Lecture 1
Solving the Standard Incomplete Markets Model

Adrien Auclert (Stanford University)

Paris Dauphine Minicourse
January 2024

Overview

- In the heterogeneous-agent macro literature, models can rarely be solved analytically

Overview

- In the heterogeneous-agent macro literature, models can rarely be solved analytically
- So we begin with a brief overview over how you can efficiently compute
    - steady states
    - transitional dynamics / impulse responses

  of the "standard incomplete markets" model
  (aka Bewley-Huggett-Aiyagari-Imrohoroglu-Zeldes-Deaton-Carroll model)
- This will be the backbone of everything we do

- In the heterogeneous-agent macro literature, models can rarely be solved analytically
- So we begin with a brief overview over how you can efficiently compute
    - steady states
    - transitional dynamics / impulse responses

  of the "standard incomplete markets" model
  (aka Bewley-Huggett-Aiyagari-Imrohoroglu-Zeldes-Deaton-Carroll model)
- This will be the backbone of everything we do
- For the most part, this is automated in the SSJ toolkit, but it's still good to know what's going on under the hood
    - cf Jupyter notebooks that go through every single step of this lecture in detail

Roadmap

1. Steady state
2. Dynamics

Steady state

The standard incomplete markets model (SIM)

$$
\begin{aligned}
V_t(s, a) = \max_{c, a'} u(c) + \beta \mathbb{E}[V_{t+1}(s', a') | s] \\
\text{s.t. } a' + c = (1 + r_t)a + y_t(s) \\
a' \geq \underline{a}
\end{aligned}
$$

The standard incomplete markets model (SIM)

$$V_t(s,a) = \max_{c,a'} u(c) + \beta \mathbb{E}[V_{t+1}(s',a')|s]$$
$$\text{s.t. } a' + c = (1+r_t)a + y_t(s)$$
$$a' \geq \underline{a}$$

The model generates **four key outcomes**:

1. endogenous buffer stock saving
2. a well defined stationary wealth distribution
3. high MPCs, consistent with those estimated in the data
4. a finitely elastic long-run asset demand curve

Four steps to solving the steady state

1. **Discretizing the state space:** Markov chain for s & grid for a.
2. **Solving for steady-state policy function:**
    - get policies $a'(s, a)$ and $c(s, a)$ on the grid
3. **Solving for steady-state distribution:**
    - get distribution $D(s, a)$ of agents on grid
4. **Aggregating:** get aggregate assets and consumption

Asset grid: could assume uniform grid ... but most action / nonlinearity will be close to borrowing constraint $\underline{a}$!

Asset grid: could assume uniform grid ... but most action / nonlinearity will be close to borrowing constraint $\underline{a}$!

Better choice: double exponential grid: If u_i is uniform on $[0, \overline{u}]$ then

$$a_i = \underline{a} + e^{e^{u_i} - 1} - 1$$

This makes the grid much denser at $\underline{a}$.

Asset grid: could assume uniform grid ... but most action / nonlinearity will be close to borrowing constraint $\underline{a}$!

Better choice: double exponential grid: If u_i is uniform on $[0, \overline{u}]$ then

$$a_i = \underline{a} + e^{e^{u_i} - 1} - 1$$

This makes the grid much denser at $\underline{a}$.

Markov chain for s: Typically want to approximate continuous Markov chain, e.g. AR(1) with persistence ρ. How do we discretize on n gridpoints?

Asset grid: could assume uniform grid ... but most action / nonlinearity will be close to borrowing constraint $\underline{a}$!

Better choice: double exponential grid: If u_i is uniform on $[0, \overline{u}]$ then

$$a_i = \underline{a} + e^{e^{u_i}-1} - 1$$

This makes the grid much denser at $\underline{a}$.

Markov chain for s: Typically want to approximate continuous Markov chain, e.g. AR(1) with persistence ρ. How do we discretize on n gridpoints?

We'll use **Rouwenhorst**'s method. Idea:

- $n-1$ switches (0 or 1). Each flips with probability $p = (1-\rho)/2$.
- [Also common: Tauchen method, but worse for usual $\rho \simeq 1$ case.]

We'll use two key optimality conditions:

Envelope condition:

$$V_{a,t}(s,a) = (1+r_t)u'(c_t(s,a)) \tag{1}$$

First order condition:

$$u'(c_t(s,a)) \geq \beta\mathbb{E}[V_{a,t+1}(s',a'_t(s,a))|s] \tag{2}$$

with equality unless borrowing constraint binds.

Step 2: Steady state policies

We'll use two key optimality conditions:

Envelope condition:

$$V_{a,t}(s,a) = (1+r_t)u'(c_t(s,a)) \tag{1}$$

First order condition:

$$u'(c_t(s,a)) \geq \beta \mathbb{E}[V_{a,t+1}(s', a'_t(s,a))|s] \tag{2}$$

with equality unless borrowing constraint binds.

Key insight of Carroll's (2006) Endogenous Gridpoints Method (EGM): More efficient & accurate to use (1) and (2) instead of value function iteration on V.

Step 2: Steady state policies

We'll use two key optimality conditions:

Envelope condition:

$$V_{a,t}(s,a) = (1 + r_t)u'(c_t(s,a)) \tag{1}$$

First order condition:

$$u'(c_t(s,a)) \geq \beta \mathbb{E}[V_{a,t+1}(s', a_t'(s,a))|s] \tag{2}$$

with equality unless borrowing constraint binds.

Key insight of Carroll's (2006) Endogenous Gridpoints Method (EGM): More efficient & accurate to use (1) and (2) instead of value function iteration on V.

This is what we'll do:

$$V_{a,t+1}(s', a') \overset{\text{FOC}}{\to} c_t^{endo}(s, a') \overset{\text{interpolate}}{\to} a_t'(s,a) \overset{\text{budget}}{\to} c_t(s,a) \overset{\text{Envelope}}{\to} V_{a,t}(s,a)$$

Step 2: Steady state policies: Details of EGM (backward iteration)

- First arrow is simple:

$$c_t^{endo}(s, a') = \left(u'\right)^{-1}\left(\beta \mathbb{E}[V_{a,t+1}(s', a'_t)|s]\right)$$

- First arrow is simple:

$$c_t^{endo}(s, a') = \left(u'\right)^{-1}\left(\beta \mathbb{E}[V_{a,t+1}(s', a'_t)|s]\right)$$

- But how to get to policy defined over current grid (s, a)? Note:

$$c_t^{endo}(s, a') + a' = \text{coh}_t(s, a) \equiv (1 + r_t)\, a + y_t(s)$$

We want to solve for a' as function of a.

- First arrow is simple:

$$c_t^{endo}(s, a') = (u')^{-1}\left(\beta \mathbb{E}[V_{a,t+1}(s', a'_t)|s]\right)$$

- But how to get to policy defined over current grid (s, a)? Note:

$$c_t^{endo}(s, a') + a' = \text{coh}_t(s, a) \equiv (1 + r_t)\, a + y_t(s)$$

  We want to solve for a' as function of a.

