Week 3: Stationary Equilibrium of HA Model

Computation Study Group

Peking University, HSBC Business School

Current slides are mainly based on Prof.Jinhui Bai's lecture notes.
Special thanks to Prof.Jinhui Bai!

June 30, 2021

Aiyagari (1994) Model

- A household saving problem

$$V(k,\epsilon) = \max_{c,a'} \left\{ \frac{c^{1-\sigma}}{1-\sigma} + \beta \, \mathrm{EV}\left(k',\epsilon'\right) \right\}$$

subject to

$$c + k' = (1 + r - \delta)k + w\epsilon\bar{l}$$
$$c \geq 0, k' \geq -\phi$$

ϵ is idiosyncratic labor productivity shock.

- Firm's problem

$$\max = K^{\alpha} N^{1-\alpha} - rK - wN$$

For now, no aggregate TFP shock.

Stationary Recursive Competitive Equilibrium

A stationary recursive competitive equilibrium is a set of functions, $v(k,\epsilon)$ and $g(k,\epsilon)$, a set of prices and quantities (r,w,K,N), and a stationary distribution $\lambda(k,\epsilon)$ such that

- Given (r,w), $v(k,\epsilon)$ and $g(k,\epsilon)$ solve the household's dynamic programming problem.
- Prices are competitively determined:

$$w = (1-\alpha)\left(\frac{K}{N}\right)^{\alpha}, \quad r = \alpha\left(\frac{K}{N}\right)^{\alpha-1} - \delta$$

- Market clears:

$$K = \sum_{\epsilon}\sum_{k}\lambda(k,\epsilon)g(k,\epsilon), \quad N = \sum_{\epsilon}\sum_{k}\lambda(k,\epsilon)\epsilon\bar{l}$$

- $\lambda(k,\epsilon)$ is a stationary distribution from $g(k,\epsilon)$.

Some Math Preparation

Some Key Elements in Numerical Computation

- Discretization
- Function Approximation
- Optimization
- Root Finding / Equation Solving

Function Approximation

How to approximate a continuous function from discrete function values?

- We can use piece-wise polynomial approximation
- Idea: Construct a low-order polynomial for every two neighboring grid points.
- We introduce two methods
  - Cubic Spline
  - Piecewise Cubic Hermite Interpolation Polynomial (PCHIP)

Function Approximation

Cubic Spline: MATLAB function "spline"

- A Cubic Spline is a set of piecewise cubic polynomials $\widehat{f}(x)$ for each $n = 1, 2, \ldots, N-1$ and $x \in [x_n, x_{n+1}]$

$$\widehat{f}_n(x) = c_{n0} + c_{n1}(x - x_n) + c_{n2}(x - x_n)^2 + c_{n3}(x - x_n)^3$$

such that

  - Function value is continuous for all nodes: $\widehat{f}_n(x_n) = y_n$ and $\widehat{f}_{n+1}(x_{n+1}) = y_{n+1}$ for all $n = 1, 2, \ldots, N-1$
  - First-order derivative is continuous for each interior node: $\widehat{f}'_n(x_n) = \widehat{f}'_{n+1}(x_n)$ for $2 \leq n \leq N-1$
  - Second-order derivative is continuous for each interior node: $\widehat{f}''_n(x_n) = \widehat{f}''_{n+1}(x_n)$ for $2 \leq n \leq N-1$

- How can we pin down the coefficients?
  - We have $4(N-1)$ unknown parameters, but only $2(N-1) + 2(N-2) = 4N - 6$ restrictions.
  - Need two more conditions, for example
    - "not-a-knot": $\widehat{f}'''_1(x_2) = \widehat{f}'''_2(x_2)$, $\widehat{f}'''_{N-2}(x_{N-1}) = \widehat{f}'''_{N-1}(x_{N-1})$
    - Requirements on $\widehat{f}'_1(x_1)$ and $\widehat{f}'_{N-1}(x_N)$.

Function Approximation

Piecewise Cubic Hermite Interpolation Polynomial: MATLAB function "pchip"

- Suppose on each node, we have data on both function value and first derivative value: $(x_n, y_n, y'_n)_{n=1}^N$, where

$$y_n = f(x_n)$$
$$y'_n = f'(x_n)$$

- Then on each interval $[x_n, x_{n+1}]$, the data uniquely determines a cubic polynomial

$$\widehat{f}_n(x) = c_{n0} + c_{n1}(x - x_n) + c_{n2}(x - x_n)^2 + c_{n3}(x - x_n)^3$$

for $x \in [x_n, x_{n+1}]$ through four conditions:

$$y_n = \widehat{f}_n(x_n), y_{n+1} = \widehat{f}_{n+1}(x_{n+1}),$$
$$y'_n = f\widehat{f}'_n(x_n), y'_{n+1} = \widehat{f}'_n(x_{n+1}).$$

- In reality, we usually don't have data on derivatives. MATLAB function "pchip" approximate it by average of two slopes.

