An Advanced Course On The Science and Art of DSGE Modelling

Day 3: Classical Heterogeneous Agent Models

Szabolcs Deák

Kirill Shakhnov

Hyungseok Joo

September 2021

No Aggregate Uncertainty

No Aggregate Uncertainty

- We start with a simplified model: there is no aggregate uncertainty, $A_t \equiv A$
- This is the setup studied by Aiyagari (1994).
- No aggregate uncertainty:
  - Idiosyncratic risk is the only source of uncertainty
  - Aggregate variables are constant: including the wealth distribution
- Wealth distribution still matters for aggregate variables, but it is not time-varying.

Model Summary

The model consists of the household's maximization problem

$$V(k,\varepsilon,\lambda,A) = \max_{k'} U(Rk + \varepsilon W - k') + \beta E\{V(k',\varepsilon',\lambda',A')|\varepsilon,A\} \tag{14}$$

$$\text{s.t.} \quad \lambda' = \Gamma(\lambda, A, A') \tag{15}$$

$$k' \geq \underline{k}, \tag{16}$$

the firm's first order conditions

$$W = \alpha A^{\alpha} \left(\frac{K}{H}\right)^{1-\alpha} \tag{17}$$

$$R = (1-\alpha) A^{\alpha} \left(\frac{K}{H}\right)^{-\alpha} + 1 - \delta, \tag{18}$$

and the market clearing conditions

$$K' = \int k' d\lambda'(k',\varepsilon) \tag{19}$$

$$H = E\varepsilon \tag{20}$$

A Generic Solution Algorithm

- We need to operationalize the steps from our generic algorithm:

  1. Guess the law of motion Γ.
  2. Solve for the optimal individual decision rules given the law of motion.
  3. Aggregate the individual decisions in order to determine the aggregate demand functions.
  4. If the resulting aggregate demand functions are consistent with our guess about Γ, then we have found the solution.
  5. Otherwise update our guess. Rinse and repeat.

Guessing the Law of Motion

Guessing the Law of Motion

- Recall the households' maximization problem:

$$V(k,\varepsilon,\lambda,A) = \max_{k'} U(Rk + \varepsilon W - k') + \beta E\{V(k',\varepsilon',\lambda',A')|\varepsilon, A\} \tag{14}$$

$$\text{s.t.} \quad \lambda' = \Gamma(\lambda, A, A') \tag{15}$$

$$k' \geq \underline{k} \tag{16}$$

- No aggregate uncertainty:
  - All aggregate variables are constant
  - $\lambda' = \lambda$, $A' = A$: we can simplify the maximization problem

Guessing the Law of Motion

- Recall the households' maximization problem:

$$V(k,\varepsilon) = \max_{k'} U(Rk + \varepsilon W - k') + \beta E\{V(k',\varepsilon')|\varepsilon\} \tag{21}$$

$$\text{s.t.} \quad k' \geq \underline{k} \tag{22}$$

- No aggregate uncertainty:
  - All aggregate variables are constant
  - $\lambda' = \lambda$, $A' = A$: we can simplify the maximization problem

Guessing the Law of Motion

- Recall the households' maximization problem:

$$V(k,\varepsilon) = \max_{k'} U(Rk + \varepsilon W - k') + \beta E\{V(k',\varepsilon')|\varepsilon\} \tag{21}$$

$$\text{s.t.} \quad k' \geq \underline{k} \tag{22}$$

- No aggregate uncertainty:
  - All aggregate variables are constant
  - $\lambda' = \lambda$, $A' = A$: we can simplify the maximization problem
  - So we *know* what the law of motion is.
  - In order to compute R and W: we need to know the (constant) distribution

Guessing the Law of Motion

- Recall the households' maximization problem:

$$V(k,\varepsilon) = \max_{k'} U(Rk + \varepsilon W - k') + \beta E\{V(k',\varepsilon')|\varepsilon\} \tag{21}$$

$$\text{s.t.} \quad k' \geq \underline{k} \tag{22}$$

- No aggregate uncertainty:
  - All aggregate variables are constant
  - $\lambda' = \lambda$, $A' = A$: we can simplify the maximization problem
  - So we *know* what the law of motion is.
  - In order to compute R and W: we need to know the (constant) distribution
  - But do we need to guess the entire distribution?

Guessing the Law of Motion

- Both factor prices can be expressed as functions of the aggregate capital-labour ratio:

$$W = \alpha A^{\alpha} \left(\frac{K}{H}\right)^{1-\alpha} \tag{17}$$

$$R = (1-\alpha) A^{\alpha} \left(\frac{K}{H}\right)^{-\alpha} + 1 - \delta \tag{18}$$

- Aggregate labour supply is constant: it is given by the unconditional mean of idiosyncratic labour productivity (from market clearing)

$$H = E\varepsilon$$

Guessing the Law of Motion

- Consequently: we can compute factor prices given aggregate capital, K.
- So we only need to guess the first moment of the wealth distribution.
- We still need to compute the entire distribution to solve the model, but no need to guess the entire distribution.

A Generic Solution Algorithm

- We need to operationalize the steps from our generic algorithm:

  1. Guess the law of motion Γ.
  2. Solve for the optimal individual decision rules given the law of motion.
  3. Aggregate the individual decisions in order to determine the aggregate demand functions.
  4. If the resulting aggregate demand functions are consistent with our guess about Γ, then we have found the solution.
  5. Otherwise update our guess. Rinse and repeat.

Optimal Individual Decision Rules

Optimal Individual Decision Rules

- We need to solve:

$$V(k,\varepsilon) = \max_{k'} U(Rk + \varepsilon W - k') + \beta E\{V(k',\varepsilon')|\varepsilon\} \tag{21}$$

$$k' \geq \underline{k} \tag{22}$$

given R and W implied by our guess on K.

- Nonlinear problem: value function iteration

Optimal Individual Decision Rules

- We need to solve:

$$V(k,\varepsilon) = \max_{k'} U(Rk + \varepsilon W - k') + \beta E\{V(k',\varepsilon')|\varepsilon\} \tag{21}$$

$$k' \geq \underline{k} \tag{22}$$

given R and W implied by our guess on K.

