Problem Set #8

ECON 833, Prof. Jason DeBacker
Due Tuesday, December 2, 1:15 p.m.

For this problem set, you will replicate some of the findings from Cooper and Ejarque, "Financial frictions and investment: requiem in Q" (*RED*, 2003). This paper tests classical investment theory and will be a good exercise in both dynamic programming and using simulated method of moments (SMM).

Specifically, I want you to replicate the results in Table 3, the model with costly external finance. You won't need any data – you can use the same values for the moments reported in Table 3 of the paper. You'll use a simulated method of moments (SMM) approach to estimate the parameters of the model. The general algorithm is as follows:

1. Choose a set of parameters to estimate (as in Table 3, you will estimate α, γ, ρ, σ, and $\tilde{\phi}_0$ – the other model parameters, δ, β, ϕ_1 will be calibrated using the same values as in Cooper and Ejarque (2003)).
2. Simulate the model to generate data on investment, Q, and cash flow from the simulated panel of firms.
3. Compute the moments from the simulated data that correspond to the moments reported in Table 3 of the paper.
4. Compute the distance between the simulated moments and the actual moments from the paper.
5. Use an optimization routine to choose new parameters to minimize the distance between the simulated and actual moments.
6. Repeat steps 2-5 until convergence.

Note that at Step 2, you need to solve the dynamic programming problem of the firm to find the policy functions. These policy functions are functions of the model parameters, so you will need to re-solve the DP problem each time you change the parameters. You can use value function iteration or policy function iteration to solve the DP problem. Once you have the policy functions, you can simulate a panel of firms over time, drawing shocks to productivity as needed. Because you'll need to solve the DP problem many times, you should try to make your code as efficient as possible. TEST this code and make it efficient before writing the estimation routine around it.

This is an involved problem. Plan your code before you write it. Make it modular: e.g., a module to solve the DP problem, a module to simulate the panel of firms, a module to compute the moments, and a module to compute the distance between simulated and actual moments. This will make it easier to debug and test your code.

Optimal Weighting Matrix and Standard Errors

The SMM estimator is given as:

$$\hat{\theta}_{S,T}(W) = argmin_{\theta} \left[\hat{\psi}^d_T - \hat{\psi}^s_{S,T}(\theta)\right]' W_T \left[\hat{\psi}^d_T - \hat{\psi}^s_{S,T}(\theta)\right]$$

The weighting matrix, W_T, can be chosen in different ways. A common choice is to use the identity matrix, which gives equal weight to all moments. A more efficient choice is to use the inverse of the variance-covariance matrix of the moments, which gives more weight to moments that are estimated with less variance.

You don't have the data to compute the variance-covariance matrix of the data moments, but you can estimate it using a two-step approach. Specifically, you first estimate the model parameters using the identity matrix. Next, can simulate many panels of firms using the same parameter estimates (but different draws of the shocks in the model), and compute the moments for each panel, and then compute the variance-covariance matrix of the moments across the simulated panels. You can then use the inverse of this matrix as your weighting matrix and re-estimate the model parameters. This is a two-step efficient SMM estimator. A good description of this approach can be found in these notes

Once you have this weighting matrix, you can compute the standard errors of your parameter estimates using the following:

- Variance covariance matrix for parameter estimates is given by: $Q_S(W) = \left(1 + \frac{1}{S}\right)\left[\frac{\partial b(\theta_0)}{\partial \theta}' W^* \frac{\partial b(\theta_0)}{\partial \theta}\right]^{-1}$
- Where $\frac{\partial b(\theta_0)}{\partial \theta}$ is the derivative of the vector of moments with respect to the parameter vector (so this will be a $q \times p$ matrix for q moments and p parameters).
- Calculate the derivatives numerically - moving θ just a bit and calculating the new vector of moments
- The std errors will be the square roots of the diagonal elements of $Q_S(W)$

Notes

Please follow good research practices:

1. Define functions to make your code modular.
2. Use docstrings to document all functions.
3. When reading/writing data from/to disk, please use **relative paths** and make sure you use Python functions that specify directories in a way that is compatible across operating systems (i.e., your code should work when anyone else runs it from your repository regardless of them using Windows/Mac/Linux).
4. Automate your process: Write a bash script to run your Python (or R) programs and compile your TeX file. A person should be able to clone your repository, run your bash script, and get the same results that you have in your submission.

I am not requiring it, but writing unit tests might be helpful.

Deliverable

You will submit your problem set by pushing the files to your GitHub repository that you created from forking the repository for this class. You will place all files for the problem set in the path `/CompEcon_Fall2025/ProblemSets/ProblemSet8/` on your `ProblemSets` branch. These files will include:

1. Multiple `*.py` files that solve, simulate
2. One `ProblemSet8_LastName.tex` file that contains your problem set summary. The file will read in any images and tables produced with the above (at a minimum this is your replication of Table 3). Also include the complied PDF, but if you do this right, I should be able to recreate it. This file will include a description of the economic model, the moments from the data, and what you found in your estimation.
3. One `*.sh` file that will run your Python script(s), update the table(s) (as well as any figures you want to include), and compile a new pdf of your ProblemSet.

Your overall grade will be determined by your ability to accurately replicate the results in Table 3, the efficiency and organization of your code, and the clarity of your write-up.

Helpful hints

- Firm productivity is stochastic and follows an AR(1) process in logs. You can discretize this process using seveal different approaches, such as Tauchen's method. The `ApproxAR.ipynb` notebook in the course repo provides examples of different methods to approximate AR(1) processes over a discrete grid.
- Create a section of your code with all the parameters (including these determining grid size) so that they can easily be changed.