Function Approximation

Comparison between Interpolation Methods

- Cubic spline is more smooth. We can easily calculate first and second order derivatives from it.
- PCHIP is more shape-preserving. It can better preserve the shape of a kinked line (for example, the policy function in the Aiyagari model).

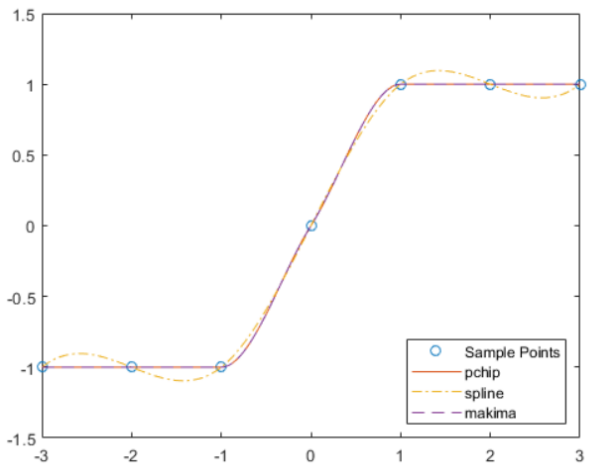


Figure: Comparison between Interpolation Methods

Root Finding

How to find root(s) for a non-linear equation $f(x) = 0$?

- Bracketing Method
  - Step 1: Find an interval (bracket) (a, b) such that $f(a)f(b) < 0$.
  - Step 2: Find a point x inside the bracket. If $f(a)f(x) > 0$, let $a = x$; if $f(b)f(x) > 0$, let $b = x$
  - Step 3: Redo Step 2 on new (a, b)
  - Step 4: Break when $|b - a|$ is sufficiently small. Then x is the root we find.

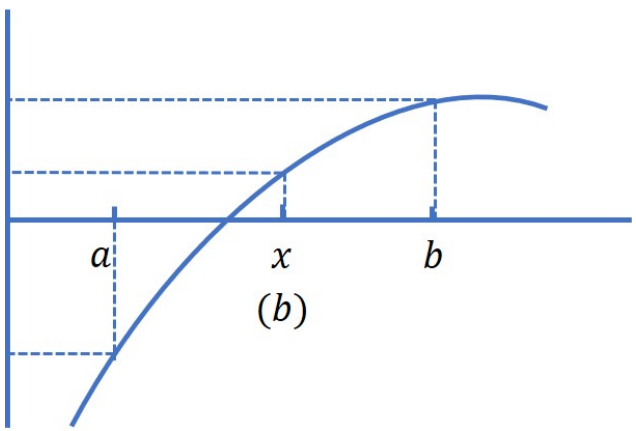


Figure: Bracketing Method

Root Finding

Now the question is: how to find such a x inside the bracket (a, b)?

- A naive way: bisection.
- More efficient way: by linear approximation.
  In Step k, approximate $f(x)$ around last Step's x_{k-1}:

$$f(x) \approx f(x_{k-1}) + A_k(x - x_{k-1})$$

$$f(x_{k-1}) + A_k(x - x_{k-1}) = 0 \Rightarrow x_k = x_{k-1} - A_k^{-1} f(x_{k-1})$$

- How to choose A_k?
  - Fixed point iteration: A_k =1.
  - Newton's method: $A_k = f'(x_{k-1})$.

Root Finding

- Fixed point iteration

$$x_k = x_{k-1} - f(x_{k-1})$$

- Newton's method

$$x_k = x_{k-1} - f(x_{k-1})^{-1} f(x_{k-1})$$

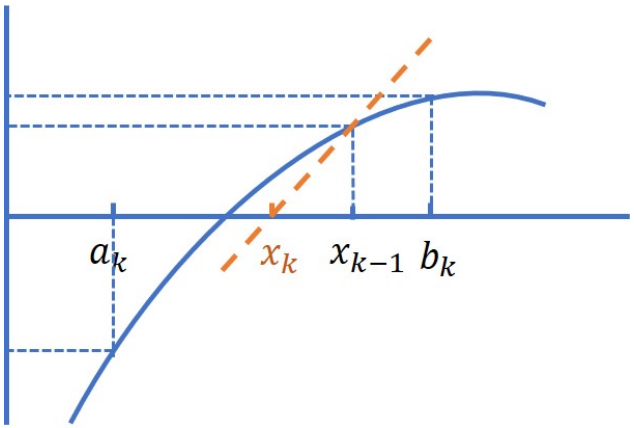


Figure: Fixed Point Iteration

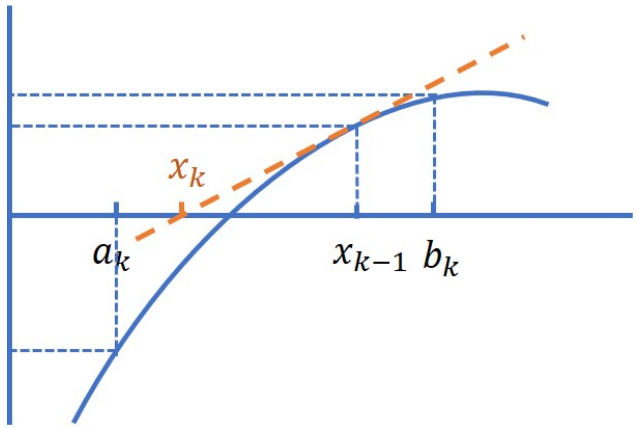


Figure: Newton's Method

Root Finding

MATLAB built-in functions for equation solving

- fzero: solves one-dimensional non-linear equation
- fsolve: solves multi-dimensional non-linear equations
- Note: The idea of N-D non-linear equation solving is different from 1-D case: it actually tries to solve the global minimum of a quadratic function and uses function optimization. Hence, directly uses optimization algorithm if you can.
- Recommend you to read MATLAB documentation.