- Labour productivity: 2-state Markov chain
  - can take only two values: $\varepsilon_1 = \varepsilon^L < \varepsilon_2 = \varepsilon^H$
  - the transition probabilities of these two realizations are defined by the 2×2 matrix $\mathcal{P}^\varepsilon$

Optimal Individual Decision Rules

- We need to solve:

$$V(k,\varepsilon_j) = \max_{k'} U(Rk + \varepsilon_j W - k') + Q(k',\varepsilon_j),\ j = 1,2 \tag{23}$$

$$\text{s.t.} \quad Q(k',\varepsilon_j) = \beta E\{V(k',\varepsilon')|\varepsilon_j\} = \beta \sum_{m=1}^{2} \mathcal{P}^{\varepsilon}_{mj} V(k',\varepsilon_l) \tag{24}$$

$$k' \geq \underline{k} \tag{25}$$

given R and W implied by our guess on K.

- Labour productivity: 2-state Markov chain
  - can take only two values: $\varepsilon_1 = \varepsilon^L < \varepsilon_2 = \varepsilon^H$
  - the transition probabilities of these two realizations are defined by the 2×2 matrix $\mathcal{P}^{\varepsilon}$
  - Note: We use left stochastic matrices whose columns sum up to 1

Optimal Individual Decision Rules

- We need to solve:

$$V(k,\varepsilon_j) = \max_{k'} U(Rk + \varepsilon_j W - k') + Q(k',\varepsilon_j),\ j = 1,2 \tag{23}$$

$$\text{s.t.} \quad Q(k',\varepsilon_j) = \beta E\{V(k',\varepsilon')|\varepsilon_j\} = \beta \sum_{m=1}^{2} \mathcal{P}^{\varepsilon}_{mj} V(k',\varepsilon_l) \tag{24}$$

$$k' \geq \underline{k} \tag{25}$$

given R and W implied by our guess on K.

- $Q(k',\varepsilon)$:
  - the continuation value of the savings decision
  - fully determined by $V(k,\varepsilon)$ and $\mathcal{P}^{\varepsilon}$
  - a convenient tool when we implement the algorithm

Optimal Individual Decision Rules

- We skip the implementation details (see Day 1).
- Capital is represented by an n-point discrete grid:

$$\{\underbrace{k_1 = \underline{k}}_{\text{borrowing constraint}}, k_2, \ldots, k_{n-1}, \underbrace{k_n = \overline{k}}_{\text{suitably chosen upper bound}}\}$$

- The solution is the decision rule $k' = g(k, \varepsilon)$:
  - We restrict the households to choose one of the n gridpoints as their capital holding tomorrow, k'.
  - For both values of the idiosyncratic productivity shock ε_j, $j = 1, 2$, the decision rule can be represented by an $n \times 1$ vector whose lth element is $k' = g(k_l, \varepsilon_j)$.

A Generic Solution Algorithm

- We need to operationalize the steps from our generic algorithm:

  1. Guess the law of motion Γ.
  2. Solve for the optimal individual decision rules given the law of motion.
  3. Aggregate the individual decisions in order to determine the aggregate demand functions.
  4. If the resulting aggregate demand functions are consistent with our guess about Γ, then we have found the solution.
  5. Otherwise update our guess. Rinse and repeat.

Aggregation

Aggregation

- Market clearing for capital requires that:

$$K' = \int k' d\lambda(k', \varepsilon) \tag{19}$$

- We need to determine $\lambda(k', \varepsilon)$ to evaluate the integral.

Aggregation

- We have discretized the state space:
  - there are n possible asset levels, $\{k_1, \ldots, k_n\}$
  - there are 2 possible productivity levels, $\{\varepsilon_1, \varepsilon_2\}$
- We have restricted households to choose one of the n gridpoints for capital.
- So there are $2n$ distinct types of households and we can represent the distribution $\lambda(k, \varepsilon)$ as an $n \times 2$ matrix:

$$\lambda(k, \varepsilon) = \begin{bmatrix} \lambda(k_1, \varepsilon_1) & \lambda(k_1, \varepsilon_2) \\ \vdots & \vdots \\ \lambda(k_n, \varepsilon_1) & \lambda(k_n, \varepsilon_2) \end{bmatrix} \tag{26}$$

and the integral can be computed as

$$\int k' d\lambda(k', \varepsilon) = \sum_{l=1}^{n} \sum_{j=1}^{2} \lambda(k_l, \varepsilon_j) k_l \tag{27}$$

Aggregation

- There are two different methods in the literature:

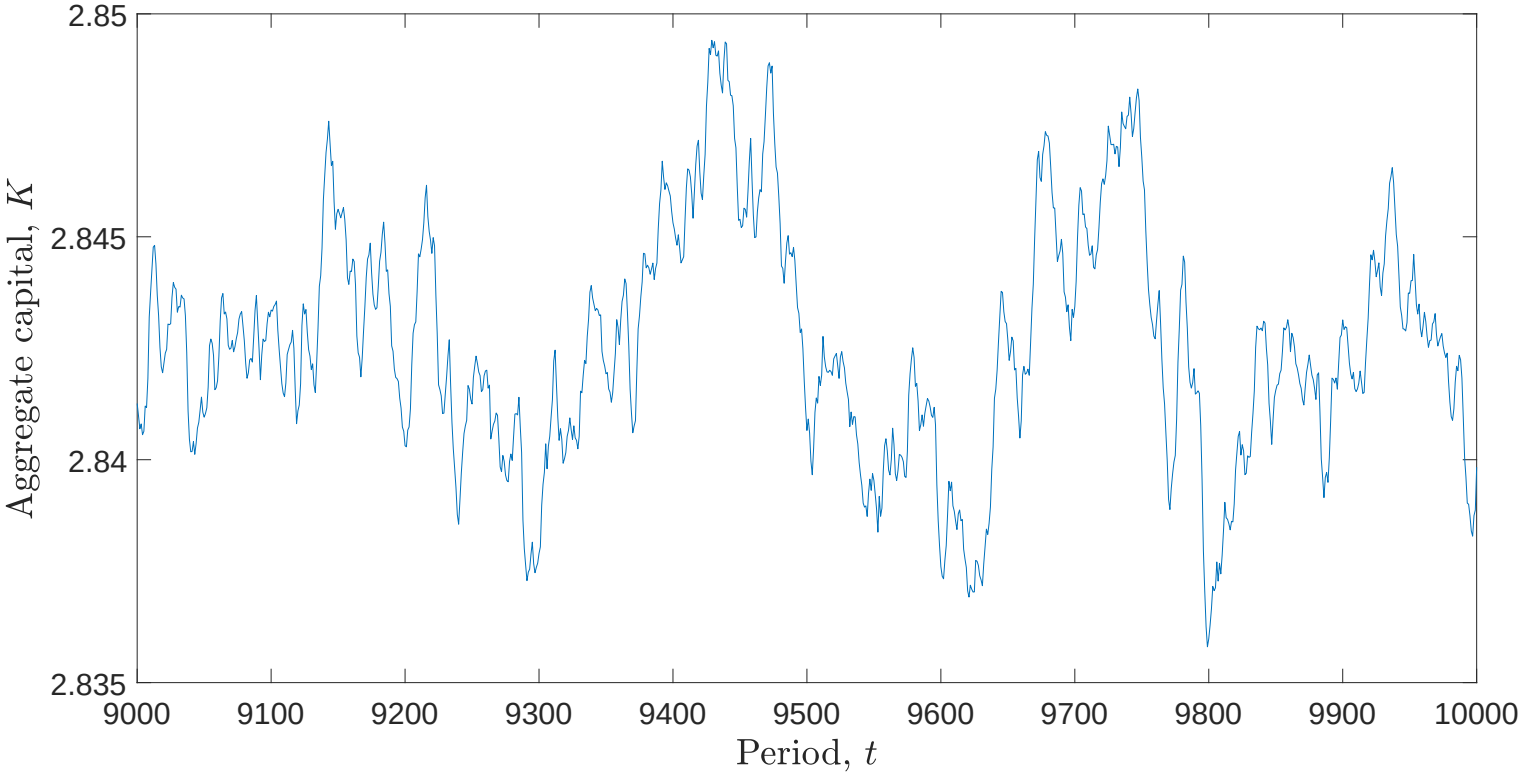
  1. **Aggregation by simulation:**
     - We simulate the model to determine the distribution from the simulated sample.
     - Easy to implement, but subject to sampling uncertainty.
  2. **Aggregation by non-stochastic simulation:**
     - No need to generate idiosyncratic shocks.
     - We determine the evolution of the histogram directly, i.e. we simulate the distribution itself.
     - May be computationally costly, but no sampling uncertainty.