- Can do this with simple interpolation:
  - Plot $(c_t^{endo}(s, a') + a', a')$ points and interpolate at values of $\text{coh}_t(s, a)$...

Illustration of key EGM step

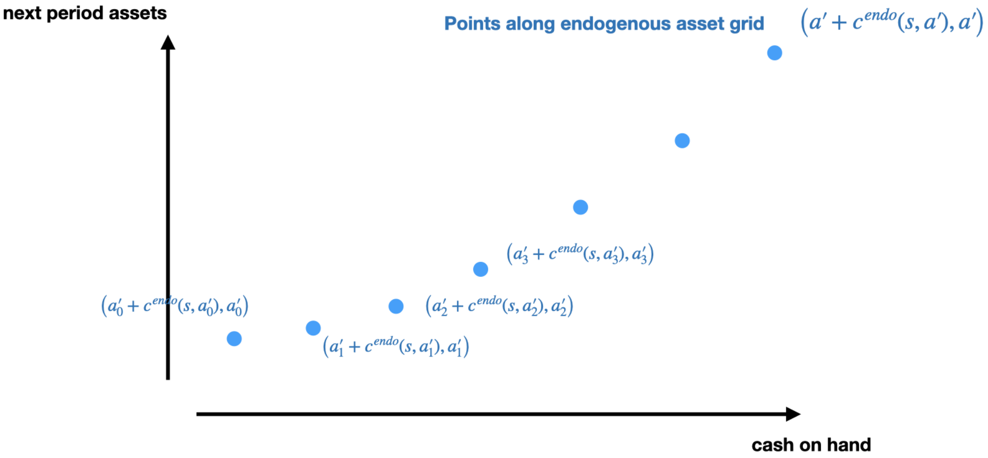


Illustration of key EGM step

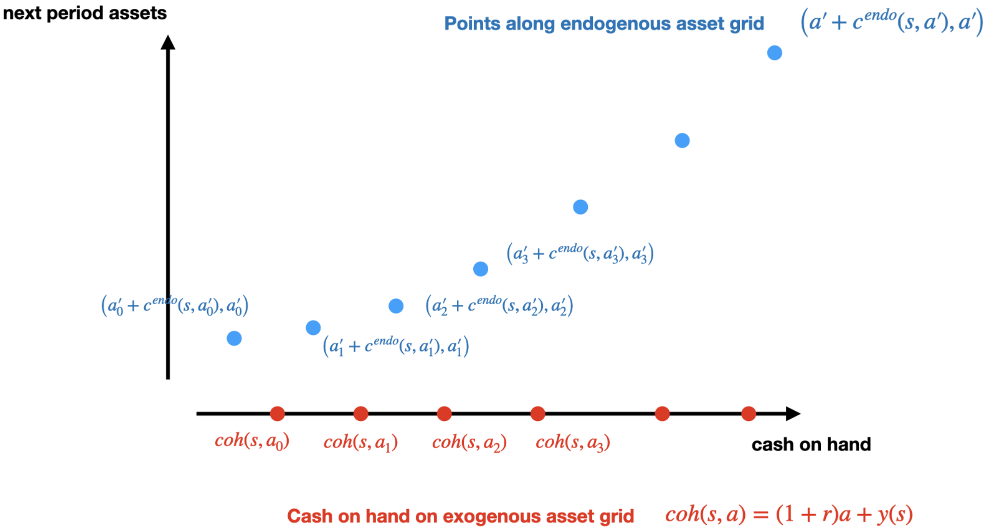
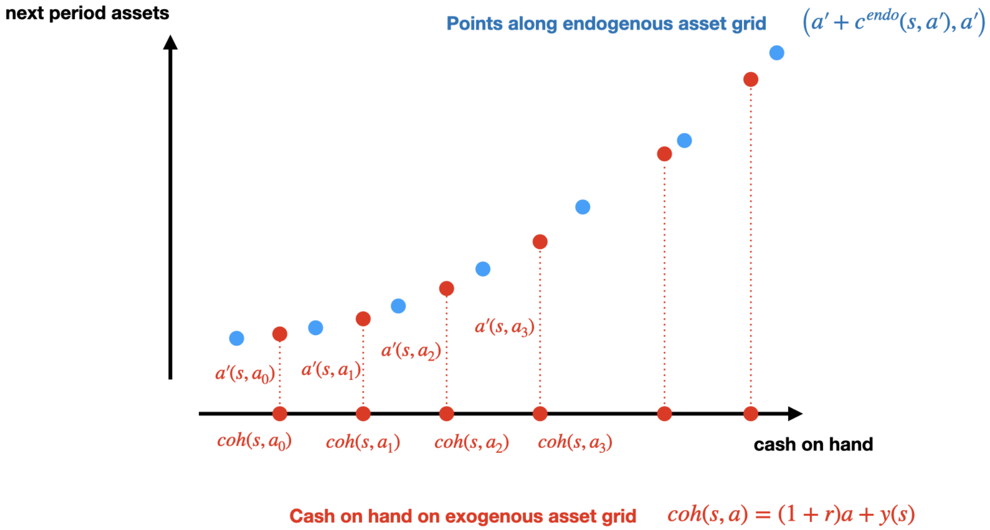


Illustration of key EGM step



- Recall our steps:

$$V_{a,t+1}(s', a') \overset{\text{FOC}}{\rightarrow} c_t^{endo}(s, a') \overset{\text{interpolate}}{\rightarrow} a_t'(s, a) \overset{\text{budget}}{\rightarrow} c_t(s, a) \overset{\text{Envelope}}{\rightarrow} V_{a,t}(s, a)$$

- We just found $a_t'(s, a)$, but that ignored the borrowing constraint.

Step 2: Steady state policies: Details of EGM (backward iteration)

- Recall our steps:

$$V_{a,t+1}(s',a') \overset{\text{FOC}}{\to} c_t^{endo}(s,a') \overset{\text{interpolate}}{\to} a_t'(s,a) \overset{\text{budget}}{\to} c_t(s,a) \overset{\text{Envelope}}{\to} V_{a,t}(s,a)$$

- We just found $a_t'(s,a)$, but that ignored the borrowing constraint.
    - Solution: just enforce it at this point, ie set $a_t'(s,a) \geq \underline{a}$

Step 2: Steady state policies: Details of EGM (backward iteration)

- Recall our steps:

$$V_{a,t+1}(s',a') \overset{\text{FOC}}{\to} c_t^{endo}(s,a') \overset{\text{interpolate}}{\to} a_t'(s,a) \overset{\text{budget}}{\to} c_t(s,a) \overset{\text{Envelope}}{\to} V_{a,t}(s,a)$$

- We just found $a_t'(s,a)$, but that ignored the borrowing constraint.
  - Solution: just enforce it at this point, ie set $a_t'(s,a) \geq \underline{a}$
- Then compute $c_t(s,a)$ from $\text{coh}_t(s,a) - a_t'(s,a)$.

