Using the Sequence-Space Jacobian to Solve and Estimate Heterogeneous-Agent Models

by Auclert, Bardóczy, Rognlie, and Straub

presented by Wendy Morrison

March 2021

Motivation

"A rapidly expanding literature at the frontier of macroeconomics incorporates rich heterogeneity into dynamic general equilibrium models. A central challenge in this literature is that equilibrium involves the time-varying, high-dimensional distribution of agents over their state variables"

Plan for today:

- Why is incorporating 'rich heterogeneity' interesting and important?
- Why can't we just use standard methods to solve models like these? What methods are currently available?
- What is the Sequence Space Jacobian method, and how do I go about actually implementing it?

Additional Sources: McKay (https://alisdairmckay.com/Notes/HetAgents/index.html) ; Fernandez-Villaverde, Rubio-Ramirez, Schorfheide (Handbook of Macro)

Rich heterogeneity

By rich heterogeneity we mean models that replace a single representative household (or firm) with a continuous distribution of households (or firms).

These households may or may not be *ex ante* heterogeneous along some other dimensions, but the primary source of heterogeneity is an idiosyncratic shock (say to labor productivity or health).

When you add in incomplete markets (typically a borrowing constraint), this prevents agents from fully insuring themselves against idiosyncratic risk, and leads to a rich distribution, not just over productivity and wealth, but also consumption behavior.

You can calibrate the shock process such that the steady state distribution targets real world distributions.

Why this matters

Heterogeneity will be important when studying any macroeconomic trend or policy that has *distributional* consequences.

- Effect of lower nominal rates on borrowers vs. savers (Auclert, 2019)
- The effect of targeted transfers from rich and healthy to poor and sick (Oh and Reis, 2012)

These matter both for welfare and because MPCs vary across distribution.

- If consumption has diminishing marginal returns, redistribution could be welfare improving
- If poor have higher MPCs than rich, redistribution can amplify aggregate demand in the short run

Why rich heterogeneity matters

Difficult to calibrate distributions to match real world without it.

When heterogeneity is 'rich' instead of simple (e.g. 2 agent models), households engage in precautionary savings behavior. In other words, every household believes there is some change (even if it's remote) that they'll hit the borrowing constraint.

- This makes consumption for agents much less directly responsive to interest rate changes, changing the way monetary policy works (Kaplan, Moll, and Violante, 2018)
- It also tempers the direct impact of *future* interest rate changes, making forward guidance less potent (McKay, Nakamura, and Steinsson, 2016)

So why can't we use standard tools?

Solving basic RA models

Consider a basic RBC model with just a representative household and a firm. Going forward the notation $F(x, y) = 0$ corresponds to the equilibrium conditions where y are the controls (C_t) and x are the states (K_{t-1} and Z_t).:

$$C_t^{-\gamma} = \beta E_t[R_{t+1} C_{t+1}^{-\gamma}]$$
$$R_t = \alpha Z_t (K_{t-1}/L)^{\alpha-1} + 1 - \delta$$
$$K_t = (1-\delta)K_{t-1} + Y_t - C_t$$
$$Y_t = Z_t K_{t-1}^{\alpha} L^{1-\alpha}$$
$$log(Z_t) = \rho log(Z_{t-1}) + \epsilon_t$$

We could easily plug a model like this into Dynare and get impulse response functions for output, capital, and consumption to changes in productivity. Review: what is Dynare actually doing?

Perturbation Around Steady State

The solution to this model consists of decision rules for the controls:

$$y = g(x, \sigma_\epsilon)$$

As well as a law of motion for the states:

$$x' = h(x, \sigma_\epsilon) + \eta\epsilon$$

In this context this will look like:

$$\begin{aligned} C_t &= g(K_{t-1}, Z_t) \\ K_t &= (1-\delta)K_{t-1} + Z_t K_{t-1}^{\alpha} L^{1-\alpha} - g(K_{t-1}, Z_t) \\ log(Z_t) &= \rho log(Z_{t-1}) + \epsilon_t \end{aligned}$$

So $\eta = [0 \;\; 1]'$

Perturbation Around Steady State

(1) Find the deterministic steady state (where $\epsilon = 1$ and $\sigma_\epsilon = 0$)

(2) Use a 1st order, linear Taylor approximation of g and h around the steady state, $(\bar{y}, \bar{x})$.

$$g(x, \sigma_\epsilon) = \bar{y} + g_x(\bar{x}, 0)(x - \bar{x})'$$
$$h(x, \sigma_\epsilon) = \bar{x} + h_x(\bar{x}, 0)(x - \bar{x})'$$

(3) Find g_x and h_x by writing $F(y, x)$ in terms of h and g, then exploiting the fact that $F(y, x) = 0$ implies that $F_x(g(x), x) = 0$. Note that this means we are expressing the system F in **state space**.