- We discuss non-stochastic simulation among the alternative solution methods.

Aggregation

- Let us look at simulations from the Aiyagari model:
  - I have solved the model.
  - I have simulated a panel of $N = 100,000$ households for $T = 10,000$ periods.
  - I have evaluated the market clearing condition to compute aggregate capital.

Aggregation



- Not converging even after $10,000$ periods.
- Actually: it never will.

Aggregation

- The reason: sampling uncertainty
- In period $t = 9,400$ for example:
  - $10,050$ households have low productivity (ε_1)
  - The transition probabilities used in the model:

$$\mathcal{P}^{\varepsilon}_{11} = 0.6 \qquad\qquad \mathcal{P}^{\varepsilon}_{21} = 0.4$$

  - So $10,050 \times 0.6 = 6,030$ should stay in the same state.
  - The random numbers generated by Matlab imply that actually $6,035$ households stay.
  - Sampling uncertainty implies that the actual transition probabilities are biased: the Law of Large Numbers does not hold in finite samples.

Aggregation

- Two ways to solve the problem:
  1. "Fix" the transition probabilities.
  2. Change the size of the cross-section (N).

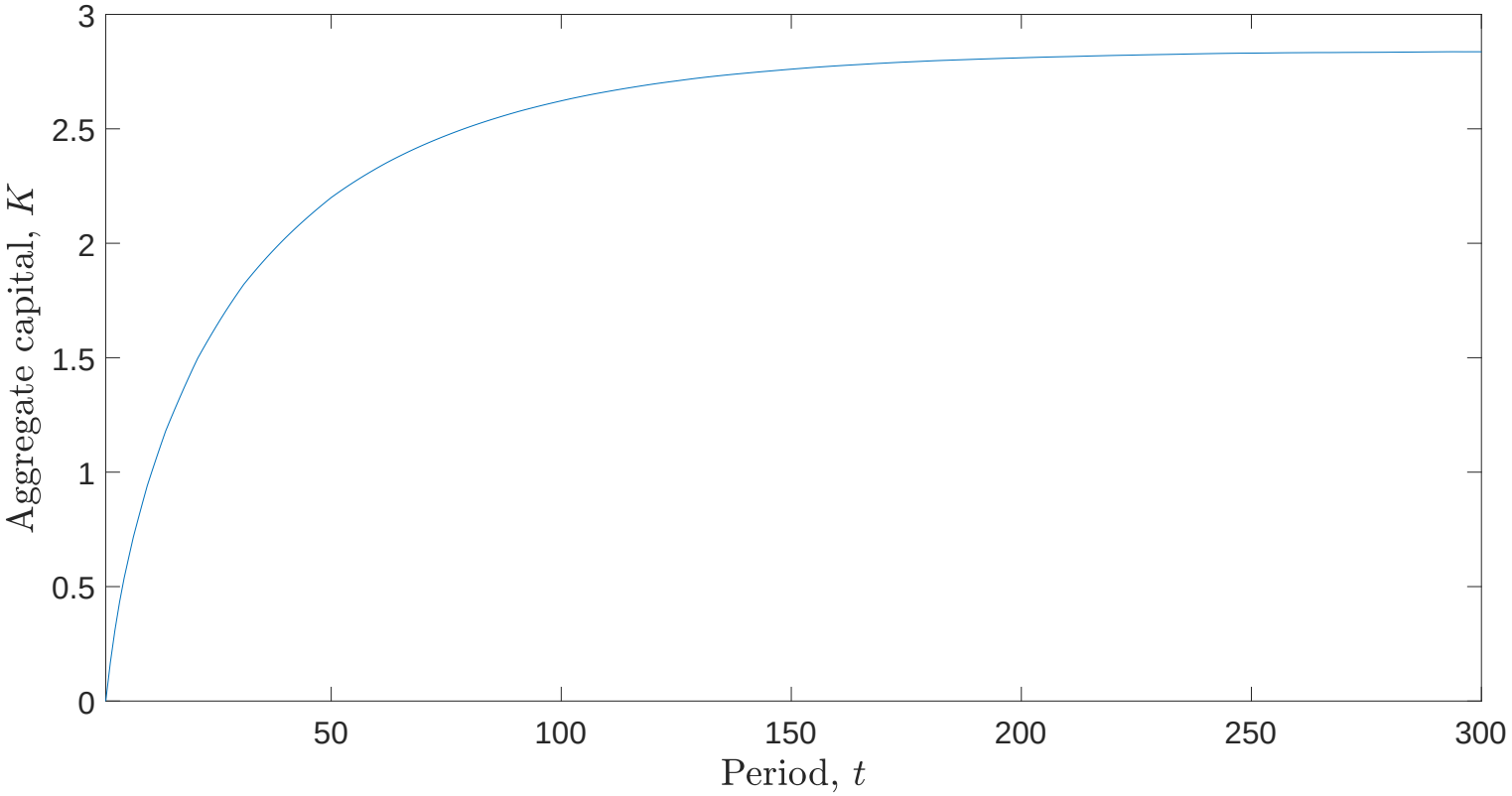
Aggregation

- We can generate shocks that "respect" the transitional probabilities; just like Krusell and Smith (1998) do.
- The way we usually generate 2-state Markov chains:
  1. For each low productivity household generate a uniform random number in $[0, 1]$.
  2. If a household has a random number ≤ 0.60, then the household stays in the same state.
  3. Otherwise the household transitions to the other state.
- This is how the previous plot was generated.
- And this is how the Law of Large Numbers gets violated.

