

1 Theory

1.1 Equilibrium Model

This describes the model we would like to solve. The next sections describes another model which allows us to compute equilibria that are approximately equilibria in this model.

This model is complicated by the heterogeneity in employment state e and assets a . The value function V and the investment and labor policies g and h are computationally intractable because they depend on their entire demographic CDF function, $x(e, a)$.

$$V(z, x, e, a) = \max_{n, a'} u(c, n) + \beta \sum_{z', e'} \Pi_{zz'} \Gamma_{e'|zez'} V(z', x', e', a')$$

where $x(e, a)$ is a measure, $K = \int adx(e, a)$, $N = \int eh(z, x, e, a)dx(e, a)$, $c = (r + 1 - \delta)a + enw - a'$, and

$$x'(e', a') = \sum_{\bar{e} \in E} \Gamma_{e'|z\bar{e}z'} \int_{\{(e, a): e=\bar{e}, g(e, a) \leq a'\}} dx(e, a).$$

We are putting the utility function as

$$u(c, n) = \alpha \log c + (1 - \alpha) \log(1 - n)$$

and the production function as

$$f(z, K, N) = e^z K^\theta N^{1-\theta},$$

which implicitly define prices as $r = f_K(z, K, N)$ and $w = f_N(z, K, N)$.

1.2 Quasi-equilibrium Model

This model substitutes a small number of statistics – aggregate capital and aggregate labor – for the demographic CDF, x . The value function in this model approximates the value function from the previous model with

$$\Omega(z, K(x), N(x), e, a) \approx V(z, x, e, a).$$

Note that this approximation is constructed by a Monte Carlo simulation, and is therefore only likely to be a good approximation near the states visited by the simulation.

$$\Omega(z, K, N, e, a) = \max_{n, a'} u(c, n) + \beta \sum_{z', e'} \Pi_{zz'} \Gamma_{e'|zez'} \Omega(z', K', N', e', a')$$

where $K' = G(z, K)$, $N' = H(z', K')$ and $c = (r + 1 - \delta)a + enw - a'$, and u and f are the same as before.

The aggregate expectation rules G and H are constructed from a simulation so that for “most” demographics x ,

$$G(z, K(x)) \approx \int g(z, K(x), N(x), e, a) dx(e, a)$$

and

$$H(z, K(x)) \approx \int eh(z, K(x), N(x), e, a) dx(e, a).$$

2 Algorithms

Notation: parameters that are in square brackets are initial guesses. Similarly, outputs that are in square brackets are only useful as subsequent guesses.

Algorithm: Solve Heterogeneous Economy

input: nothing. output: $G^*(z, K, N), H^*(z, K, N)$.

- Compute steady state variables, $\bar{C}, \bar{K}, \bar{N}, \bar{x}$.
- Set

$$\begin{aligned} G^0(z, K, N) &= \exp[\bar{K} + \epsilon(K - \bar{K} + N - \bar{N})] \\ H^0(z, K, N) &= \exp[\bar{N} + \epsilon(K - \bar{K} + N - \bar{N})] \\ \Omega^0(z, K, N, e, a) &= \frac{1}{1 - \beta} u(\bar{C}, \bar{N}). \end{aligned}$$

- Iterate on $i = 1, 2, \dots$, until G^i and H^i converge:
 - Use *Algorithm Compute Decision Rules* to compute $(g_*^i, h_*^i, [\Omega_*^i])$ from $(G^{i-1}, H^{i-1}, [\Omega_*^{i-1}])$.
 - Use *Algorithm Simulation* to generate a sequence of z_t^i, K_t^i, N_t^i .
 - Use *Algorithm Capital Approximation* to approximate the new aggregate laws of motion G^i and H^i from the simulated data.

Algorithm: Compute Decision Rules

input: $G^{i-1}(z, K, N), [\Omega_*^{i-1}(z, K, N, e, a)]$

output: $g_*^i(z, K, N, e, a), h_*^i(z, K, N, e, a), [\Omega_*^i]$.