Step 2: Steady state policies: Details of EGM (backward iteration)

- Recall our steps:

$$V_{a,t+1}(s',a') \overset{\text{FOC}}{\to} c_t^{endo}(s,a') \overset{\text{interpolate}}{\longrightarrow} a'_t(s,a) \overset{\text{budget}}{\longrightarrow} c_t(s,a) \overset{\text{Envelope}}{\longrightarrow} V_{a,t}(s,a)$$

- We just found $a'_t(s,a)$, but that ignored the borrowing constraint.
  - Solution: just enforce it at this point, ie set $a'_t(s,a) \geq \underline{a}$
- Then compute $c_t(s,a)$ from $\text{coh}_t(s,a) - a'_t(s,a)$.
- Then use (1) for $V_{a,t}(s,a)$. Done!

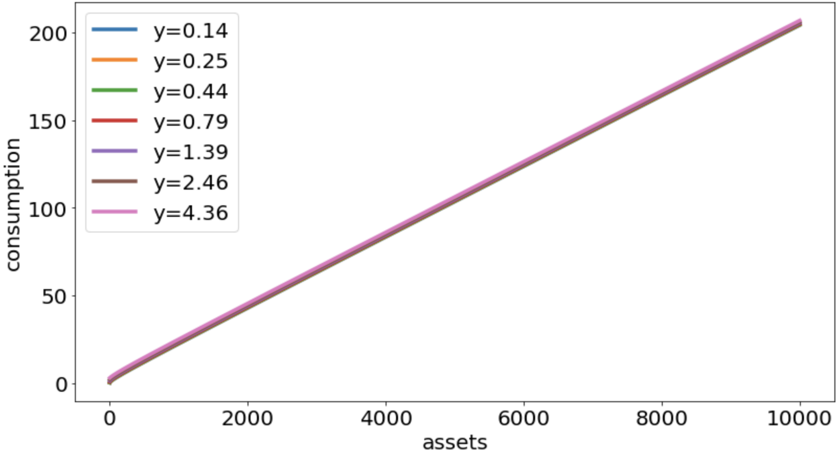
Step 2: Steady state policies: Details of EGM (backward iteration)

- Recall our steps:

$$V_{a,t+1}(s',a') \overset{\text{FOC}}{\rightarrow} c_t^{endo}(s,a') \overset{\text{interpolate}}{\rightarrow} a'_t(s,a) \overset{\text{budget}}{\rightarrow} c_t(s,a) \overset{\text{Envelope}}{\rightarrow} V_{a,t}(s,a)$$

- We just found $a'_t(s,a)$, but that ignored the borrowing constraint.
  - Solution: just enforce it at this point, ie set $a'_t(s,a) \geq \underline{a}$
- Then compute $c_t(s,a)$ from $\text{coh}_t(s,a) - a'_t(s,a)$.
- Then use (1) for $V_{a,t}(s,a)$. Done!
- In the steady state: no $t \rightarrow$ just iterate until policies converge.

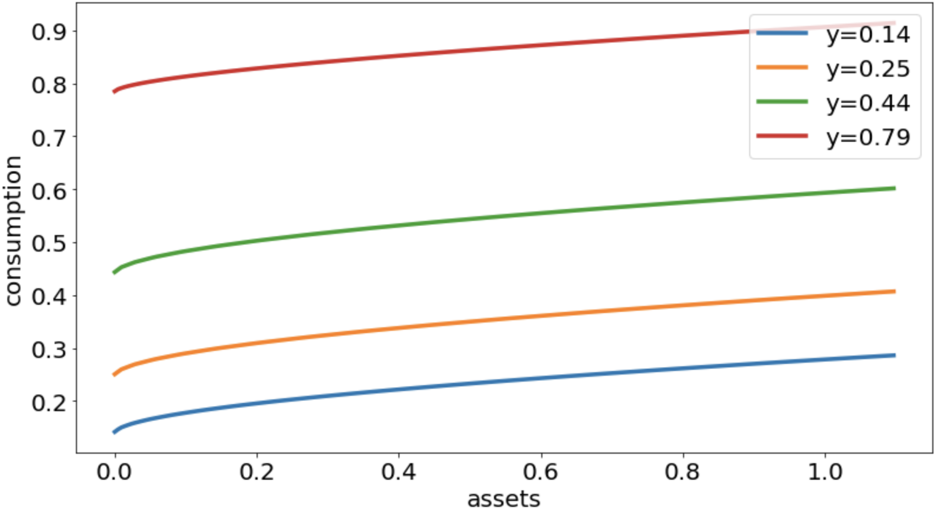
Step 2: Illustration



[Parameters: $\rho = 0.75, \sigma_y = 0.7, n_y = 7, r = 0.0025, \beta = 0.98, u = \log c$]

Step 2: Zooming in to see the nonlinearity

[Next Dist]



Step 3: Distribution (forward iteration)

Assume we have histogram (pdf) $D_t(s, a)$ over grid points for (s, a). (2-d array.)

Idea: use $a_t'(s, a)$ policy to send the mass on (s, a) to $(s', a_t'(s, a))$. Any issues?

Step 3: Distribution (forward iteration)

Assume we have histogram (pdf) $D_t(s, a)$ over grid points for (s, a). (2-d array.)

Idea: use $a_t'(s, a)$ policy to send the mass on (s, a) to $(s', a_t'(s, a))$. Any issues?

- $a_t'(s, a)$ often falls in between grid points!

Assume we have histogram (pdf) $D_t(s,a)$ over grid points for (s,a). (2-d array.)

Idea: use $a_t'(s,a)$ policy to send the mass on (s,a) to $(s', a_t'(s,a))$. Any issues?

- $a_t'(s,a)$ often falls in between grid points!

Solution: use lotteries! E.g. if for two grid points a_i, a_{i+1} we have

$$a_t'(s,a) = \pi a_i + (1-\pi)\, a_{i+1}$$

Then send mass $\pi D_t(s,a)$ to (s', a_i) and $(1-\pi)\, D_t(s,a)$ to (s', a_{i+1}).

Step 3: Distribution (forward iteration)

Assume we have histogram (pdf) $D_t(s, a)$ over grid points for (s, a). (2-d array.)

Idea: use $a'_t(s, a)$ policy to send the mass on (s, a) to $(s', a'_t(s, a))$. Any issues?

- $a'_t(s, a)$ often falls in between grid points!

Solution: use lotteries! E.g. if for two grid points a_i, a_{i+1} we have

$$a'_t(s, a) = \pi a_i + (1 - \pi)\, a_{i+1}$$

Then send mass $\pi D_t(s, a)$ to (s', a_i) and $(1 - \pi)\, D_t(s, a)$ to (s', a_{i+1}).

