12,4))
ax1 =fig.add_subplot(1,2,1)
model.irs['e_a'][['a','k','c']].plot(lw='5',alpha=0.5,ax=ax1).legend(loc='upper right',ncol=3)
ax2 =fig.add_subplot(1,2,2)
model.irs['e_a'][['e_a','a']].plot(lw='5',alpha=0.5,ax=ax2).legend(loc='upper right',ncol=2)
```

Out[1]: <matplotlib.legend.Legend at 0x10160b94e0>