Adding heterogeneity

Suppose instead we considered the same model, except now the economy had a *continuum* of households (mass 1), each with their own unique exogenous stochastic labor market productivity (e).

$$k_t + c_t = R_t k_{t-1} + W_t e_t$$
$$R_t = \alpha Z_t (K_{t-1}/L)^{\alpha-1} + 1 - \delta$$
$$W_t = (1-\alpha) Z_t (K_{t-1}/L)^{\alpha}$$

If $\Gamma(k_{t-1}, e_t)$ is the joint distribution over *individual* states, then we can write:

$$K_{t-1} = \int_0^1 k_{t-1} d\Gamma(k_{t-1}, e_t)$$
$$L = \int_0^1 e_t d\Gamma(k_{t-1}, e_t) = 1$$

What's the issue here?

In the first model, our decision rule $y = g(x, \sigma_\epsilon)$ was $C_t = g(K_{t-1}, Z_t, \sigma_\epsilon)$

Now, to get C_t we need the integral of individual c_t:

$$C_t = \int_0^1 c_t d\Gamma = \int_0^1 c(k_{t-1}, e_t, Z_t, \sigma_\epsilon) d\Gamma$$

If we had a reason to think that this c_t was *linear* in the state variables, then we could simply pull out those linear terms and write C_t as a function of K_t and Z_t. I.e. **constant MPCs** across distribution.

Equivalently, if we had complete markets, then we could use the aggregation theorem and argue that agents fully insure against any idiosyncratic risk.

But what if neither work? Then we have an **infinite dimensional** state variable. So how would we take g_x or h_x?

Value Function Iteration

Note that if instead you don't use linearization and perturbation, but instead try to use value function iteration (VFI), the same issue arises.

Recall that VFI involves constructing guess vectors for the value function and policy functions.

e.g. $V(k) = [0, .1, .2, ...]$

How to do that when the state is infinite dimensional?

Work-Around Without Aggregate Uncertainty

This is the case when their is *individual* risk/uncertainty but *aggregates* remain constant (e.g. Aiyagari, 1994).

That is, even if things are changing for individuals, there is a **stationary equilibrium** where things are not changing for the economy as a whole. I.e. **prices** are constant.

Basic logic: if we can use guess of fixed aggregates to find the decision rules of individuals at each individual state, then we just integrate these individual decisions over the stationary distribution of states.

Stationary Equilibrium

How to find the **stationary equilibrium**?

Like a steady state, except it's the *distribution* $\Gamma(k_{t-1}, e_t)$ - and thus the aggregates - that stays constant, even if e_t, k_t, and c_t switch around for individual agents.

Finding the stationary equilibrium involves:

1. taking a guess of the distribution (and thus the aggregate variables)
2. finding corresponding prices
3. solving the individual household's problem given those prices at every point on the joint distribution of states
4. then seeing if aggregating all those households validates your initial guess and updating if it doesn't

Aggregate Uncertainty

But what if we want to think about aggregate shocks? I.e. monetary policy shocks?

Boppart, Krussell, and Mitman (2017) present a (relatively) intuitive method using **MIT Shocks** and are one of the two main antecedents to the Sequence Space Jacobian method.

An **MIT Shock** is a one-time unexpected shock in the context of an otherwise perfect foresight model. I.e. before the shock, agents acted as if there was no aggregate uncertainty and after the shock, believe that there will be no further shocks.

What does an MIT shock/perfect foresight allow you to do? Allows you to write the problem in the **sequence space**.

State Space vs. Sequence Space

Rather than endogenous variables (e.g. C_t) being expressed recursively as policy functions of a state (e.g. K_{t-1}, Z_t), they're expressed as functions of the sequence of future aggregates which agents can anticipate with perfect foresight.

e.g. $C_t(K_{t-1}, Z_t)$ vs. $C_t(r_t, w_t, Z_t, ..., r_{t+s}, w_{t+s}, Z_{t+s}, ...)$

So writing a system in sequence space assumes perfect foresight with respect to aggregates.

BKM write their HA problem in sequence space. They assume that T periods after a shock the economy returns to the SE (which gives w_t, r_t, $Z_t, ...$ etc for all $t > T$).

They then simply *impose* a sequence of exogenous aggregates and *guess* a sequence of endogenous aggregates, solve for the response of agents to that sequence, and then see if the response and the sequences 'match'

Boppart Krusell and Mitman (2017)

Basic Logic:

- Start with the stationary equilibrium (SE) and its prices $\{r_0, w_0\}$
- Shock the economy creating a new deterministic path of the exogenous variable, $\{i_0 + 1, i_0 + \rho, i_0 + \rho^2, ...\}$
- Use the fact that the economy will eventually end up back at the SE, so $r_T = r_0$ and $w_T = w_0$, so you have the value functions for all types of agents at T, $V_T(k_T^i, e_T^i)\ \forall\ i$
- If you had r_{T-1} and w_{T-1} you could solve for $c_{T-1}^i(k_{T-1}^i, e_{T_1}^i)$ for all i, giving you V_{T-1}, and so forth...
- Guess a **sequence** of prices $\{r_0\ w_0, r_1\ w_1, ...r_{T-1}\ w_{T-1}\}$, solve the HH problem backwards, then simulate the economy forwards to see what aggregates the guess generates. If markets clear, then you're done! If not, update guess.

