

PhD Industrial Organization: Homework 2

Due: Start of class Tuesday April 11
You can work in groups of up to 2 people.

Instructions

- You can work in groups of 2 people. It might be a good idea to code up the solutions individually and then compare answers so that you both get experience with the code and understand the solutions.
- As you work through the questions, I strongly recommend using Git to version control your code (and pushing to your Github account as you make changes). Git is also great for collaboration if you are working in a group of 2.
- Please hand in both your solutions and also the code you used to generate the solutions. You can work in whichever programming language you want.
- Good luck!

Background


In this homework assignment you will be coding up a replication of Rust (1987). Specifically, I want you to attempt to replicate (or, at least, produce estimates that closely resemble) the estimates for ‘Group 4’ buses in Table IX of the paper (on page 1021) for the discount factor $\beta = 0.9999$ and fixed point dimension = 90. The estimates I want you to reproduce are for the replacement cost RC and the cost function θ_{11} ; more details are below.

Data

There is one dataset:

- **group_4.csv:**
- This is a nicely formatted version of the older rust data files, which are in a more difficult format to work with.
- For some reason, there are some minor discrepancies with this dataset and the one described in the paper. When I tried to replicate the paper myself the final estimates were within a few

percentage points of the ones in the paper, e.g a replacement cost of something like 10.207 rather than the number in the paper of 10.0750. So, just be aware that your replication will likely get very close but not exactly to the paper numbers.

- This data is for the ‘Group 4’ buses and so this is the column in Table IX of the paper your estimates should correspond to.
- The descriptions of the variables are:
 - *Bus_ID*: ID of the bus
 - *period*: period since the start of the data
 - *state*: binned state of mileage (i.e. in 5000 mile increments) 
 - *mileage*: raw mileage number
 - *decision*: Harold Zurcher’s decision to replace the bus engine. If he does replace then engine this =1 and you can see that the mileage will reset to 0 the following period.

Good resources if you have questions

- It is extremely likely that you will run into some confusion about how all the pieces in the algorithm fit together. After all, understanding this is the point of the assignment!
- The paper is a classic in the literature, and much has been written online about how to code it up in various languages. Rather than provide you with specific references, I suggest just doing a quick google search which will lead to many examples and tutorials.
- In the homework below I also provide some computational tricks/hints.
- You can also talk to me after class, send me an email to set up a meeting, or just ask the question by email: nvreugde@asu.edu.

1. Rust (1987) replication

Consider the Rust (1987) model that we discussed in detail in class (see the lecture slides if you need to be reminded of all the details). Assume the following specifications, which correspond to the model estimated in Table IX of the paper:

- Cost function $c(x, \theta_1) = 0.001\theta_{11}x$
- Discount parameter: $\beta = 0.9999$
- Idiosyncratic errors are i.i.d. logit
- Fixed point state space discretization is dimension 90
- Transition parameters given as follows:
 - $\theta_{30} = 0.3919$
 - $\theta_{31} = 0.5953$
- Parameters that are unknown and need to be estimated from the data:
 - RC : replacement cost
 - θ_{11} : parameter in the cost function (given above)

Instructions:

- a. Estimate the replacement cost RC and the cost function parameter θ_{11} by maximum likelihood using the nested fixed-point method for computing the value functions. (More specifically, follow the Rust (1987) algorithm to replicate/find close numbers to the estimates for Group 4 buses in Table IX.) No need to compute standard errors.
- b. Provide pseudo-code for an alternative CCP method to the nested fixed-point method. Discuss one benefit and one limitation of this CCP method as compared to the full solution method you used in (a.). (To emphasize: no need to write the exact code or compute values with this method, pseudo-code is enough.)

Notes/tips/tricks (mainly for part a.):

- For the state transition parameters $(\theta_{30}, \theta_{31})$, use the ones specified above that come from the paper when estimating your model.
- Computationally, you are likely to run into an overflow error if you are not careful. Here is a description about what this error is and the solution.

- What is an overflow error? Essentially this comes from putting large absolute numbers into exponentials. E.g. $\exp(3000)$ is a huge number that can run into machine precision errors when you try to compute directly, leading to incorrect results or the code returning an infinity. These errors can be common in discrete choice problems and particularly problems like Rust (1987) involving dynamics where the value of each choice depends on a value function which can be large. In particular, they may arise when you compute choice probabilities or value functions where you need to exponentiate these absolute values (e.g. the multinomial logit formula).
- How to solve overflow errors? You should work with equivalent but slightly different formulas for the multinomial logit and logsum formulas. These basically exploit that it is *differences* in payoffs that affect choices and not the absolute values.
- For the multinomial logit formula normalize each choice with a constant c , which you can set to a carefully chosen value. For example, one that might work is $c = \text{maximum payoff of all choices}$:

$$\frac{\exp(u_j)}{\sum_k \exp(u_k)} = \frac{\exp(u_j - c)}{\sum_k \exp(u_k - c)}$$

- For the log-sum formula (which you will use when computing value functions) use the following normalization by a constant c :

$$\log \sum_k \exp(u_k) = \log \left(\sum_k \exp(u_k - c) \right) + c$$

- For derivations and further explanations see this website: <https://blog.feedly.com/tricks-of-the-trade-logsumexp/>
- Computationally, the bottleneck in terms of speed is doing the value-function iteration. To speed this up people usually leverage the fact that the state is discretized and so specify a transition matrix, then use matrix multiplication to update the value functions. Many references to examples of this can easily be found online. E.g.: <https://mark-ponder.com/tutorials/discrete-choice-models/dynamic-discrete-choice-nested-fixed-point-algorithm/>
- For the value function loop tolerance, I am currently using 0.0001. I found smaller values didn't make much difference in this model. Note that I am just letting you know this to speed up your homework code, in general this is quite a loose tolerance that may run into errors more generally and should be checked in other applications.