Optimization

How to find local minimum for a function $f(x)$?
Idea:

- We still use Bracketing Method: shrink bracket $[a, b]$ until we find a local minimum.
- A simple way: Bisection section search.
- More efficient way: by quadratic approximation.

$$\widehat{f}(x) = c_0 + c_1 x + c_2 x^2$$

If $c_2 > 0$, a candidate iteration point is given by the minimizer

$$\arg\min \widehat{f}(x) = -\frac{c_1}{2c_2}$$

If $c_2 < 0$ or $\arg\min \widehat{f}(x) \notin [a, b]$, update by safe methods like bisection search.

Optimization

How to solve coefficient c_0, c_1 and c_2 in $\widehat{f}(x)$?

- Brent's Method: Use three function values.

  MATLAB function: fminbnd

- Quasi-Newton Method: Use one function value and two first derivatives.

$$\widehat{f}(x) = f\left(x^{(k)}\right) + f'\left(x^{(k)}\right)\left(x - x^{(k)}\right) + \frac{1}{2}A^{(k)}\left(x - x^{(k)}\right)^2$$

  where

$$A^{(k)} = \frac{f'\left(x^{(k)}\right) - f'\left(x^{(k-1)}\right)}{x^{(k)} - x^{(k-1)}}$$

  MATLAB function: fmincon

- Newton Method: Use one function value, one first derivative and one second derivative.

$$\widehat{f}(x) = f\left(x^{(k)}\right) + f'\left(x^{(k)}\right)\left(x - x^{(k)}\right) + \frac{1}{2}f''\left(x^{(k)}\right)\left(x - x^{(k)}\right)^2$$

  MATLAB function: fmincon

Individual household's dynamic programming problem

Solution Methods: An Overview

- Bellman Equation Methods
  - Value function iteration
  - Value function iteration with Howard improvement
- Euler Equation Methods
  - Euler equation iteration / Policy function iteration
  - Euler equation perturbation method (Dynare)

Bellman Equation Methods

- Bellman Equation:

$$V(k,\epsilon) = \max_{k'} \left\{ \frac{((1+r-\delta)k + w\epsilon - k')^{1-\sigma}}{1-\sigma} + \beta EV\left(k', \epsilon'\right) \right\}$$

subject to

$$-\phi \leq k' \leq (1+r-\delta)k + w\epsilon$$

- Our goal:
Solve value function $V(k,\epsilon)$ and policy function $k' = G(k,\epsilon)$.

Discretization of State Variables

We discretize the domain of functions $V(k, s)$ and $G(a, s)$. That is, we discretize state variables k and s.

- Discretization of k : k and k' lies on a N by 1 grid with

$$n \in \mathcal{N} = \{1, 2, \ldots, N\}$$
$$k \in \mathcal{K} = \{k_1, k_2, \ldots, k_N\}$$

- Discretization of ϵ: ϵ follows S-state Markov Chain with state space

$$s \in \mathcal{S} = \{1, 2, \ldots, S\}$$
$$\epsilon \in \mathcal{E} = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_S\}$$

and a S by S Transition Probability Matrix $\mathcal{P}$

$$\mathcal{P}\left(s, s'\right) = \Pr\left(\epsilon_{t+1} = \epsilon_{s'} \mid \epsilon_t = \epsilon_s\right)$$

Note: methods to discretize an AR(1) process into $\mathcal{P}$
(1) Rouwenhorst (1995); (2) Tauchen (1991).

Discretization of State Variables

Now the Bellman Equation becomes

$$V(k_n, \epsilon_s) = \max_{k'} \left\{ \frac{\left((1 + r - \delta)k_n + w\epsilon_s - k'\right)^{1-\sigma}}{1 - \sigma} + \beta \sum_{s'=1}^{S} \mathcal{P}(s, s') V\left(k', \epsilon_{s'}\right) \right\}$$

subject to

$$-\phi \leq k' \leq (1 + r - \delta)k_n + w\epsilon_s$$

$$k' \in \mathcal{K} = \{k_1, k_2, \ldots, k_N\}$$

- Our goal:
  Solve value function $V(k_n, \epsilon_s)$ and policy function $k' = G(k_n, \epsilon_s)$ for

$$n \in \mathcal{N} = \{1, 2, \ldots, N\}$$

$$s \in \mathcal{S} = \{1, 2, \ldots, S\}$$

Value Function Iteration: Idea

- We are essentially solving a root finding problem:

$$V = TV$$

$$f(V) = V - TV = 0$$

- We can solve it by fixed point iteration
  - Step 0: Choose an initial value function V.
  - Step 1: Obtain new value function V' by

$$V' = V - f(V) = V - (V - TV) = TV$$

  - Step 2: Check if $||V' - V|| < t_v$, where t_v is a predetermined tolerance level. If not, let $V = V'$, and redo Step 1-2. Break if $||V' - V|| < t_v$ or number of iteration $> MaxIter_v$
- It is value function iteration.