- Often called “Young’s method”. Note: this is “non-stochastic simulation”

Step 3: Distribution (forward iteration)

Assume we have histogram (pdf) $D_t(s,a)$ over grid points for (s,a). (2-d array.)

Idea: use $a'_t(s,a)$ policy to send the mass on (s,a) to $(s', a'_t(s,a))$. Any issues?

- $a'_t(s,a)$ often falls in between grid points!

Solution: use lotteries! E.g. if for two grid points a_i, a_{i+1} we have

$$a'_t(s,a) = \pi a_i + (1-\pi)\, a_{i+1}$$

Then send mass $\pi D_t(s,a)$ to (s', a_i) and $(1-\pi)\, D_t(s,a)$ to (s', a_{i+1}).

- Often called "Young's method". Note: this is "non-stochastic simulation"
- Golden rule in this class: **avoid actual simulation if you can!**

Step 3: Distribution (forward iteration)

Assume we have histogram (pdf) $D_t(s, a)$ over grid points for (s, a). (2-d array.)

Idea: use $a'_t(s, a)$ policy to send the mass on (s, a) to $(s', a'_t(s, a))$. Any issues?

- $a'_t(s, a)$ often falls in between grid points!

Solution: use lotteries! E.g. if for two grid points a_i, a_{i+1} we have

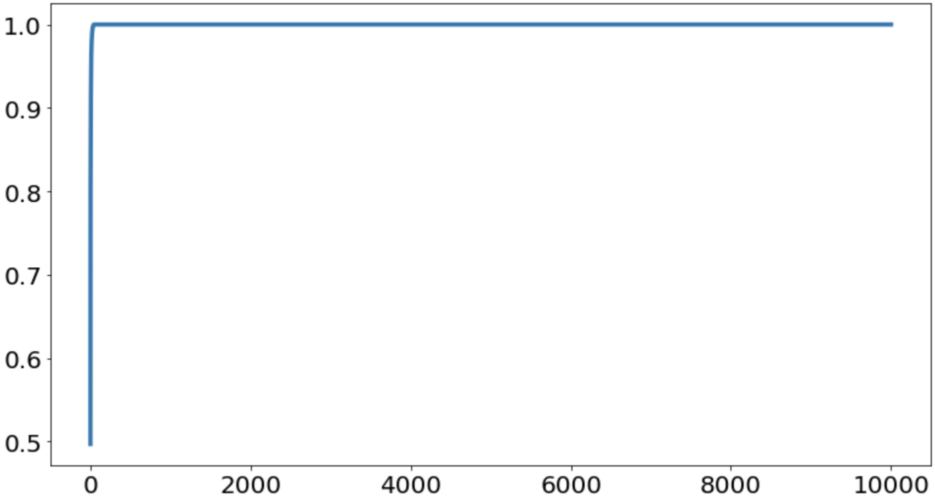
$$a'_t(s, a) = \pi a_i + (1 - \pi)\, a_{i+1}$$

Then send mass $\pi D_t(s, a)$ to (s', a_i) and $(1 - \pi)\, D_t(s, a)$ to (s', a_{i+1}).

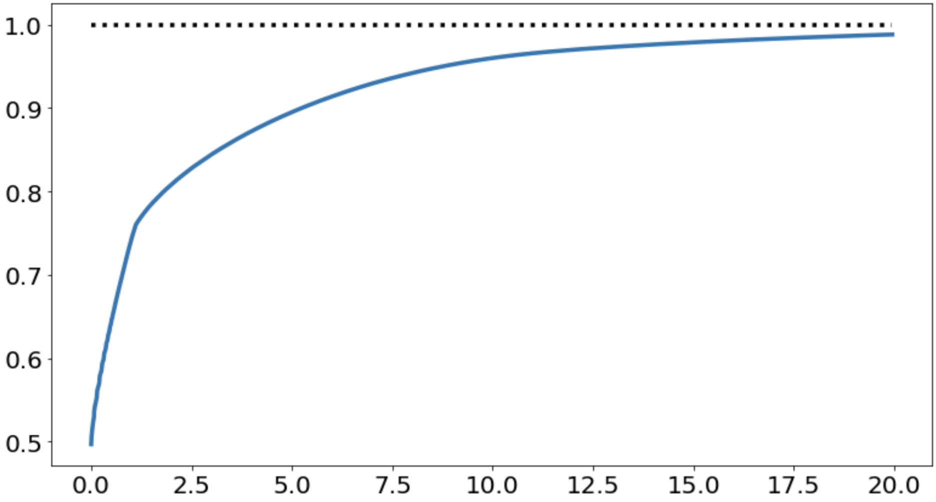
- Often called "Young's method". Note: this is "non-stochastic simulation"
- Golden rule in this class: **avoid actual simulation if you can!**

Steady state: no t! Iterate to convergence...

Step 3: Illustrating the marginal asset distribution

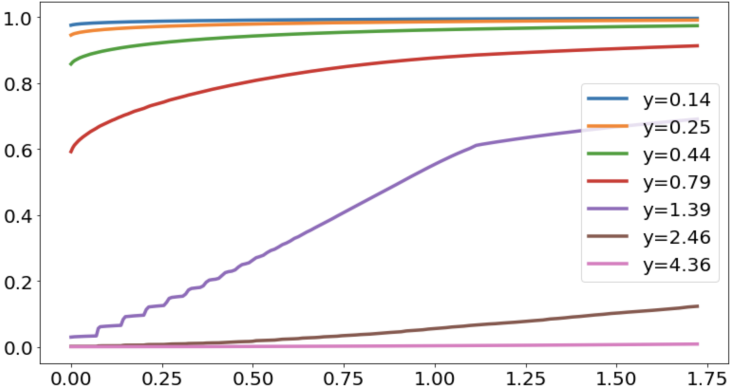


Step 3: Zooming in



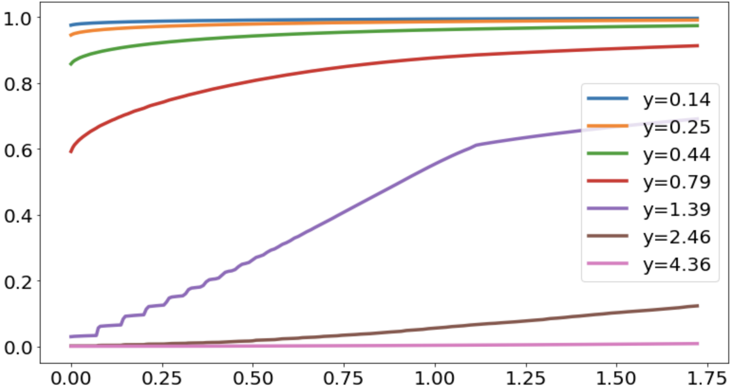
Step 3: Zooming in even more, by income

[Next Agg]



Step 3: Zooming in even more, by income

[Next Agg]



Wiggles: mass slowly moving away from borrowing constraint

Kink: target saving point for buffer stock saving

True distribution has these features, even with continuous distribution & continuous income process.