Aggregation

- We can generate shocks that "respect" the transitional probabilities; just like Krusell and Smith (1998) do.
- We can modify this as follows:
  1. For each low productivity household generate a uniform random number in $[0, 1]$.
  2. Sort them in increasing order of their generated random numbers.
  3. The first $6,030$ households (with the lowest numbers) will stay in the same state.
  4. The rest (households with the highest 40 per cent of the random numbers generated) transition to the other state.
- This is how the next plot was generated.

Aggregation



- Convergence is quick.
- Sampling uncertainty is completely eliminated; which is the basic idea behind non-stochastic simulation.

Aggregation

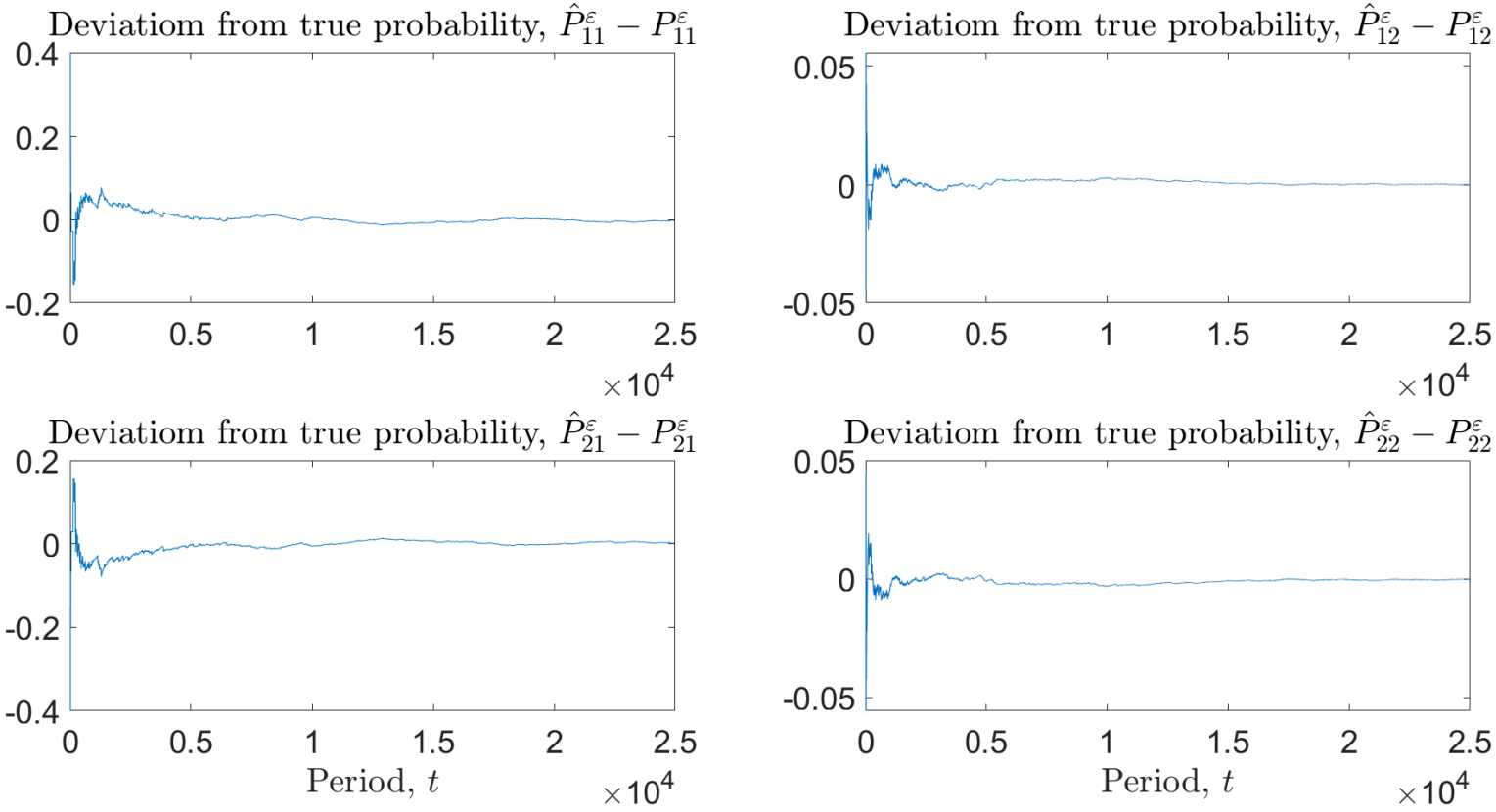
- We can also change the size of the cross-section; just like many papers in the literature do.
- Your first thought probably: increase N.
- It would certainly decrease the transitional probabilities' bias.
- But the bias would asymptotically disappear only of we kept increasing N. Which is what you need for convergence.
- But N needs to be kept fixed.

Aggregation

- We can also change the size of the cross-section; just like many papers in the literature do.
- Or we can simulate a single household only ($N = 1$).
- As we increase T the transitional probabilities for that household converge to their true value asymptotically.
- What is different? We compute the transitional probabilities longitudinally and not cross-sectionally.

Aggregation

- We can also change the size of the cross-section; just like many papers in the literature do.