Value Function Iteration

Then how can we perform this iteration?
Here we use a continuous-state method, by using function interpolation.

- Begin with old value function
  $V(k_n, \epsilon_s)$ $(n \in \mathcal{N} = \{1, 2, \ldots, N\},\ s \in \mathcal{S} = \{1, 2, \ldots, S\})$
- Our goal: obtain new value function on each grid point (k_n, ϵ_s).
- Interpolation
  - Purpose:
    $V(k_n, \epsilon_s),\ n \in \mathcal{N} = \{1, 2, \ldots, N\}, s \in \mathcal{S} = \{1, 2, \ldots, S\} \rightarrow$
    $V(k, \epsilon_s),\ k \in [k_1, k_N], s \in \mathcal{S} = \{1, 2, \ldots, S\}$
  - Cubic spline: vfn = spline(agrid,v.');
  - Evaluation: ppval(vfn,aprime);

Value Function Iteration

- Maximization

$$V(k_n, \epsilon_s) = \max_{k'} \left\{ \frac{((1 + r - \delta)k_n + w\epsilon_s - k')^{1-\sigma}}{1 - \sigma} + \beta \sum_{s'=1}^{S} \mathcal{P}(s, s') V\left(k', \epsilon_{s'}\right) \right\}$$

subject to

$$k' \in [\max\{-\phi, k_1\}, \min\{(1 + r - \delta)k_n + w\epsilon_s, k_N\}]$$

  - Constrained Optimization
    MATLAB built-in functions: fminbnd, fmincon.
  - We obtain
    (1) new value function $V'(k_n, \epsilon_s)$
    (2) policy function $k' = g(k_n, \epsilon_s)$

Howard Improvement: Idea

- In value function iteration, we have a byproduct: policy function $k' = g(k_n, \epsilon_s)$.
- But in previous value function iteration, we completely ignore the information in g.
- Now, how about utilizing the information in g ?
  An idea: if $g(k_n, \epsilon_s)$ is the true policy function, then we have

$$V(k_n, \epsilon_s) = \frac{\left((1 + r - \delta)k_n + w\epsilon_s - k'\right)^{1-\sigma}}{1-\sigma} + \beta \sum_{s'=1}^{S} \mathcal{P}(s, s') V\left(k', \epsilon_{s'}\right)$$

  Then, given
  $k' = g(k_n, \epsilon_s), (n \in \mathcal{N} = \{1, 2, \ldots, N\},\ s \in \mathcal{S} = \{1, 2, \ldots, S\})$,
  we can solve for $V(k_n, \epsilon_s)$.

Howard Improvement

But how can we solve for $V(k_n, \epsilon_s)$?
Again, it is an equation solving problem – we can use fixed point iteration!

- Step 0: Choose an initial value function $V(k_n, \epsilon_s)$.
- Step 1: Obtain a new value function $V'(k_n, \epsilon_s)$ by

$$V'(k_n, \epsilon_s) = \frac{((1 + r - \delta)k_n + w\epsilon_s - k')^{1-\sigma}}{1 - \sigma} + \beta \sum_{s'=1}^{S} \mathcal{P}(s, s') V(k', \epsilon_{s'})$$

- Step 2: Check if $||V' - V|| < t_h$. If not, let $V = V'$ and redo Step 1-2.
  Break if $||V' - V|| < t_h$ or number of iteration $> MaxIter_h$

Value Function Iteration + Howard Improvement

Now we combine VFI and Howard Improvement.

- Step 0: Initialization
  (1) Set initial value function $V^0(k_n, \epsilon_s)$ and policy function $G^0(k_n, \epsilon_s)$.

  (2) Set tolerance level for value function, policy function and Howard Improvement step: t_v, t_p, t_{hp}, and t_h.

  (3) Set maximum iteration number K_v, and J_h.

- Step 1: Value function iteration.

  In iteration $k = 1, ..., K_v$, use continuous state VFI to calculate value function $\hat{V}^k$ and policy function G^k.

- Step 2: Check. If $||V^k - V^{k-1}|| < t_v$ and $||G^k - G^{k-1}|| < t_p$, declare success with the solution $V = V^{k-1}$ and $G = G^k$. Otherwise, go to Step 3.

Value Function Iteration + Howard Improvement

- Step 3: Update.

  If $||G^k - G^{k-1}|| < t_{hp}$, then update V^k by Howard Improvement.