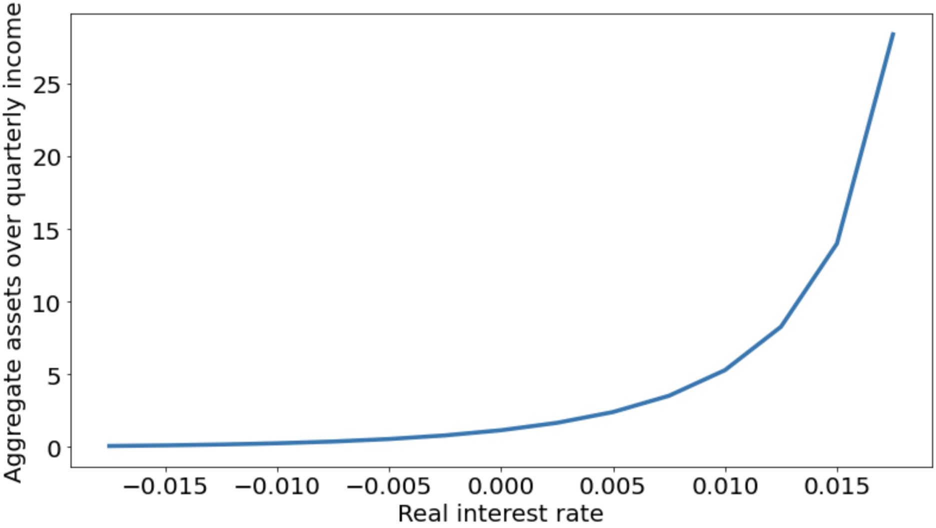
Aggregate consumption and assets

$$C_t = \sum_{s,a} c_t(s,a) \cdot D_t(s,a)$$

$$A_t = \sum_{s,a} a'_t(s,a) \cdot D_t(s,a)$$

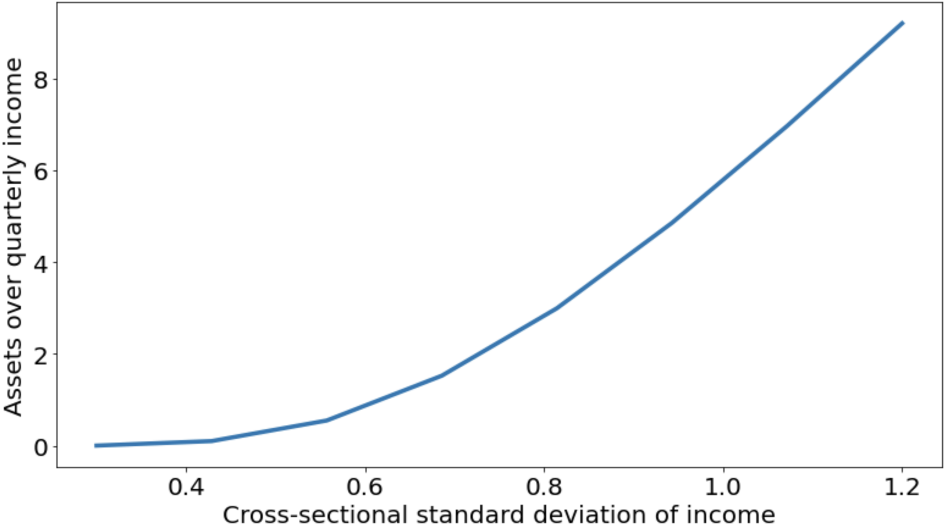
Again: get these via simple dot product of vectors. Do not simulate.

Step 4: Steady state comparative statics



Step 4: Steady state comparative statics

[Next PE dyn]



Dynamics

Two kinds of dynamics ...

Partial equilibrium (PE): How do households respond to shock in interest rates or income

General equilibrium (GE): How do we find response of endogenous price (e.g. r_t) in response to shock (e.g. Y_t)

PE now, GE later!

Partial equilibrium dynamics

Suppose households see time-varying $\{r_t\}$ or $\{y_t(s)\}$. What is C_t, A_t?

Partial equilibrium dynamics

Suppose households see time-varying $\{r_t\}$ or $\{y_t(s)\}$. What is C_t, A_t?

Our existing backward & forward iteration routines work:

- From $V_{aT} = V_{a,ss}$, iterate backward$\rightarrow$ policies $a_t(s,a), c_t(s,a)$ for all $t \geq 0$.
- From $D_0(s,a) = D_{ss}(s,a)$, iterate forward using $a_t(s,a) \rightarrow D_t(s,a)$ for all $t \geq 1$.
- Compute $A_t = \sum_{s,a} a_t(s,a) D_t(s,a)$, $C_t = \sum_{s,a} c_t(s,a) D_t(s,a)$ as before!

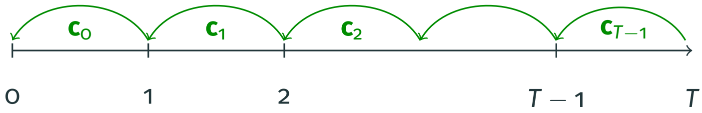
Partial equilibrium dynamics

Suppose households see time-varying $\{r_t\}$ or $\{y_t(s)\}$. What is C_t, A_t?

Our existing backward & forward iteration routines work:

- From $V_{aT} = V_{a,ss}$, iterate backward$\rightarrow$ policies $a_t(s,a), c_t(s,a)$ for all $t \geq 0$.
- From $D_0(s,a) = D_{ss}(s,a)$, iterate forward using $a_t(s,a) \rightarrow D_t(s,a)$ for all $t \geq 1$.
- Compute $A_t = \sum_{s,a} a_t(s,a)D_t(s,a)$, $C_t = \sum_{s,a} c_t(s,a)D_t(s,a)$ as before!

Visualize this:



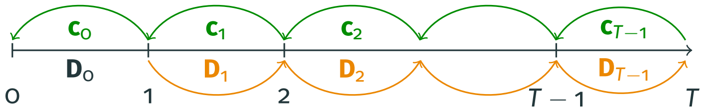
Partial equilibrium dynamics

Suppose households see time-varying $\{r_t\}$ or $\{y_t(s)\}$. What is C_t, A_t?

Our existing backward & forward iteration routines work:

- From $V_{aT} = V_{a,ss}$, iterate backward$\to$ policies $a_t(s,a), c_t(s,a)$ for all $t \geq 0$.
- From $D_0(s,a) = D_{ss}(s,a)$, iterate forward using $a_t(s,a) \to D_t(s,a)$ for all $t \geq 1$.
- Compute $A_t = \sum_{s,a} a_t(s,a)D_t(s,a)$, $C_t = \sum_{s,a} c_t(s,a)D_t(s,a)$ as before!

