Teaching Nash Equilibrium with Python

Allison Oldham Luedtke

Department of Economics, Saint Michael's College, Colchester, Vermont, United States

Dr. Luedtke can be reached by phone at (540) 905-2622 or by email at aluedtke@smcvt.edu.

Teaching Nash Equilibrium with Python

This paper describes an assignment in an undergraduate Game Theory course in which students develop basic computer science skills while applying their game theory knowledge to real world situations. In the assignment, students create a simultaneous-choice game with two players and write a computer program to find all of the pure strategy Nash Equilibria of the game. This assignment is an efficient way for undergraduate economics students to gain valuable computer science skills without having to take classes outside of the economics major and without economics faculty having to restructure entire courses or curricula. This paper provides tips and resources for implementing the assignment, including all of the computer program code.

Keywords: game theory; Nash equilibrium; computer science;

JEL codes: A22; C70, C63

Introduction

The concept of a Nash equilibrium is one of the most important definitions that students learn in undergraduate Game Theory and economics as a whole. Simultaneously, the definition of a Nash equilibrium is complex and students often struggle with it. The textbook that I use in my Game Theory course, "Games of Strategy," by Avinash Dixit, Susan Skeath, and David Reiley Jr., provides the following definition of a Nash equilibrium: "A configuration of strategies (one for each player) such that each player's strategy is best for him, given those of the other players." (Dixit, Skeath, and Reiley 2015) It is critical to the understanding of economics that students have a deep, intuitive understanding that a Nash equilibrium is a set of choices for each player that cannot be improved given what other people have chosen. In this paper, I present an assignment that uses computer programming to help students internalize this definition.

Whether economics undergraduates become financial consultants, political advocates, or go on to graduate school, a basic knowledge of computer science is necessary for success. ("Two Sciences Tie the Knot" n.d.) But for most economics majors, it is not feasible to add a computer science major to their already full academic plates. Nor is it practical for most economics departments to devote faculty resources to a "Computer Science for Economics" elective. As such, in order to ensure that economics majors are exposed to computer programming, faculty must incorporate it into our existing curriculum. In this paper, I describe an assignment that I gave to students in my undergraduate Game Theory course in which students with no prior computer science experience write a program in Python to find the pure strategy Nash equilibrium in a simultaneous discrete-choice game, often referred to as a matrix game.

In the assignment, students create a matrix game with two players. They then write a computer program in the programming language, Python, that finds any Nash equilibria in pure strategies in their game. We work together as a class to outline the algorithm and then students independently implement their own program. The assignment introduces as little new terminology as possible so as to require as little computer science overhead as possible. Students turn in a description of their game, their Python code, the output that the code creates, and a discussion of the results.

Experience with computer science is not only helpful because of the increasing digitalization of jobs, but also because of the skills that it develops. Students with degrees in economics are already sought after because of their problem-solving skills. Computer programming requires these same skills as well as communication and abstraction. The ability to break down a problem into lines of code for a computer to process is an excellent complement to the existing economics major skill set. These problem-solving, communication, and abstraction skills are particularly honed by working with a definition as complex as that of a Nash equilibrium. Students break down the process of identifying a Nash equilibrium. Students break down the process of identifying a Nash equilibrium into its foundational components and must explain to a computer what they mean.

The assignment is designed to require as little time introducing computer science topics as possible. No important Python packages are required so there is no complicated header needed at the top of students' code. My class was able to complete this assignment with three lecture hours devoted to background and outlining the algorithms. It could be done with less time in class if the instructor wants to leave more for the students to complete independently. Additionally, this assignment is easier to grade than a traditional Game Theory problem set. Grading is discussed in more detail in the next section. Resources to assist instructors with the implementation of this assignment are discussed in Section 3, "Tips and Resources."

Not only did students perform well on this assignment – all but two students received a B or better – they also felt that it improved their resumés and job prospects. Multiple students reported discussing this project with potential employers in interviews for jobs and internships. Students also seemed to legitimately enjoy the assignment. I provided students in this course with several options for their final project, one of which was an extension of this assignment. The final project assignment required them to write a new computer program that built on what they had produced previously. The vast majority of students chose to complete this option for their final project. One student even mentioned it as her favorite aspect of the class on my teaching evaluation for the course.

To best prepare students for life after college – whatever form that may take – we need to expose them to computer programming at some point throughout their economics major. This paper describes an assignment that accomplishes this without requiring any prior experience on the students' part and with little extra work for instructors. Assignments of this type will become increasingly necessary as departments are doing more with less.