  - Step 3.0 Let $\hat{V}^k$ be initial value in Howard Improvement: $V_h^0 = \hat{V}^k$.
  - Step 3.1 For iteration $j = 1, ..., J_h$,

$$V_h^j(k_n, \epsilon_s) = \frac{\left((1 + r - \delta)k_n + w\epsilon_s - G^k(k_n, \epsilon_s)\right)^{1-\sigma}}{1 - \sigma}$$
$$+\beta \sum_{s'=1}^{S} \mathcal{P}(s, s') V_h^{j-1}\left(G^k(k_n, \epsilon_s), \epsilon_{s'}\right)$$

  - Step 3.2 Check: if $||V^j - V^{j-1}|| < t_v$, break; otherwise, back to Step 3.1.

  Update V^k by $V^k = V_h$, which is obtained in the Howard Improvement process.

  If $||G^k - G^{k-1}|| \geq t_{hp}$, then update V^k by original VFI value: $V^k = \hat{V}^k$.

Euler Equation Methods

Euler equation in the Aiyagari Model (suppose interior solution)

$$c^{-\sigma} = \beta(1+r)Ec'^{-\sigma}$$

$$((1+r-\delta)k + w\epsilon - k')^{-\sigma} = \beta(1+r)E((1+r-\delta)k' + w\epsilon' - k'')^{-\sigma}$$

$$((1+r-\delta)k + w\epsilon - g(k,\epsilon))^{-\sigma} = \beta(1+r)E((1+r-\delta)g(k,\epsilon) + w\epsilon' - g(k',\epsilon'))^{-\sigma}$$

where g is the policy function: $k' = g(k,\epsilon)$ and $k'' = g(k',\epsilon')$.

- Euler equation gives a functional equation of policy function g: again, an equation-solving problem.
- Again, we can use fixed point iteration.
  Given a policy function $\tilde{g}$, we can solve for a new policy function g by solving the root of equation

  $$((1+r-\delta)k+w\epsilon-g(k,\epsilon))^{-\sigma} = \beta(1+r)E((1+r-\delta)g(k,\epsilon)+w\epsilon'-\tilde{g}(k',\epsilon'))^{-\sigma}$$

  Iterate until $||\tilde{g} - g|| < t_p$.

Policy Function Iteration

Recall that in practice, policy function is on discrete grids:

$$k' = g(k_n, \epsilon_s), (n \in \mathcal{N} = \{1, 2, \ldots, N\},\ s \in \mathcal{S} = \{1, 2, \ldots, S\})$$

Then there are two types on policy function iteration methods:

- Exogenous Grid Method
- Endogenous Grid Method

Policy Function Iteration: Exogenous Grid Method

Euler Equation:

$$((1+r-\delta)k_n+w\epsilon_s-k')^{-\sigma} = \beta(1+r)E((1+r-\delta)k'+w\epsilon_{s'}-g(k',\epsilon_{s'}))^{-\sigma}$$

- Step 0: Choose an initial policy function $g(k_n, \epsilon_{s'})$ $(n \in \mathcal{N} = \{1, 2, \ldots, N\},\ s \in \mathcal{S} = \{1, 2, \ldots, S\})$.
- Step 1: Use interpolation to approximate continuous policy functions $\tilde{g}(k', \epsilon_{s'})$, $s \in \mathcal{S} = \{1, 2, \ldots, S\}$.
- Step 2: For each n and s, solve new policy function $k' = g'(k_n, \epsilon_s)$ from Euler Equation.
- Iterate until the convergent of policy function g on grid points.

Step 2 is time-consuming, since it involves solving a non-linear equation.

Policy Function Iteration: Endogenous Grid Method

Euler Equation:

$$((1+r-\delta)k_n+w\epsilon_s-k')^{-\sigma}=\beta(1+r)E((1+r-\delta)k'+w\epsilon_{s'}-g(k',\epsilon_{s'}))^{-\sigma}$$

- Step 0: Choose an initial policy function $g(k_n,\epsilon_{s'})$.
- Step 1: Endogenous Grid.
  For each today's $\epsilon=\epsilon_s$ and each future $k'=k_{n'}$ and $k''=g\left(k_{n'},\epsilon_{s'}\right)$, solve today's k from Euler Equation:

  $$\widehat{k}_{n's}=\frac{RHS^{-\frac{1}{\sigma}}+k_{n'}-w\epsilon_s}{1+r-\delta},\ RHS=\beta(1+r)E((1+r-\delta)k_{n'}+w\epsilon_{s'}-g(k_{n'},\epsilon_{s'}))^{-\sigma}$$

- Step 2: Function Approximation and Interpolation
  For each today's $\epsilon=\epsilon_s$, now we have $\left(\widehat{k}_{n's},k_{n'}\right)_{n'=1}^{N}$. Use interpolation to obtain a continuous policy function $\tilde{g}\left(k,\epsilon_s\right)$. Evaluate $\tilde{g}$ at exogenous grid point $\{k_1,k_2,\ldots,k_N\}$ to get new policy function $g'(k_n,\epsilon_s)$.
- Iterate until the convergent of policy function g on grid points.