- Set the initial guess, Ω_0^i , to be the converged solution from the previous iteration, Ω_*^{i-1} .
- Do value function iteration until $\Omega_j^i \rightarrow \Omega_*^i$.
- Compute the policy rules g_*^i, h_*^i from the value function Ω_*^i .

Algorithm: Simulation

input: $g_*^i(z, K, N, e, a), h_*^i(z, K, N, e, a)$

output: z_t^i, K_t^i, N_t^i for $t \in \{0, \dots, 1000\}$, where 1000 is the number of time periods in the simulation.

- Pick initial conditions (with aggregate stats near SS) for demographics and aggregate shocks, (x_0^i, z_0^i) .
- Set $K_0^i = \int g_0^i(e, a) dx_0^i(e, a)$.
- Generate z_t^i randomly (distributed according to the exogenously given Markov transition matrix Π).
- For each time period t :
 - Solve $N_t^i = \int e h^i(z_t, K_t^i, N_t^i, e, a) dx_t^i(e, a)$ for aggregate labor N_t^i .
 - Compute time $(t + 1)$'s aggregate capital with $K_{t+1}^i = \int g^i(z_t, K_t^i, N_t^i, e, a) dx_t^i(e, a)$.
 - Then compute x_{t+1}^i from $(g^i, h^i, K_t^i, N_t^i, x_t^i)$.
 - At the end of this, we have computed $(x_{t+1}^i, K_{t+1}^i, N_t^i)$.
- Output z_t^i, K_t^i, N_t^i .

Algorithm: Law of Motion Approximation

input: z_t^i, K_t^i, N_t^i for $t \in \{0, \dots, 1000\}$

output: log-linear coefficients for $G^i(z, K, N)$.

- Throw out the first 100 observations (z_t^i, K_t^i, N_t^i) for $t \in \{0, \dots, 100\}$, so that initial conditions don't bias things.
- Run a linear regression of

$$\log \frac{K_{t+1}^i}{\bar{K}} = \beta_0 + \beta_1 \log \frac{K_t^i}{\bar{K}} + \beta_2 \log \frac{N_t^i}{\bar{N}} + \epsilon_t,$$

where (\bar{K}, \bar{N}) are steady state values.

- Set the approximation to be

$$G^i(z, K, N) = \exp[\beta_0 + \beta_1 \log \frac{K}{\bar{K}} + \beta_2 \log \frac{N}{\bar{N}}].$$

Do a similar thing for $H^i(z, K, N)$.

3 Implementation

This section describes the contents of the files in our implementation. Files ending in *.f90* are written in Fortran 95, and files ending in *.r* are written in GNU R.

- *ks-innerloop.f90* implements the portions of the value function iteration and interpolation that need to be fast. In particular, it implements
 - *interpolate_state_function()* evaluates functions such as the policy rules g, h and the value function Ω . It uses Schumacher splines to interpolate.
 - *iterate_omega()* is the key portion of *Algorithm: Compute Decision Rules* that computes Ω_{j+1}^i from Ω_j^i . This is the inner loop of the value function iteration algorithm.
- *ks.r* implements the outer loops and computes initial conditions (by computing steady states, etc). Important functions include:
 - *compute_policy()* is the outer loop of the value function iteration *Algorithm: Compute Decision Rules*. It computes Ω_*^i and associated policy rules g_*^i and h_*^i by repeatedly calling *iterate_omega()*.
 - *simulate()* implements *Algorithm: Simulation*.
 - *regress_motion()* implements *Algorithm: Law of Motion Approximation*.
- *sspline.f90* is an implementation of Schumacher splines.
- *ks-innerloop.r* and *sspline.r* are wrappers for their similarly named Fortran counterparts, that allow R code to access the Fortran implementations.
- *bisect.r* contains a bisection implementation with lots of debugging features, such as graph drawing.
- *presentation.r* is some scrappy code for drawing graphs. Ignore it.