Visualize this:



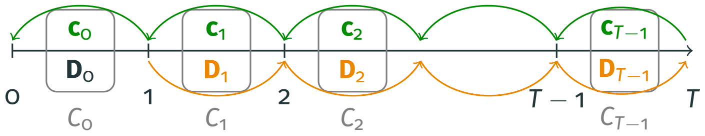
Partial equilibrium dynamics

Suppose households see time-varying $\{r_t\}$ or $\{y_t(s)\}$. What is C_t, A_t?

Our existing backward & forward iteration routines work:

- From $V_{aT} = V_{a,ss}$, iterate backward$\rightarrow$ policies $a_t(s,a), c_t(s,a)$ for all $t \geq 0$.
- From $D_0(s,a) = D_{ss}(s,a)$, iterate forward using $a_t(s,a) \rightarrow D_t(s,a)$ for all $t \geq 1$.
- Compute $A_t = \sum_{s,a} a_t(s,a) D_t(s,a)$, $C_t = \sum_{s,a} c_t(s,a) D_t(s,a)$ as before!

Visualize this:

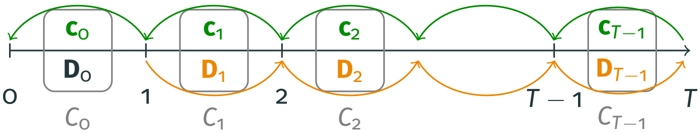


Suppose households see time-varying $\{r_t\}$ or $\{y_t(s)\}$. What is C_t, A_t?

Our existing backward & forward iteration routines work:

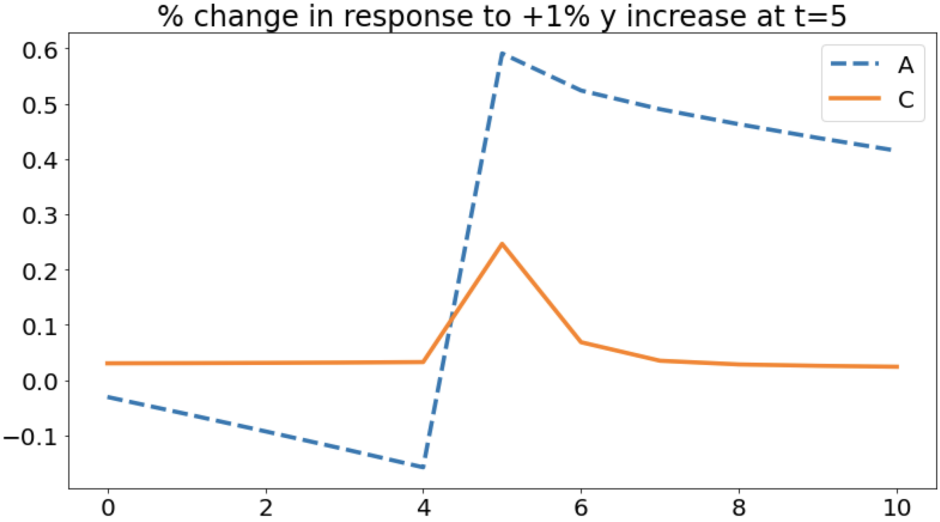
- From $V_{aT} = V_{a,ss}$, iterate backward$\rightarrow$ policies $a_t(s,a), c_t(s,a)$ for all $t \geq 0$.
- From $D_0(s,a) = D_{ss}(s,a)$, iterate forward using $a_t(s,a) \rightarrow D_t(s,a)$ for all $t \geq 1$.
- Compute $A_t = \sum_{s,a} a_t(s,a) D_t(s,a)$, $C_t = \sum_{s,a} c_t(s,a) D_t(s,a)$ as before!

Visualize this:

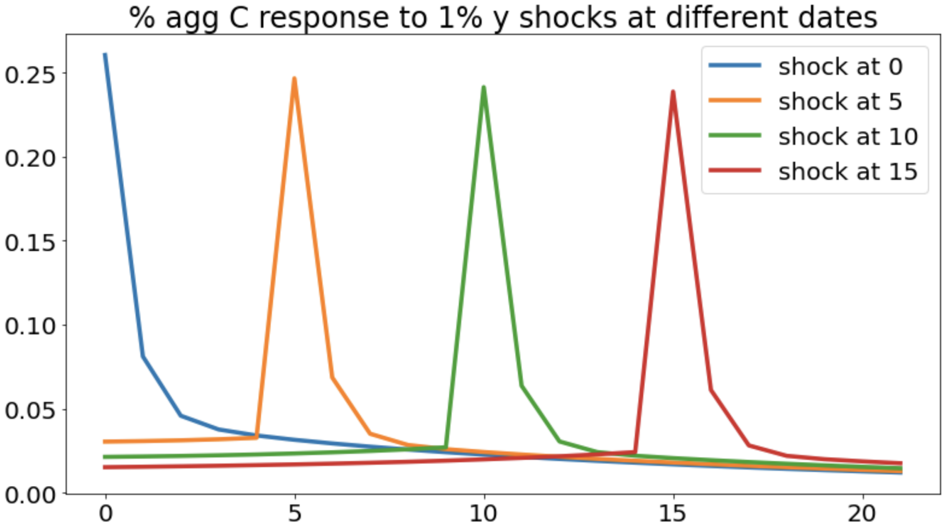


Let's look at an example ...

One time shock to income at date 5



Shocks at many dates s



Denote the impulse response to a one time shock at date s by $\frac{\partial C_t}{\partial y_s}$.

We can stack this into a matrix

$$\mathcal{J}^{C,y} \equiv \left(\frac{\partial C_t}{\partial y_s}\right)$$

Each column corresponds to the impulse response of C for a different s.

Denote the impulse response to a one time shock at date s by $\frac{\partial C_t}{\partial y_s}$.

We can stack this into a matrix

$$\mathcal{J}^{C,y} \equiv \left(\frac{\partial C_t}{\partial y_s} \right)$$

Each column corresponds to the impulse response of C for a different s.

We will call $\mathcal{J}^{C,y}$ **(sequence-space) Jacobians.** Those will turn out to be extremely useful objects for *general equilibrium* impulse responses!

We'll discuss why in the next class.

How do we compute sequence-space Jacobians?

To compute Jacobians, we have to truncate to some finite time T, say $T = 300$.

To compute T columns, we need to go backward $T \times T$ times and forward $T \times T$ times. That can be slow...

To compute Jacobians, we have to truncate to some finite time T, say $T = 300$.

To compute T columns, we need to go backward $T \times T$ times and forward $T \times T$ times. That can be slow...

Next: a method that can be used to compute the entire matrix in only T backward and T "forward iterations"! (Auclert et al., 2021)

Computing $\mathcal{J}$: Only T backward iterations

Denote by $c_t^s(s,a), a_t^s(s,a)$ the policies at date t in response to s date shock.