Endogenous Grid Method: Corner Solutions

Considering the possibility of corner solutions, Euler Equation becomes

$$((1+r-\delta)k_n+w\epsilon_s-k')^{-\sigma} \geq \beta(1+r)E((1+r-\delta)k'+w\epsilon_{s'}-g(k',\epsilon_{s'}))^{-\sigma}$$

"$>$" implies $k'=0$ while $k'>0$ implies "$=$".
How to deal with it?
– Add & Drop

Endogenous Grid Method: Corner Solutions

If $\widehat{k}_{n's} < \phi$, discard $\left(\widehat{k}_{n's}, k_{n'}\right)$ pair. For $k' = k_1 = \phi$, add all grid point pair (k_n, ϕ) to endogenous grids, where $k_n \leq \widehat{k}_{1s}$.

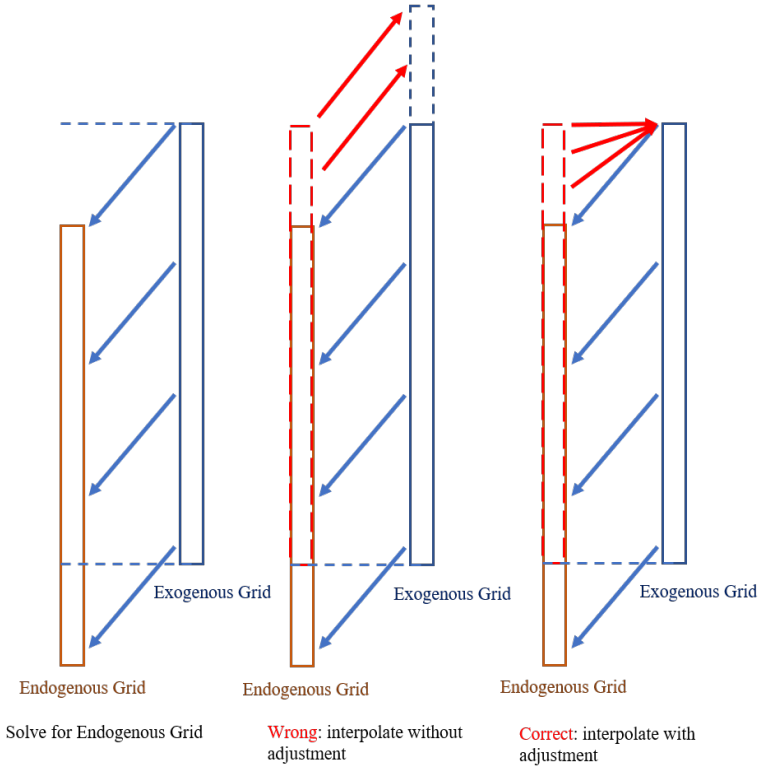


Figure: "Add" Case

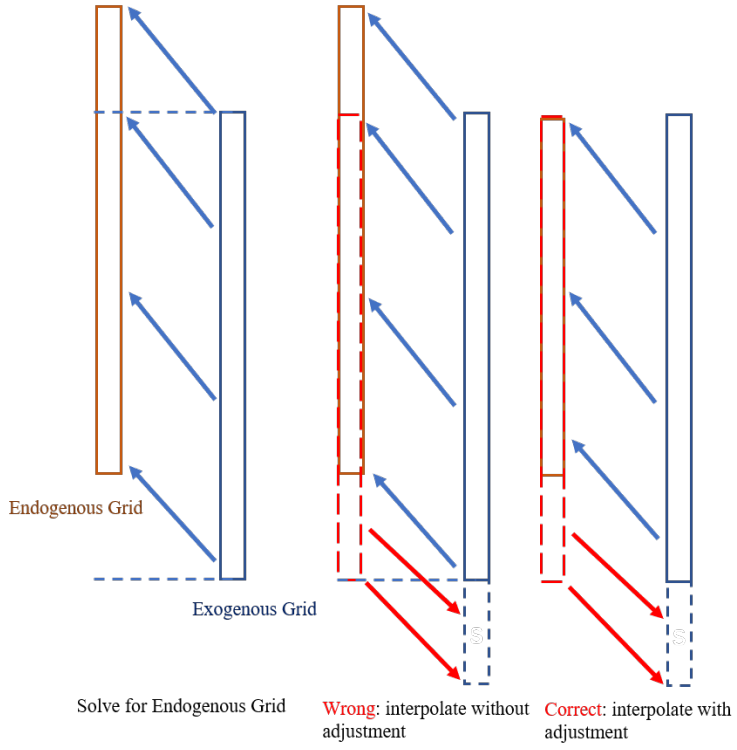
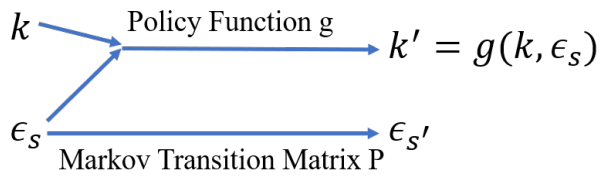


Figure: "Drop" Case

Stationary Distribution

Evolution of Probability Distribution

- Evolution of an individual's state (k, ϵ_s)