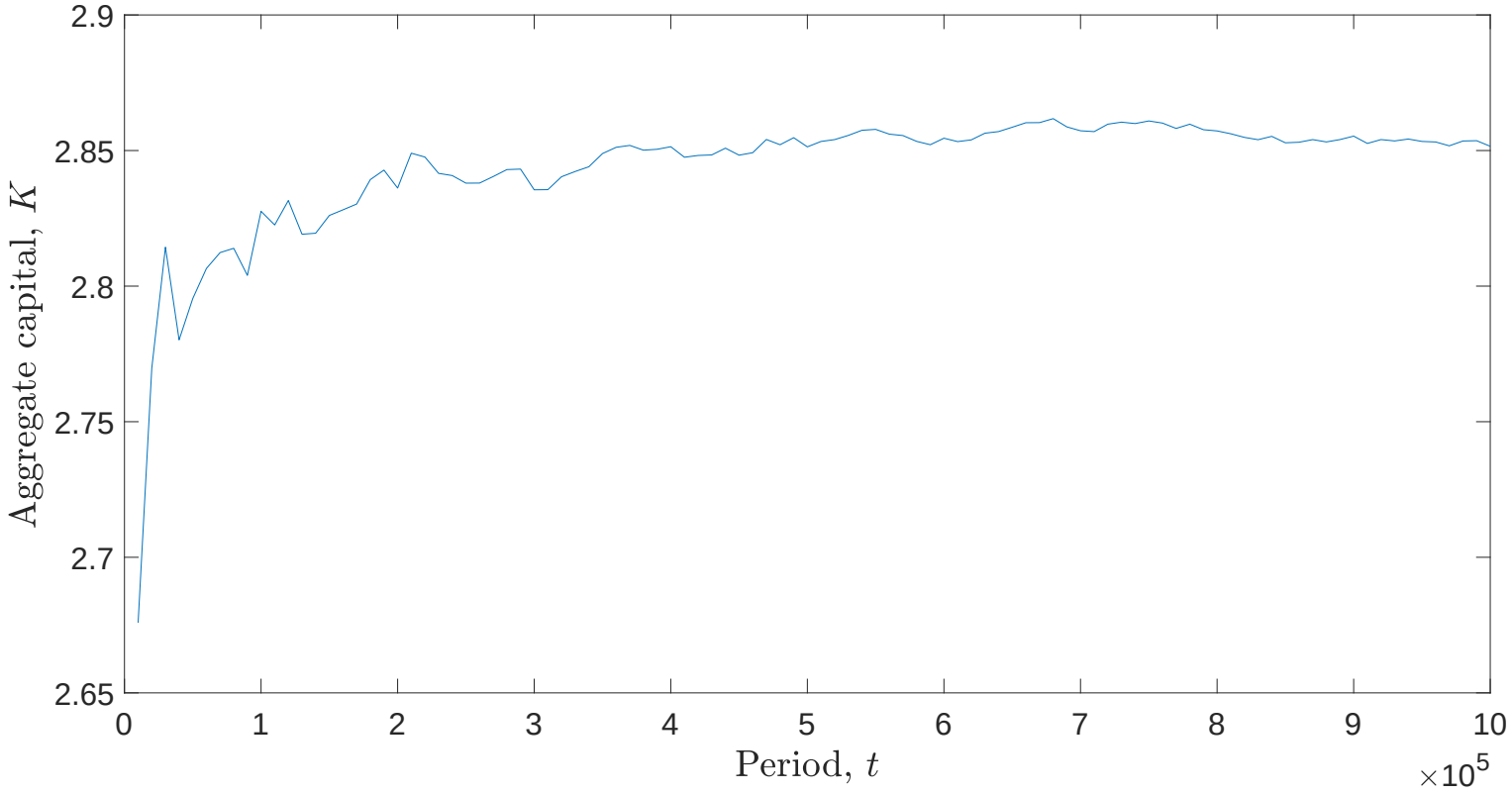


Aggregation

- We can also change the size of the cross-section; just like many papers in the literature do.
- But can we obtain the *cross-sectional* wealth distribution from a single household?
- Uhlig (1996) shows that:
  - The fraction of time any given household spends in each state (k_l, ε_j) over time is the same as the fraction of households in each state (k_l, ε_j) in the stationary cross-sectional wealth distribution.
  - So the distribution of the states a single household visits over time asymptotically converges to the stationary cross-sectional wealth distribution.

Aggregation

- We can also change the size of the cross-section; just like many papers in the literature do.



A Generic Solution Algorithm

- We need to operationalize the steps from our generic algorithm:

  1. Guess the law of motion Γ.
  2. Solve for the optimal individual decision rules given the law of motion.
  3. Aggregate the individual decisions in order to determine the aggregate demand functions.
  4. If the resulting aggregate demand functions are consistent with our guess about Γ, then we have found the solution.
  5. Otherwise update our guess. Rinse and repeat.

Evaluating our Guess

Evaluating our Guess

- Recall that no aggregate uncertainty implies constant aggregate variables:

$$K = K' \tag{38}$$

- Combine it with market clearing to get:

$$K = \int k' d\lambda(k', \varepsilon) \tag{29}$$

- LHS: aggregate capital at the beginning of the period that is used to determine prices (capital demand by firms)

- RHS: aggregated individual saving decisions implied by those prices (capital supply by households at the end of the period)