Why is a single backward iteration enough?

Key idea 1: Only the **time** $s-t$ **until the perturbation matters:**

$$c_t^s(s,a) = \begin{cases} c_{ss}(s,a) & s < t \\ c_{T-1-(s-t)}^{T-1}(s,a) & s \geq t \end{cases}$$

Thus, only need a single backward iteration with $s = T-1$ to get all the $c_t^s(s,a)$!

Computing $\mathcal{J}$: Only T backward iterations

Denote by $c_t^s(s,a), a_t^s(s,a)$ the policies at date t in response to s date shock.

Why is a single backward iteration enough?

Key idea 1: Only the **time** $s-t$ **until the perturbation matters:**

$$c_t^s(s,a) = \begin{cases} c_{ss}(s,a) & s < t \\ c_{T-1-(s-t)}^{T-1}(s,a) & s \geq t \end{cases}$$

Thus, only need a single backward iteration with $s = T-1$ to get all the $c_t^s(s,a)$!

From this we get:

- $C_0^s = \sum_{s,a} c_0^s(s,a) D_{ss}(s,a)$, so first row of Jacobian $\mathcal{J}_{0s} = \frac{\partial C_0}{\partial y_s}$
- $D_1^s(s,a)$, distribution at date 1 implied by new policy $c_0^s(s,a)$ at date 0

Computing $\mathcal{J}$: First column of Jacobian

Could keep going and compute $D_t^s(s, a)$, then $C_t^s = \mathbf{c}_t^{s\prime} \cdot \mathbf{D}_t^s$, for all s, t

But that's still time consuming. Can we do better? Yes!

Computing $\mathcal{J}$: First column of Jacobian

Could keep going and compute $D_t^s(s,a)$, then $C_t^s = \mathbf{c}_t^{s\prime} \cdot \mathbf{D}_t^s$, for all s,t

But that's still time consuming. Can we do better? Yes!

Why? Observe that to first order,

$$dC_t^s = d\mathbf{c}_t^{s\prime} \cdot \mathbf{D}_{ss} + \mathbf{c}_{ss}' \cdot d\mathbf{D}_t^s$$

Computing $\mathcal{J}$: First column of Jacobian

Could keep going and compute $D_t^s(s,a)$, then $C_t^s = \mathbf{c}_t^{s\prime} \cdot \mathbf{D}_t^s$, for all s, t

But that's still time consuming. Can we do better? Yes!

Why? Observe that to first order,

$$dC_t^s = d\mathbf{c}_t^{s\prime} \cdot \mathbf{D}_{ss} + \mathbf{c}_{ss}' \cdot d\mathbf{D}_t^s$$

Consider first $s = 0$. For $t \geq 1$, only second term remains, with

$$d\mathbf{D}_t^0 = \left(\Lambda_{ss}'\right)^t d\mathbf{D}_1^0$$

where $\Lambda_{ss} =$ s.s. transition matrix for states (s, a) (Why?).

Computing $\mathcal{J}$: First column of Jacobian

Could keep going and compute $D_t^s(s, a)$, then $C_t^s = \mathbf{c}_t^{s\prime} \cdot \mathbf{D}_t^s$, for all s, t

But that's still time consuming. Can we do better? Yes!

Why? Observe that to first order,

$$dC_t^s = d\mathbf{c}_t^{s\prime} \cdot \mathbf{D}_{ss} + \mathbf{c}_{ss}' \cdot d\mathbf{D}_t^s$$

Consider first $s = 0$. For $t \geq 1$, only second term remains, with

$$d\mathbf{D}_t^0 = \left(\Lambda_{ss}'\right)^t d\mathbf{D}_1^0$$

where Λ_{ss} = s.s. transition matrix for states (s, a) (Why?). So:

$$dC_t^0 = \mathbf{c}_{ss}' \cdot \left(\Lambda_{ss}'\right)^t d\mathbf{D}_1^0 \tag{3}$$

This gives us the first column of the Jacobian, $\mathcal{J}_{t,0}$. What about $s > 0$?

Computing $\mathcal{J}$: Other columns of Jacobian

Again, look at

$$dC_t^s = d\mathbf{c}_t^{s\prime} \cdot \mathbf{D}_{ss} + \mathbf{c}_{ss}' \cdot d\mathbf{D}_t^s$$

For $s > 0$, let's use our policy symmetry result. Since $d\mathbf{c}_t^s = d\mathbf{c}_{t-1}^{s-1}$, we have:

$$dC_t^s - dC_{t-1}^{s-1} = \mathbf{c}_{ss}' \cdot \left(d\mathbf{D}_t^s - d\mathbf{D}_{t-1}^{s-1}\right)$$

Computing $\mathcal{J}$: Other columns of Jacobian

Again, look at

$$dC_t^s = d\mathbf{c}_t^{s\prime} \cdot \mathbf{D}_{ss} + \mathbf{c}_{ss}' \cdot d\mathbf{D}_t^s$$

For $s > 0$, let's use our policy symmetry result. Since $d\mathbf{c}_t^s = d\mathbf{c}_{t-1}^{s-1}$, we have:

$$dC_t^s - dC_{t-1}^{s-1} = \mathbf{c}_{ss}' \cdot \left(d\mathbf{D}_t^s - d\mathbf{D}_{t-1}^{s-1}\right)$$

Let's call this $\mathcal{F}_{t,s}dx$. Using policy symmetry again, it is simple to show:

$$\mathcal{F}_{t,s}dx = d\mathbf{c}_{ss}' \cdot \left(\Lambda_{ss}'\right)^t d\mathbf{D}_1^s \qquad \forall t, s \tag{4}$$

This looks exactly like (3)!

Computing $\mathcal{J}$: Other columns of Jacobian

Again, look at

$$dC_t^s = d\mathbf{c}_t^{s\prime} \cdot \mathbf{D}_{ss} + \mathbf{c}_{ss}' \cdot d\mathbf{D}_t^s$$

For $s > 0$, let's use our policy symmetry result. Since $d\mathbf{c}_t^s = d\mathbf{c}_{t-1}^{s-1}$, we have:

$$dC_t^s - dC_{t-1}^{s-1} = \mathbf{c}_{ss}' \cdot \left(d\mathbf{D}_t^s - d\mathbf{D}_{t-1}^{s-1}\right)$$

Let's call this $\mathcal{F}_{t,s}dx$. Using policy symmetry again, it is simple to show:

$$\mathcal{F}_{t,s}dx = d\mathbf{c}_{ss}' \cdot \left(\Lambda_{ss}'\right)^t d\mathbf{D}_1^s \qquad \forall t, s \tag{4}$$

This looks exactly like (3)!