- With a continuous k, (k, ϵ_s) follows a continuous-state Markov process with transition prob density function given by

$$Q\left(\left(k, \epsilon_s\right), \left(k', \epsilon_{s'}\right)\right) = \mathcal{P}\left(s, s'\right) \cdot \mathcal{I}\left(k' = g\left(k, \epsilon_s\right)\right)$$

- Evolution of the distribution
  The distribution over (k, ϵ_s), $\lambda(k, \epsilon_s)$, evolves according to

$$\lambda_{t+1}\left(k', \epsilon_{s'}\right) = \sum_s \int Q\left(\left(k, \epsilon_s\right), \left(k', \epsilon_{s'}\right)\right) d\lambda_t(k, \epsilon_s),$$

- Stationary distribution is defined as $\lambda(k, \epsilon_s)$ such that

$$\lambda\left(k', \epsilon_{s'}\right) = \sum_s \int Q\left(\left(k, \epsilon_s\right), \left(k', \epsilon_{s'}\right)\right) d\lambda(k, \epsilon_s)$$

Calculation of Stationary Distribution

Stationary distribution

$$\lambda\left(k', \epsilon_{s'}\right) = \sum_{s} \int Q\left(\left(k, \epsilon_s\right), \left(k', \epsilon_{s'}\right)\right) d\lambda(k, \epsilon_s)$$

- Our goal is to numerically calculate the stationary distribution.
- Generally, there are two methods.
- Discretization Method
  Approximate transition probability density function $Q((k, \epsilon_s), (k', \epsilon_{s'}))$ by a Markov transition matrix Q.
  Then we can calculate stationary distribution by this Markov transition matrix Q.
- Stochastic Simulation Method
  Simulates a large number of households over a long period of time. Then we can finally obtain the stationary distribution.

Stationary Distribution: Discretization Method

Idea:

- First, imagine an ideal case: policy function $k' = g(k_n, \epsilon_s)$ happens to lie on the girds $\mathcal{K} = \{k_1, k_2, \ldots, k_N\}$.
  That is, for any k', there exists a $n' \in \mathcal{N} = \{1, 2, \ldots, N\}$, such that $k' = k_{n'}$.
- Then things become easy. Q becomes a $NS \times NS$ transition matrix:

$$Q\left((n,s),(n',s')\right) = \begin{cases} \mathcal{P}(s,s') & \text{if } n' = g(n,s) \\ 0 & \text{if } n' \neq g(n,s) \end{cases}$$

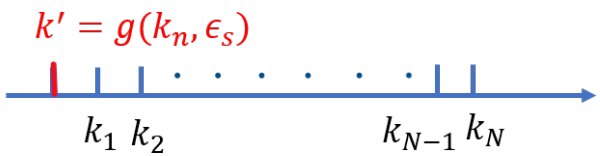
- But we know in reality, it is almost impossible that k' exactly lies on the grid points.
- Then, one feasible way is that we assign probability values to gird points based on their distance to k'.

Stationary Distribution: Discretization Method

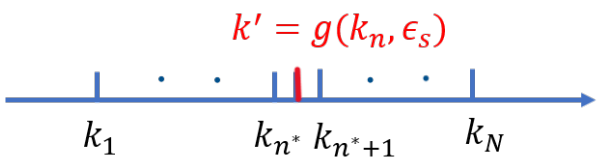
Eric Young's Method (2010, JEDC) to obtain a $NS \times NS$ transition matrix Q

- For each (k_n, ϵ_s), we can calculate $Q((n,s),(n',s'))$ $(n' \in \mathcal{N} = \{1,2,\ldots,N\},\ s' \in \mathcal{S} = \{1,2,\ldots,S\})$ by the following way:

$$Q((n,s),(n',s')) = \begin{cases} 0, n' = 1,\ldots,N-1 \\ P(s,s'), \quad n' = N \end{cases}$$



$$Q((n,s),(n',s')) = \begin{cases} 0, n' = 2,\ldots,N \\ P(s,s'), n' = 1 \end{cases}$$



$$Q((n,s),(n',s')) = \begin{cases} 0, n' = 1,\ldots,n^*-1 \\ \dfrac{k_{n^*+1} - k'}{k_{n^*+1} - k_{n^*}} P(s,s'), n' = n^* \\ \dfrac{k' - k_{n^*}}{k_{n^*+1} - k_{n^*}} P(s,s'), n' = n^*+1 \\ 0, n' = n^*+2,\ldots,N \end{cases}$$

Stationary Distribution: Discretization Method

Calculate stationary distribution from transition matrix Q.

- Probability Evolution

$$\lambda_t = \lambda_t(k_n, \epsilon_s)$$

$$\lambda_{t+1} = Q^T \lambda_t$$

- Stationary Distribution

$$\lambda = Q^T \lambda$$

- Two methods
  - Method of eigenvalue and eigenvector
    λ is the eigenvector which corresponds to eigen value 1 of matrix Q^T.
  - Iteration
    Again, it is a equation solving problem. Just use fixed point iteration.