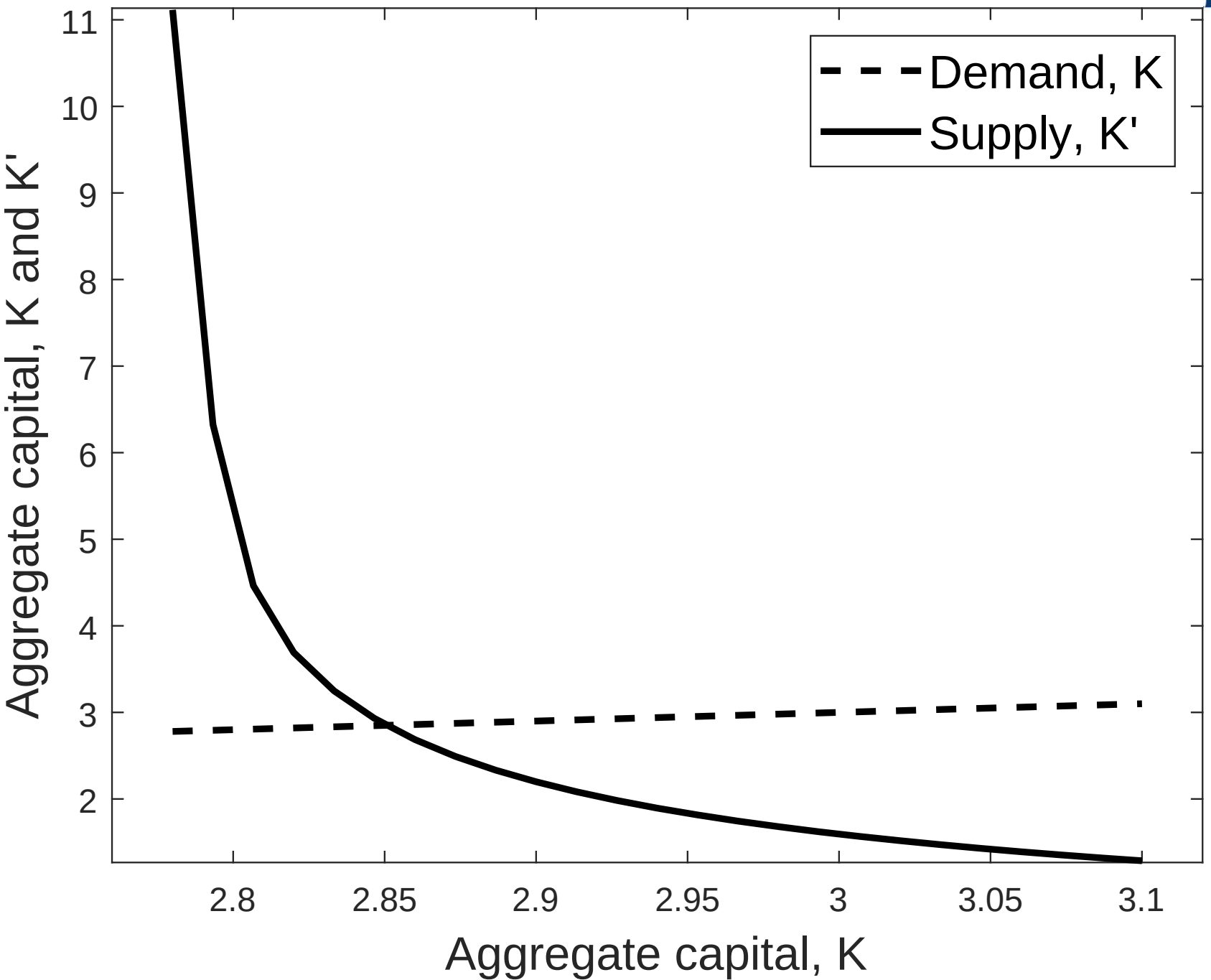
Evaluating our Guess

- Consistency condition:

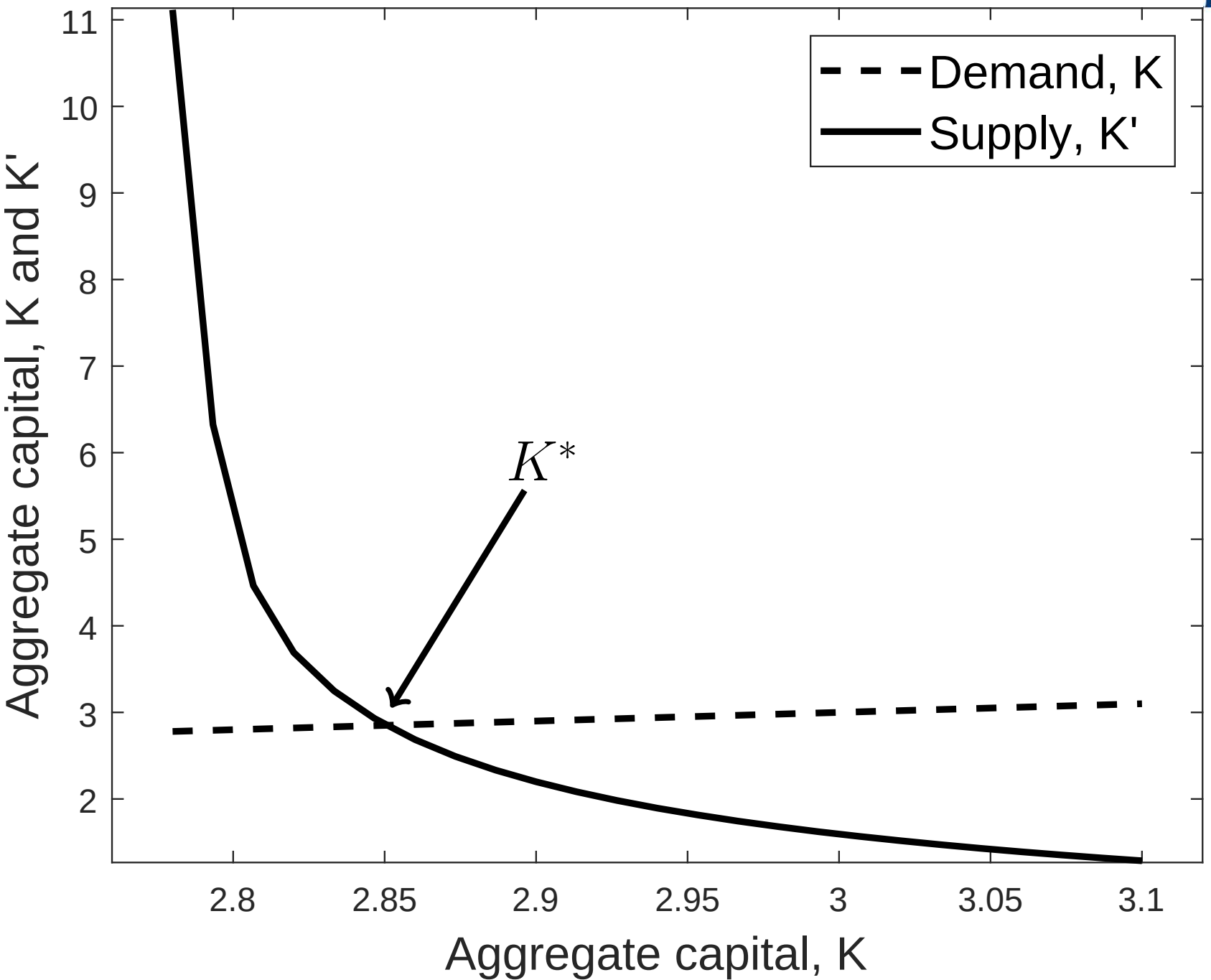
$$\underbrace{K}_{\text{demand}} = \underbrace{\int k' d\lambda(k', \varepsilon)}_{\text{supply}} \tag{39}$$

- Recall that we solve the individuals' problem given a guess about K, so both sides are functions of K:
  - The left-hand side is increasing in K
  - The right-hand side is decreasing in K