Assignment

In this assignment, students solve a matrix game by writing a computer program that finds Nash equilibria. Students create a simultaneous game with two players (a matrix game) in which the players have enough choices so as to make it tedious to solve the game by hand using standard techniques, such as, cell-by-cell inspection or iterated elimination of dominated strategies. The students then write a computer program in Python that will find all of the pure strategy Nash equilibria in a given matrix game. The final product of this assignment – what the students turn in – consists of the following elements: (1) a description of their simultaneous game, (2) their python code, (3) the output when their code is run, and (4) a discussion of the solution of the game. In the description of the game, students specify who they players are, what choices are available to them, and what the payoffs for each possible outcome are. In this description, they also include the matrix that represents their game. I encouraged students to be creative in the creation of their games because, when they delineate each step of the solution process using their own terms instead of mine or the textbook's, they gain a deeper grasp of the economic meaning of them. Students turn in their Python code as a Python file so that I can run it to ensure that it works correctly. In their discussion of the solution found by their code, students explain whether any of the Nash equilibria are socially optimal, whether they make sense in the context that the students originally envisioned, what would need to change about the game to achieve a different outcome, and any other observations that the students wish to share. Students turn in two files: one is a typed document containing the description of the game, the output of the Python code, and the discussion and the second is their code.

Let us use the following example throughout the remainder of the paper. Two friends are hosting a dinner party. One friend must provide the main dish and the other friend must provide the side dish. The two friends have different preferences for main-side combinations. The payoffs to each friend for each possible combination are displayed in matrix format in Table 1. Many students used contexts similar to this in their own games. Often their games involved an interaction between them and a roommate, sibling, or teammate.

The Python code consists of three main components: inputting a matrix of payoffs, checking each cell for the characteristics of a Nash equilibrium, and printing out the cells that are equilibria. Each of these components forces students to break down their real-world scenario into the foundational elements of a game, payoffs, and equilibrium. We worked collectively as a class to come up with pseudocode, which I wrote on the board, and then I typed and explained the actual Python code on the classroom computer display. Because the students had no prior experience with programming, with the correct code displayed on the projection screen, the translation onto their own computers was sufficiently difficult to present a challenge but students felt comfortable enough to try and fail a few times before they got it entirely correct.

Because I wanted to introduce as little Python terminology as possible to complete this assignment, I chose to use Python's innate matrix structure, rather than to import a package like Numpy, which would have more elegant and complex matrix operations. This required the use of a cumbersome number of brackets and parentheses, but saved at least an extra day of explaining how to import Python packages, which would not add anything to the students' understanding of Game Theory. The code to input the game matrix for the game represented in Table 1 is as follows:

```
M = [[[3,1],[2,3],[10,2]],
     [[4,5],[3,0],[6,4]],
     [[2,2],[5,4],[12,3]],
     [[5,6],[4,5],[9,7]]]
```

Each pair of numbers represents the payoffs to Player 1 and Player 2, respectively. In class, I did a single example and left it to students to independently input their own game matrices.

A pure strategy Nash equilibrium in this type of game consists of a choice by each player such that, given one player's choice, the other player cannot improve his payoff by changing his choice. Consider the first cell, [3, 1]. Given that Player 1 is choosing to bring Roast Beef, Player 2 can improve his payoff from 1 to 3 by switching to bringing Macaroni & Cheese rather than Mashed Potatoes. As such, the first cell is not a Nash equilibrium. The middle cell in the third row, [5, 4], is the only Nash equilibrium. There are several ways to identify the pure strategy Nash equilibria in a matrix game, including cell-by-cell inspection, iterated removal of dominated strategies, and best-response analysis. Of these, the method that students struggle with the most is cell-by-cell inspection. This method requires the deepest knowledge of the Nash equilibrium definition and at the same time is well suited for computers. To find all Nash equilibria using cell-by-cell inspection, you check each cell of the matrix to see if it meets the definition of a Nash equilibrium. The algorithm that the students write visits each cell of their game matrix and checks to see if either player can be made better off, given the other player's choice. If neither player can be made better off, the algorithm records that cell as an equilibrium. If either player can be made better off, the algorithm records that cell as not an equilibrium. Every cell of the matrix is visited and labelled as "equilibrium" or "not." In order to achieve this, students must write an algorithm that visits each cell, queries the payoffs for each player, compares them to the relevant other payoffs, and makes the correct determination about the equilibrium status of that cell. The pseudocode for this is as follows:

```
for each row:
    for each column:
        check if row player can be made better off
        check if column player can be made better off
        if neither can be:
            label as equilibrium
        if any can be:
            label as not an equilibrium
```

The final task of the computer program is to print any cells that are Nash equilibria in the output window. I chose to make this a second separate task of the algorithm, rather than to have it print the Nash equilibria as it found them. I did this in part to make the algorithm more modular, so that students could use portions of it for other projects, and in part because I wanted to teach each task separately. This method requires the use of a second matrix, the same size as the game matrix, that contains the equilibrium label for each cell. As the algorithm determines the equilibrium status of each cell, it stores the appropriate label in the label matrix. When the equilibrium-determination portion of the algorithm finishes, the label matrix contains the equilibrium status of each cell. To print out any Nash equilibria, the algorithm must now parse the label matrix and print out any cells of the game matrix that correspond to a label of "equilibrium." Every label starts out as a "0," if it is a Nash equilibrium it is switched to a "1," and if it is not a Nash equilibrium it is switched to a "2." This allowed students to practice the critical computational skill of converting qualitative information ("is a Nash equilibrium") into numerical data ("1"). The pseudocode for printing out any Nash equilibrium cells is as follows:

```
for each row:
    for each column:
        if label(row,col) == 2:
            print cell(row,col)
```