Stationary Distribution: Stochastic Simulation

- Step 0

  Fix I agents, T periods, and an initial distribution $\left(k_0^i, s_0^i\right)_{i=1}^I$.

- Step 1

  In $0 \leq t \leq T-1$, use the policy function $k' = g(k, s)$ to calculate $\left(k_{t+1}^i\right)_{i=1}^I$ for each $i \in I$, i.e.

  $$k_{t+1}^i = g\left(k_t^i, s_t^i\right)$$

  and use transition matrix $\mathcal{P}\left(s, s'\right)$ of shock s and a random number generator to generate $\left(s_{t+1}^i\right)_{i=1}^I$

- Step 2

  Collect the simulated panel data with $(T+1)$ periods and I households, $\left(k_t^i, s_t^i\right)_{i=1,t=0}^{I,T}$.

- Step 3

  If the change in distributions is small between $T-1$ and T, stop. Otherwise, pick a larger T and go back to Step 0.

Solve Equilibrium

Capital Market Clearing Condition

- Capital demand from firms: K
- Capital supply from household:

$$\sum_{s=1}^{S}\sum_{n=1}^{N}\lambda(k_n,\epsilon_s)g(k_n,\epsilon_s)$$

or equivalently

$$\sum_{n=1}^{N}\left(\sum_{s=1}^{S}\lambda(k_n,\epsilon_s)\right)k_n$$

- Market clears:

$$K=\sum_{s=1}^{S}\sum_{n=1}^{N}\lambda(k_n,\epsilon_s)g(k_n,\epsilon_s)=\sum_{n=1}^{N}\left(\sum_{s=1}^{S}\lambda(k_n,\epsilon_s)\right)k_n$$

Equilibrium Conditions

Recall our equilibrium conditions.

- Given (K, N), (w, r) is determined competitively by

$$w = (1-\alpha)\left(\frac{K}{N}\right)^{\alpha}, \quad r = \alpha\left(\frac{K}{N}\right)^{\alpha-1} - \delta$$

- Given (r, w), $g(k, \epsilon)$ is the policy function from household's dynamic programming problem.
- Given policy function $g(k, \epsilon)$ and transition matrix P, $\lambda(k, \epsilon)$ is the stationary distribution.
- Market clearing condition for K and N:

$$K = \sum_{s=1}^{S}\sum_{n=1}^{N}\lambda(k_n, \epsilon_s)g(k_n, \epsilon_s), \quad N = \sum_{s=1}^{S}\sum_{n=1}^{N}\lambda(k, \epsilon_s)\epsilon_s\bar{l} = \sum_{s=1}^{S}\mu(\epsilon_s)\epsilon_s\bar{l}$$

where μ is the invariant distribution of labor productivity shock, given by $P^{-1}\mu = \mu$.

Equilibrium Conditions

We know in this model, N is exogenously determined by P and $\bar{l}$. Then equilibrium conditions can be summarized as a equation of K: $f(K) = 0$, where function value $f(K)$ is defined by the following procedure.

- Step 1: Given $N = \sum_{s=1}^{S} \mu(\epsilon_s)\epsilon_s \bar{l}$ and K, solve (w, r) by $w = (1-\alpha)\left(\frac{K}{N}\right)^{\alpha}, \quad r = \alpha\left(\frac{K}{N}\right)^{\alpha-1} - \delta$
- Step 2: Given (r, w), solve a DP problem to obtain policy function $g(k, \epsilon)$.
- Step 3: Given policy function $g(k, \epsilon)$ and transition matrix P, solve the stationary distribution $\lambda(k, \epsilon)$.
- Step 4: From $\lambda(k, \epsilon)$ and $g(k_n, \epsilon_s)$, calculate capital supply

$$K^S = \sum_{s=1}^{S} \sum_{n=1}^{N} \lambda(k_n, \epsilon_s) g(k_n, \epsilon_s)$$

- Step 5: Define $f(K) = K - K^S$, which can be interpreted as excess demand for capital

Hence, by market clear condition, excess demand is zero: $f(K) = 0$.

Solve for Equilibrium

Again, we have an equation solving problem. Apply equation solving methods to solve the equilibrium.
For example: A Dampened Fixed Point Iteration.
Procedure:

- Step 0: Choose an initial conjecture for capital demand $K^0 > 0$, a stopping criterion $\varepsilon > 0$, and a parameter $\gamma \in (0, 1]$.
- Step 1. In Iteration $0 \leq j \leq J$, start with K^j and compute r^j and w^j from pricing functions.
- Step 2. Given $\left(r^j, w^j\right)$, compute the household problem to get $g^j(k, s)$ and associated stationary distribution $\lambda^j(k, s)$.
- Step 3. Calculate capital supply $\widehat{K}^j = \sum_{k,s} \lambda^j(k, s)k$
- Step 4. If $\left|K^j - \widehat{K}^j\right| \leq \varepsilon$, stop. Otherwise, let $K^{j+1} = (1 - \gamma)K^j + \gamma\widehat{K}^j$ and go back to step 1