**LSE Macroeconomics Summer Program**
**Part II: Heterogeneous Agents**
**Instructor: Wouter J. Den Haan**

**Monday Assignment**
**Solving the Aiyagari model & understanding equilibrium effects**

## 1 Objective

The objective of today's exercise is to solve a version of the Aiyagari model using perturbation methods. You will also investigate the effect of uncertainty on aggregate savings and document the insight of Aiyagari that the answer to this type of question can be fundamentally misleading if you restrict yourself to looking at partial equilibrium models.

## 2 The model

In the model, there is no aggregate risk and no aggregate dynamics. Capital is fixed at some (endogenous) level $K$ and labour is normalised to 1. There is a representative firm that pays competitive prices for capital and labour.[1] This means that the rental rate and the wage rate are given by the following equations:

$$r = \alpha K^{\alpha-1}$$

$$w = (1-\alpha)K^{\alpha}$$

With no aggregate dynamics, $w$ and $r$ are constant (they depend only on aggregate $K$). There is, however, idiosyncratic risk. Each individual is subject to a labour productivity shock, $z_{i,t}$, and markets are incomplete, so that the only way to insure against a negative shock is by investing and disinvesting in capital. An individual takes the wage and the capital rental rate as given and solves the following problem:

$$\max_{\{c_{i,t},k_{i,t}\}} \; \mathrm{E}\sum_{t=1}^{\infty}\beta^t\left(\frac{c_{i,t}^{1-\nu}}{1-\nu} - \frac{\zeta_1}{\zeta_0}exp(-\zeta_0 k_{i,t}) - \zeta_2 k_{i,t}\right), \text{s.t.}$$

$$c_{i,t} + k_{i,t} = rk_{i,t-1} + wz_{i,t} + (1-\delta)k_{i,t-1} \tag{1}$$

$$z_{i,t} = (1-\rho) + \rho z_{i,t-1} + e_{i,t} \tag{2}$$

[1]Note that under the assumption of competitive markets and constant returns to scale firm size is not determined and irrelevant. So you can focus on a representative firm without loss of generality.

!!! In this assignment, $z_{i,t}$ is an *idiosyncratic* productivity shock not an *aggregate* productivity shock. In this model, there is no aggregate uncertainty!!!

The penalty function in the utility implements ensures that the problem is well behaved and prevents Ponzi schemes. If $\zeta_0$ goes to infinity then this specification would approach the specification with a non-negativity constraint on $k_{i,t}$. For lower values of $\zeta_0$ this formulation is *different* from the specification with the standard inequality constraint, at least quantitatively. We will focus on low values of $\zeta_0$. There are two reasons for this. First, the higher the value of $\zeta_0$ the more nonlinear the problem and the harder it would be to obtain a solution with perturbation techniques. Second, the inequality constraint is quite extreem and a gradual formulation may very well be more realistic.

The additional term, $\zeta_2 k_{i,t}$, is included mainly for practical reasons in that it allows us to control the steady state value.

The first-order condition for the individual's problem is:

$$c_{i,t}^{-v} = -\zeta_2 + \zeta_1 exp(-\zeta_0 k_{i,t}) + \beta * c_{i,t+1}^{-v}(r + 1 - \delta) \tag{3}$$

## 3 Exercises

In today's exercise, you will solve and analyse the Aiyagari model. Programs that solve the model have already been partially written and you have to complete them. Several files have been provided:

1. For Dynare-based solution:
   - `SolveAiyagari.m` is the main file.
   - `diseq.m` computes for a given interest rate the difference between the amount of capital supplied by households and the demand by firms.
   - `Aiyagari.mod` is the Dynare file with the individual's problem.
2. For the algorithm of the individual problem using projection methods:
   - `proj_PE.m` is the main file.
   - `proj_coef.m` is a function that creates the grid and runs the minimisation routine to find the solution to the individual problem.
   - `projerrs.m` is a function that computes projection errors at the nodes. These are the errors you want to minimise in the minimisation routine.
   - `polyn.m` is a function that creates a polynomial for different orders of projection.
   - `hernodes.m` creates Gauss-Hermite nodes used for a grid.

You have to do the following exercises.

1. Have a quick look at `SolveAiyagari.m` to get a general picture of what the program is supposed to do (e.g., read the comments).

2. Open `diseq.m`. You have to program the recursion that generates a long series for capital, using the policy function given by Dynare. This long series of capital will be used to compute the average capital supply.

3. The setting of parameters and the model block of `Aiyagari.mod` have been left out. Start the exercise by completing the file by filling in the model equations (i.e., equations 1 to 3) and by completing the parameter block. Most parameter values are set in the main program. These together with the value of the interest rate are written to an external file in the file `diseq.m`. The Dynare file reads these.

4. In `SolveAiyagari.m`, examine the routine that solves for the equilibrium $r$. At the end of each iteration you have to update the values of $r\_x$ and $diseq\_r\_x$.