What's the meaning of $\mathcal{F}_{t,s}$? It's a "fake news shock"

- at $t = 0$: news shock that period $s > 0$ shock hits $\rightarrow$ distribution $d\mathbf{D}_1^s$
- at $t = 1$: news shock that there *won't* be a shock at $s \rightarrow$ use s.s. policies

Computing $\mathcal{J}$: Fake news matrix 1/2

Recall we want to know $\mathcal{J}_{t,s} \equiv \frac{\partial C_t}{\partial y_s}$ for $s, t \in \{0, \ldots, T-1\}$.

Can think of $\mathcal{J}$ as a **news matrix:**

- column $s =$ response to news that shock is going to hit in period s

Computing $\mathcal{J}$: Fake news matrix 1/2

Recall we want to know $\mathcal{J}_{t,s} \equiv \frac{\partial C_t}{\partial y_s}$ for $s, t \in \{0, \ldots, T-1\}$.

Can think of $\mathcal{J}$ as a **news matrix:**

- column s = response to news that shock is going to hit in period s

We just saw how to get:

- the first row $\mathcal{J}_{0,s}$
- the first column $\mathcal{J}_{t,0}$
- differences in columns $\mathcal{F}_{t,s} \equiv \mathcal{J}_{t,s} - \mathcal{J}_{t-1,s-1}$ for all $s, t \geq 1$

Key idea 2: can recover $\mathcal{J}$ from first row, first column, and **"fake news" matrix** $\mathcal{F}$.

- News shock = sequence of fake news shocks

Computing $\mathcal{J}$: Fake news matrix 2/2

Expand $\mathcal{F}$ to include first row and column of $\mathcal{J}$:

$$\mathcal{F} = \begin{pmatrix} \mathcal{J}_{00} & \mathcal{J}_{01} & \mathcal{J}_{02} & \cdots \\ \mathcal{J}_{10} & \mathcal{J}_{11} - \mathcal{J}_{00} & \mathcal{J}_{12} - \mathcal{J}_{01} & \cdots \\ \mathcal{J}_{20} & \mathcal{J}_{12} - \mathcal{J}_{10} & \mathcal{J}_{22} - \mathcal{J}_{11} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \qquad \mathcal{J} = \begin{pmatrix} \mathcal{J}_{00} & \mathcal{J}_{01} & \mathcal{J}_{02} & \cdots \\ \mathcal{J}_{10} & \mathcal{J}_{11} & \mathcal{J}_{12} & \cdots \\ \mathcal{J}_{20} & \mathcal{J}_{12} & \mathcal{J}_{22} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

- can recover $\mathcal{J}$ from $\mathcal{F}$ by adding elements from top left diagonal

Expand $\mathcal{F}$ to include first row and column of $\mathcal{J}$:

$$\mathcal{F} = \begin{pmatrix} \mathcal{J}_{00} & \mathcal{J}_{01} & \mathcal{J}_{02} & \cdots \\ \mathcal{J}_{10} & \mathcal{J}_{11} - \mathcal{J}_{00} & \mathcal{J}_{12} - \mathcal{J}_{01} & \cdots \\ \mathcal{J}_{20} & \mathcal{J}_{12} - \mathcal{J}_{10} & \mathcal{J}_{22} - \mathcal{J}_{11} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \qquad \mathcal{J} = \begin{pmatrix} \mathcal{J}_{00} & \mathcal{J}_{01} & \mathcal{J}_{02} & \cdots \\ \mathcal{J}_{10} & \mathcal{J}_{11} & \mathcal{J}_{12} & \cdots \\ \mathcal{J}_{20} & \mathcal{J}_{12} & \mathcal{J}_{22} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

- can recover $\mathcal{J}$ from $\mathcal{F}$ by adding elements from top left diagonal
- ie, the formula implementing Key idea 2 is:

$$\mathcal{J}_{t,s} = \sum_{k=0}^{\min\{t,s\}} \mathcal{F}_{t-k,s-k} \tag{5}$$

Define **expectations function**

$$\mathcal{E}_t(s,a) \equiv \mathbb{E}\left[c(s_t,a_t)|s_0 = s, a_0 = a\right]$$

$=$ expected consumption in t periods for an agent in state (s,a) today.

Define **expectations function**

$$\mathcal{E}_t(s,a) \equiv \mathbb{E}\left[c(s_t,a_t) | s_0 = s, a_0 = a\right]$$

$=$ expected consumption in t periods for an agent in state (s,a) today.

Expectations function is easy to compute: Stack as vector across (s,a), ...

$$\boldsymbol{\mathcal{E}}_0 = \mathbf{c}_{ss} \qquad \boldsymbol{\mathcal{E}}_t = \Lambda_{ss}\boldsymbol{\mathcal{E}}_{t-1}$$

Practical implementation: obtaining $\mathcal{J}_{t0}$ and $\mathcal{F}_{t,s}$

Define **expectations function**

$$\mathcal{E}_t(s,a) \equiv \mathbb{E}\left[c(s_t,a_t)|s_0=s,a_0=a\right]$$

$=$ expected consumption in t periods for an agent in state (s,a) today.

Expectations function is easy to compute: Stack as vector across (s,a), ...

$$\boldsymbol{\mathcal{E}}_0 = \mathbf{c}_{ss} \qquad \boldsymbol{\mathcal{E}}_t = \Lambda_{ss}\boldsymbol{\mathcal{E}}_{t-1}$$

Key idea 3: $\boldsymbol{\mathcal{E}}_t$'s are enough to get all rows $t \geq 1$ of $\mathcal{F}$

- First column $s=0$: use (3), $\mathcal{J}_{t,0}dx = (\boldsymbol{\mathcal{E}}_t)'\, d\mathbf{D}_1^0$
- Other columns: use (4), $\mathcal{F}_{t,s}dx = (\boldsymbol{\mathcal{E}}_t)'\, d\mathbf{D}_1^s$

[Recall that we got the first row as well as $d\mathbf{D}_1^0$ from key idea 1]

Wrapping up

- **Fake news algorithm** for obtaining Jacobian $\mathcal{J}$ of aggregate (eg, C) to input of a heterogeneous-agent problem (eg, Y) relies on following steps:

1. Get all policies $\mathbf{c}_t^s$ and date-0 perturbation $d\mathbf{D}_1^0$ to distn with one backward iteration
2. Calculate steady state expectation vectors $\mathcal{E}_t$
3. Build the Fake news matrix $\mathcal{F}_{t,s}$ using info from steps 1 and 2
4. Build Jacobian $\mathcal{J}$ from $\mathcal{F}$ using equation (5)

Conclusion

Taking stock

- Introduced the “standard incomplete markets model”
- Computed steady state
- Computed partial equilibrium impulse responses
- Collected impulse responses to one-time “unit” shocks in Jacobians
- Discussed “fake news algorithm” to compute those Jacobians very quickly

References

Auclert, A., Bardóczy, B., Rognlie, M., and Straub, L. (2021). Using the Sequence-Space Jacobian to Solve and Estimate Heterogeneous-Agent Models. *Econometrica*, 89(5):2375–2408.