This assignment can be adapted to fit into most grade categories. I used it as one of five homework assignments throughout the semester. Depending on how much you cover together in class and how much you leave to the students to complete independently, in conjunction with the level of programming experience of your students, this assignment can be a small classwork assignment, a homework assignment, a large term project, or even an extra credit assignment. In order to receive a perfect score on this assignment, students had to turn in all components, explain all components of their game, correctly identify the Nash equilibria of their game – both in their description of the game and computationally – and discuss the appropriate outcomes. It is easy to identify the components of the game in the description (players, choices, payoffs, equilibria) and the discussion (social optima, potential changes). The only numerical component that requires checking is the correct identification of the Nash equilibria. This can be done very quickly by copying their game matrix and label matrix into your code and running it. If their list of equilibria matches yours, they correctly identified them.

In addition to building computation and abstraction skills, this assignment forces students to consider carefully what constitutes a good example of a simultaneous discrete choice game, and thus what would make a good test question about simultaneous discrete choice games. No matter how many times I tell my students that the best way to prepare for a test is to create their own practice questions, they don't believe it until they are forced to do it. This is one way to accomplish that.

Tips and Resources for Instructors

To easily run Python code on a classroom computer and on your students' computers without having to install many new software packages, I recommend making use of the online Python editor and compiler, Jupyter Notebook. Alternatively, if you would prefer to use an application that does not require an internet connection, I recommend the Python editor and compiler, Thonny. It is easy for students to install and use without complicated installation procedures and works well for small, pedagogical applications such as this one.

The actual Python code for the example represented in Table 1 is replicated in its entirety below. The Python designation for a comment is "#", as such, any lines that begin with a "#" are my comments.[1]

```
M = [[[3,1],[2,3],[10,2]],
     [[4,5],[3,0],[6,4]],
     [[2,2],[5,4],[12,3]],
     [[5,6],[4,5],[9,7]]]
Label = [[0,0,0],
         [0,0,0],
         [0,0,0],
         [0,0,0]]

for row in range(4):
    for col in range(3):
        #check row payoffs
        for r in range(4):
            if Label[row][col] == 0:
                if M[r][col][0] > M[row][col][0]:
                    #this is not a Nash Equilibrium
                    Label[row][col] = 2
        #only bother checking if it's still unset
        if Label[row][col] == 0:
            for c in range(3):
                if Label[row][col]==0:
                    if M[row][c][1] > M[row][col][1]:
                        #not a Nash eq.
                        Label[row][col] = 2

        #if we made it all this way without setting to 2, it's a
Nash Eq
        if Label[row][col] == 0:
            Label[row][col] = 1

for the_row in range(4):
    for the_col in range(3):
        if Label[the_row][the_col] == 1:
            print(M[the_row][the_col])
```

[1] There are no tips or resources that I can provide that can prepare you for calling it a "pound sign" and your students not understanding what you mean until you say "you know, a hashtag."

If you copy and paste this code into a Python editor and run it, it will output "[5, 4]." Additionally, a downloadable Python (.py) file is available on my website, allisonoldhamluedtke.com. Additional resources can be found there, as well, including the assignment description that I gave students.

The assignment was designed to allow for expansions and further applications later in the semester, if the students expressed interest. Students were given a slate of choices for their final project in the class and one of the options was to complete another programming assignment. A majority of students in the class chose this option. Specifically, I asked students to modify their code to randomly generate a game matrix and use their existing code to find any Nash equilibria. The code to accomplish this can be found on my website, as well. There are many possible expansions to this assignment and I encourage you to come up with your own.

Conclusion

This paper presents an assignment for undergraduate Game Theory students. In the assignment, students create a matrix game based on their lives and write a computer program in Python that finds all of the pure strategy Nash equilibria of their game. It is designed to take only three to six hours of instruction, so that it can be added to most undergraduate Game Theory courses without faculty needing to dramatically restructure their courses. Experience with computer programming is swiftly becoming necessary for most careers that an economics graduate would pursue. This assignment efficiently provides students with workable knowledge of programming in Python.

Acknowledgments: I want to express my appreciation to my first ever game theory students for being so willing to try something new.

References:

Dixit, Avinash K., Susan Skeath, and David H. Reiley. 2015. *Games of Strategy*. Fourth edition. New York: W.W. Norton & Company.

"Two Sciences Tie the Knot." n.d. MIT News. Accessed October 1, 2019. http://news.mit.edu/2017/mit-creates-new-major-computer-science-economics-data-science-0904.

Table 1. Example Game Matrix.

		Player 2 (2nd Friend)		
		Mashed Potatoes	Macaroni & Cheese	Broccoli
Player 1 (1st Friend)	Roast Beef	3, 1	2, 3	10, 2
	Turkey	4, 5	3, 0	6, 4
	Meatloaf	2, 2	5, 4	12, 3
	Lasagna	5, 6	4, 5	9, 7