5. Now run the `SolveAiyagari.m`. The program solves the Aiyagari model twice, first for a low value of shock volatility ($\sigma = 0.001$) and then for a higher value ($\sigma = 0.3$).
   In *ks_ge* you can find the capital supply for these two values of $\sigma$. Think of what an increase in idiosyncratic risk should do to the aggregate capital stock. Do the values in *ks_ge* correspond to your intuition?

6. Now make part 4 of the program operational by uncommenting it. This part solves again for two values of $\sigma$, but this time for a fixed interest rate (the equilibrium interest rate for the case of $\sigma = 0.001$).

   (a) First define the shocks, save the parameter values, run dynare and load the policy functions.

   (b) Next, fill in the recursion for simulating the economy (same as you did before in `diseq.m` - you could actually copy it from there).

   (c) The results are displayed on screen. The first row displays the partial equilibrium results (for a fixed interest rate). The first number printed on screen is the level of capital in the partial equilibrium at the lower $\sigma$ and the second number is the percentage increase in the level of capital after volatility increases. The second row shows the effects in the general equilibrium, i.e., when the interest rate is allowed to adjust to equilibrate the demand and supply of capital. The first number is again the level of capital and the second is the percentage increase in capital when volatility increases.

7. Try to understand why the effects of volatility are larger in partial than in general equilibrium. What causes the dampening? (Hint: think what happens to the interest rate.)

8. Redo the exercise for some different parameter settings.

(a) Initially, the shock persistence, $\rho$, was set to zero. Now consider values of $\rho = 0.1$ and $\rho = 0.3$.

(b) Reset shock persistence back to zero. Now examine what happens if intertemporal risk aversion increases. Compare the results when $\nu = 0.5$ and (for example) when $\nu = 5$. Are the results such as you expected? You could also try a low value for $\nu = 0$.

## 4 Using Dynare Policy Rule Coefficients

Dynare stores the coefficients of the policy rules inside the oo_ structure. Unfortunately, what is stored is not exactly equal to the more natural representation that Dynare displays on the screen; for example, the ordering may be different and for second-order some coefficents are multiplied by 1/2). To help you out, we have provided a program "get_policy_rule_coefs.m" that will put the policy rule coefficients in a matrix "decision" in exactly the order that Dynare writes them to the screen. In principle, there is nothing you have to do in the program, but take a look at what this program does anyway.

**Some tips on get_policy_rule_coefs.m**

- As you can see, you have to set a couple variables before running the program (like the perturbation order).
- This program allows YOU to choose which variables to put into the Matlab matrix decision and in which order.
- So if you list k as the first variable, then its policy rule coefficients will appear in the first column of decision. This may or may not be different from where Dynare puts it.
- Since there are not that many variables in this program, you may want to use the same ordering as Dynare has chosen. But it is up to you. Since you only need capital, you may also choose to only list k, so that the matrix decision will be just a column vector.
- Alternatively, you can use the function provided to generate the matrix decision. This does not change anything regarding the use of global memory.

**!!!! Important comment about the global statement** The variables created by Dynare like oo_ are only available in the main program. This means that they will not be available inside a function EVEN if Dynare is run inside that function. This is why you need to add global M_ oo_ at the beginning of both the main program and the function in which you run Dynare. If you run Dynare in the main program, then you would not need this global statement.

## 5 Additional exercise

As an additional exercise you can solve the partial equilibrium with projection methods. Examine the files `proj_PE.m, proj_coef.m, projerrs.m` and `polyn.m`, so that you understand the general idea of the projection method. Note that the state variable is wealth, which is the sum of capital, capital income, and labor income.

1. Start by filling in the `proj_coef.m` file. This file creates the grid and calls the minimization routine to minimize the errors between the right and left hand side of the Euler equation.

   - Specify the steady state value for wealth.

2. Fill in the `projerrs.m` file. This function calculates the errors between the right and left hand side of the Euler equation which you want to minimize. Fill in the matrix *rhs*, which gives you the right hand side of the Euler equation for each grid point for wealth (rows) and for each Gauss-Hermite node of the shock process (columns).

   - Specify a value for the capital choice (*knew*) at each point of the grid for wealth. Use the budget constraint for this.
   - Specify a value for next periods wealth (*anew*) at each grid point. For this you need *knew* and you need to replace the innovation with the Gauss-Hermite node and the scaling factor.
   - Specify the value of consumption in the next period (*cnew*) at each grid point. That is, just evaluate the approximating polynomial at the new values for wealth, given you coefficients and polynomial order (hint: use the `polyn.m` function as is done for consumption in period t).
   - Now you can write the entire right hand side for each point of the grid and a given Gauss-Hermite node. Use the above created objects to do that. Dont forget the Gauss-Hermite weights and dont forget to divide by $\sqrt{\pi}$!

3. Now that you got the projection running, go back to `proj_PE.m.` Fill in the recursion that simulates the economy, given the policy function for consumption you calculated with you projection method.

   - First set the initial value of wealth to its steady state.
   - Remember that the state variable is wealth and not capital. Thus, in order to get a recursion for capital, you need one for wealth.

The results are again displayed on the screen. Once again, the first row are the level and the percentage increase of capital when $\sigma$ rises. The second row shows the same for the general equilibrium case which was solved for by perturbation methods.