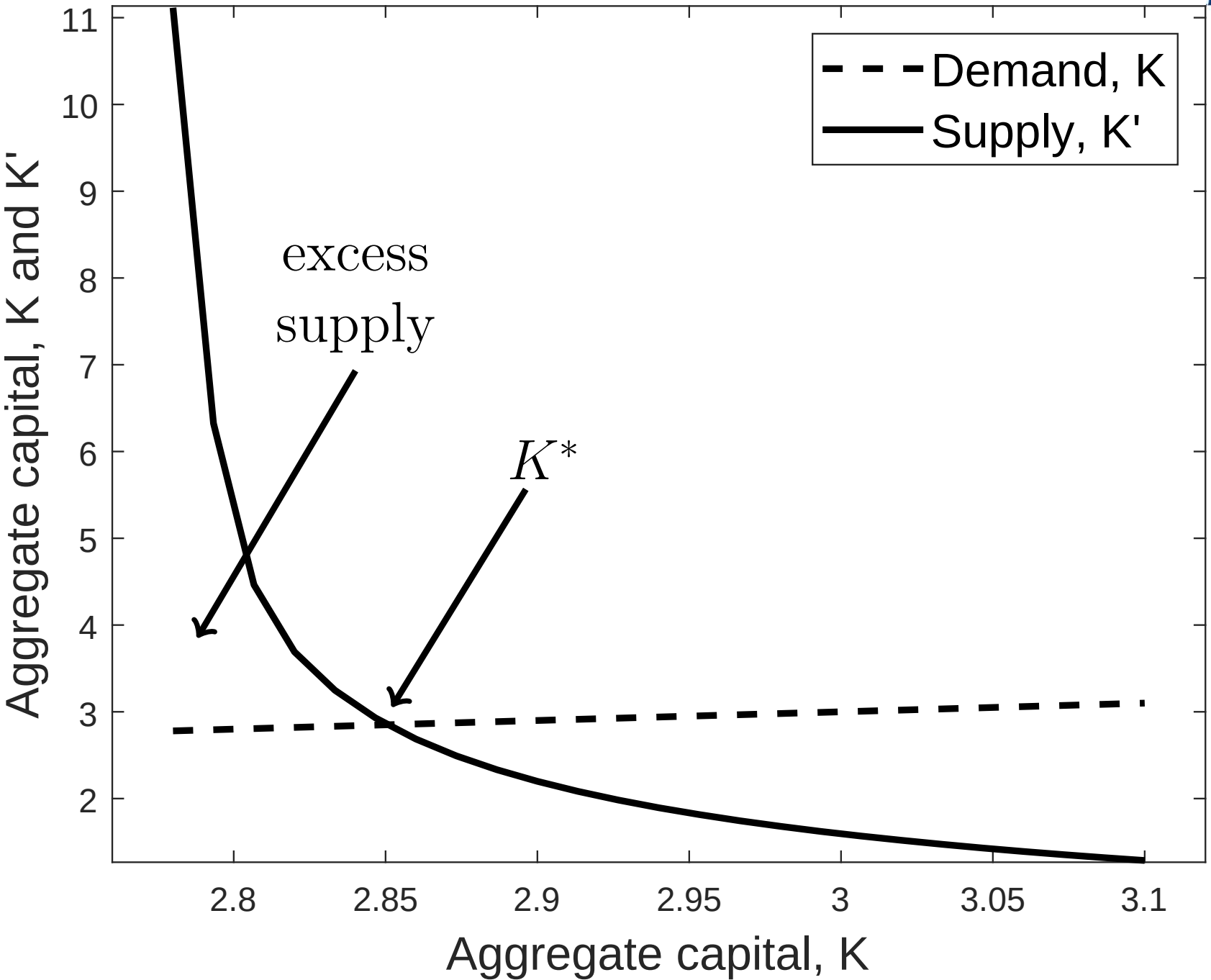
Evaluating our Guess



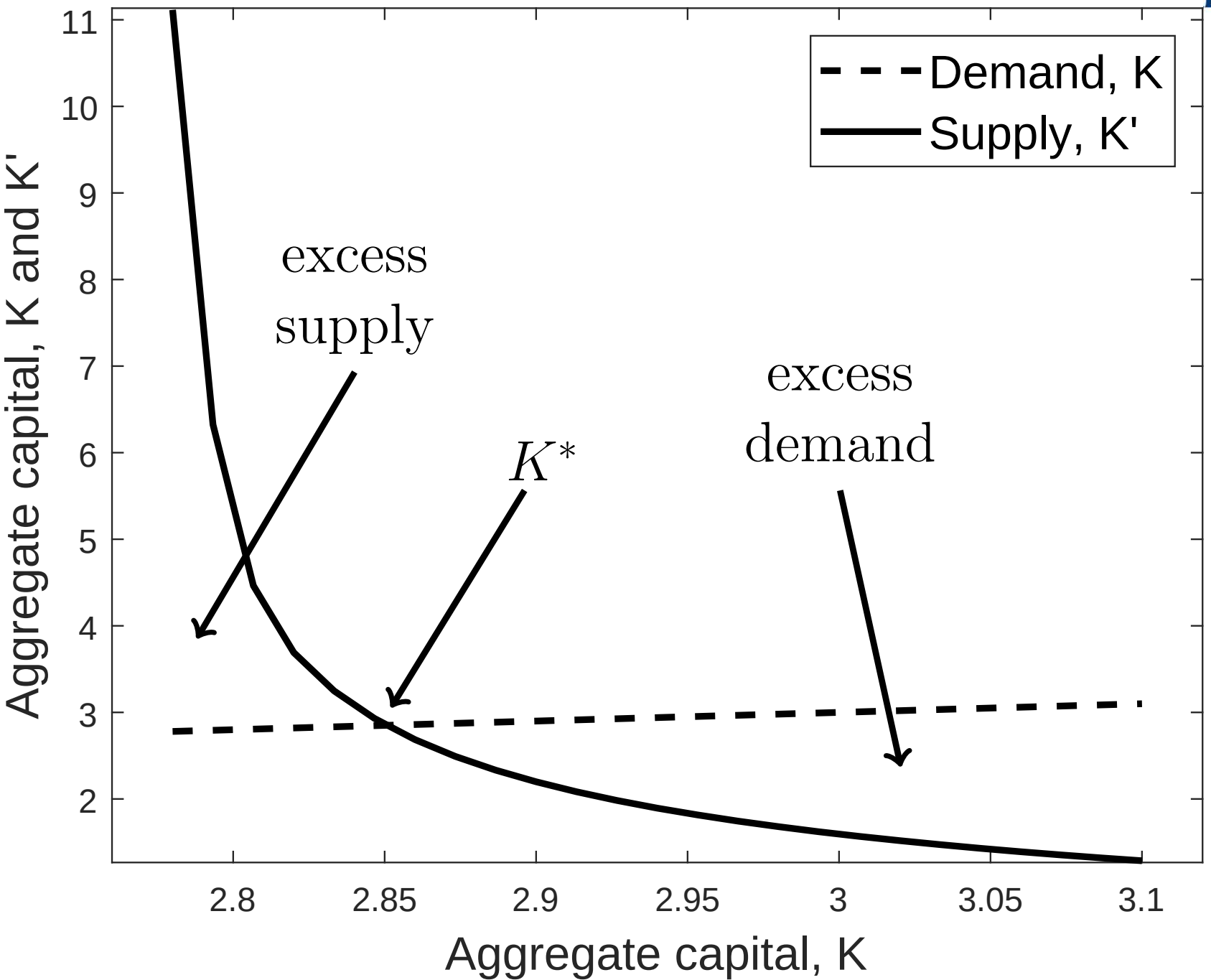
Evaluating our Guess



Evaluating our Guess



Evaluating our Guess



Evaluating our Guess

- We use bisection to solve for K^*:

1. Find two values for aggregate capital such that they bracket the solution, $\underline{K} < K^* < \overline{K}$; there is excess supply and demand of capital at $\underline{K}$ and $\overline{K}$, respectively.
2. Solve the model for the midpoint $(\underline{K} + \overline{K})/2$:
   i) Optimal individual decision rule
   ii) Stationary distribution
3. If there is excess supply of capital at the midpoint, then it is below the optimal solution and we replace the lower bound $\underline{K}$ with the midpoint. Otherwise we replace the upper bound $\overline{K}$ with the midpoint.
4. We iterate until the bracketing interval becomes sufficiently small.

Implementing the Algorithm

Implementing the Algorithm

- Utility function:

$$U(c) = \log c$$

- Parameter values:

Parameter	$\underline{k}$	A	α	β	δ	ε_1	ε_2
Value	0	1	0.64	0.96	0.1	0.15	0.7

ε'/ε	ε_1	ε_2
ε_1	0.600	0.044
ε_2	0.400	0.956

Implementing the Algorithm

```
%% Parameters
% Model parameters
p.beta                 = 0.96;            % discount factor
p.alpha                = 0.64;            % labour elasticity of output
p.delta                = 0.1;             % depreciation rate

% Aggregate shock
p.A                    = 1;               % constant aggregate productivity

% Idiosyncratic shock
p.mean_length_lowprod = 2.5;              % expected duration of being in low productivity state
p.fraction_lowprod     = 0.1;             % fraction of low productivity households
p.e                    = [0.15 0.7];      % productivity values in the two states
p.N_e                  = length(p.e);     % number of grid points along e dimension
ergdist_e              = [p.fraction_lowprod 1-p.fraction_lowprod];
                                          % stationary distribution of productivity process

% Aggregate variables
p.H                    = ergdist_e*p.e'; % aggregate labour

% Solution method
p.K                    = ((1/p.beta-(1-p.delta))/(1-p.alpha))^(-1/p.alpha)*p.A*p.H;
                                          % steady state capital stock of repr. agent model
p.Klow                 = 0.5;             % the low and high values for K (in terms of p.K)
p.Khigh                = 2;               % for the initial bisection step

p.N_k                  = 500;               % individual capital grid size
p.k_min                = 0;               % minimum value of individual capital holding
p.k_max                = 7;               % maximum value of individual capital holding

p.tol_bis              = 1e-6;            % tolerance parameter for bisection
p.tol_vfi              = 1e-6;            % tolerance parameter for value function iteration
```

Implementing the Algorithm

```
%% Initialization
% Constructing the transition matrix
targets                 = [p.mean_length_lowprod p.fraction_lowprod]; % calibration targets
tran_prob               = tran_prob_aiyagari(targets);                % transition matrix (left-stochastic)

% Individual capital space
k                       = linspace(p.k_min,p.k_max,p.N_k)';

% Initializing bisection
K                       = [p.Klow p.Khigh].*p.K;                      % initial interval

% Allocating memory for matrices used within the main loop
V                       = zeros(p.N_k,p.N_e);                         % value function; also the initial guess

% Generating shocks for the simulation
p.T                     = 1005000;                                    % sample size to be generated during simulation
p.T_burn                = 5000;                                       % observations at the beginning of the sample to be droppe
rand('seed',1);                                                       % setting seed
epsi                    = zeros(p.T,1);                               % vector to save the shocks
epsi(1)                 = p.e(1);                                     % first realizaton is state 1
for t = 2:p.T
    previous_state      = epsi(t-1)==p.e;
    randomdraw          = rand;
    if randomdraw<tran_prob(1,previous_state)
        epsi(t) = p.e(1);
    else
        epsi(t) = p.e(2);
    end
end
```

Implementing the Algorithm

```
%% Let's party
iter       = 0;                                                    % iteration counter
keep_going = true;                                                 % exit flag

% main iteration
while keep_going
    iter                  = iter+1;
    K0                    = mean(K);                               % midpoint

    % Obtaining decision rule
    [V, pol_func, iter_V] = Individual_aiyagari(p, tran_prob, K0, k, V);
    % Note: we carry V over to the next iteration. It'll be our initial guess.

    % Obtaining stationary ditribution
    [lambda, kmax]        = Stationary_distribution_by_simulation_aiyagary(p, pol_func, epsi);
    Kprime                = sum(sum(bsxfun(@times,lambda,k)));
    % Note: lambda will also be our initial guess for the next iteration

    fprintf('Iteration %03d: interval %1.2f %2.2f, demand %s%2.2f supply %s%2.2f excess: %+1.1e, VFI iterations:
    if Kprime>K0
        K(1) = K0;                                                 % excess supply
    else
        K(2) = K0;                                                 % excess demand
    end
    length_of_interval    = diff(K)/K0;                            % is the interval still to wide?
    keep_going            = length_of_interval>p.tol_bis;
end
```

Implementing the Algorithm

```
% Individual capital space
k                   = linspace(p.k_min,p.k_max,p.N_k)';

% Simulating a single household
k_sim               = zeros(p.T+1,1);
k_sim(1)            = k(1);                                         % starting from the first grid
for t=1:p.T
    k_sim(t+1) = pol_func(k_sim(t)==k,epsi(t)==p.e);
end

% dropping some observations
k_sim(1)            = [];                                           % drop initial condition
k_sim(1:p.T_burn)   = [];                                           % drop burn-in sample
epsi(1:p.T_burn)    = [];                                           % drop burn-in sample

lambda              = hist3([k_sim epsi],{k,p.e});                  % computing the histogram over
lambda              = lambda./sum(lambda(:));                       % make sure it is a proper dis

maxk                = max(k_sim);
maxknum             = sum(k_sim==maxk);
```