Reiter Method

Contrast this with the Reiter Method, the second major antecedent to the Sequence Space Jacobian

Stay in the **state space** and create a finite/discretized version of the distribution Γ and the decision rules g^i, for individual agents at the stationary equilibrium (fully non-linearly, no aggregate uncertainty yet.) Here, Γ and g are vectors.

Now take all the equilibrium conditions **for each of the points in the discretized distribution** (e.g. Euler equations) and stack them into a new (BIG!) system.

Linearize this with respect to aggregate shocks, and solve like an RBC model.

The Sequence Space Jacobian Method (Finally!)

We'll see in a moment that this method combines insights from both BKM and Reiter.

Why I like it: This methodology is both **intuitive** and **modular**. The different sectors of the economy are represented by individual 'blocks' that take in inputs and spit out outputs. It's easy to alter or tack on different pieces (e.g. if you wanted to tack on unions to a flexible wage model to get sticky wages). It's also **fast**.

Let's first start with the basic intuition of the method to get a sense of how this method differs from earlier ones.

Then we'll move on to the details of how to actually implement this method (including code!)

The Method

What we're after are impulse response functions to MIT shocks in a perfect foresight context: e.g. $\frac{dK_t}{dz_s}$

The SSJ method involves:

- Writing the equilibrium conditions in sequence space: **H(U,Z)=0**
- Get stationary equilibrium (non-linearized)
- Taking total derivatives evaluated at the stationary equilibrium, *w.r.t. elements of the aggregate sequence*
- Rearrange to get the impulse response functions: $dU = -H_U^{-1} H_Z dZ$ The H_U and H_Z are **all you need to know** (sufficient statistics) to see the impact of small dZ on U

Relation to earlier papers

Follows BKM in that they write the problem in sequence space meaning that the equilibrium equations are written as functions of a set of aggregate variables, not states. Assume return to SE at T so you can truncate.

Follows Reiter in the sense that they first solve the model non-linearly at the stationary equilibrium and then linearize w.r.t. aggregates.

To deal with heterogeneity Reiter generates an expanded system and linearizes w.r.t. aggregate states.

Instead, they map aggregate sequences into aggregate sequences by aggregating over individual decisions. But then just linearize w.r.t. these sequences! Map $w_t \rightarrow C_t$ because $C_t = \int c_t(w_t) d(e_t, k_{t-1})$. An algorithm they've developed to quickly compute the Jacobians of heterogeneous agent problems (e.g. $\frac{\partial C_t}{\partial w_t}$) is the key to why this works.

The Jacobian

Suppose we were working with the following simple RBC model:

$$\mathbf{H}_t(\mathbf{U}, Z) \equiv \begin{pmatrix} C_t^{-\sigma} - \beta(1 + r_{t+1})C_{t+1}^{-\sigma} \\ w_t - \varphi L_t^{\nu} C_t^{\sigma} \\ K_t - (1 - \delta)K_{t-1} - I_t \\ r_t + \delta - \alpha Z_t \left(\frac{K_{t-1}}{L_t}\right)^{\alpha-1} \\ w_t - (1 - \alpha)Z_t \left(\frac{K_{t-1}}{L_t}\right)^{\alpha} \\ Y_t - Z_t K_{t-1}^{\alpha} L_t^{1-\alpha} \\ Y_t - C_t - I_t \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

Here $U_t = \{C_t, r_t, K_t, Y_t, L_t, w_t, I_t\}$ and $Z_t = Z_t$

So the Jacobian H_U will be a 7 x 7 x T matrix of the derivatives of each equation with respect to each variable for each t evaluated at the stationary equilibrium (here, the steady state).

Solving for elements of Jacobian

Do we really include *every* endogenous variable in U? That could mean having to invert a massive matrix!

Instead, they break down the model into smaller sub-functions: i.e. different sub-components of the model called 'blocks'[1]. Essentially, this is just sophisticated variable substitution on a large scale.

$$\mathbf{H}_t(\mathbf{U}, Z) \equiv \begin{pmatrix} C_t^{-\sigma} - \beta(1 + r_{t+1})C_{t+1}^{-\sigma} \\ w_t - \varphi L_t^{\nu} C_t^{\sigma} \\ K_t - (1-\delta)K_{t-1} - I_t \\ r_t + \delta - \alpha Z_t \left(\frac{K_{t-1}}{L_t}\right)^{\alpha-1} \\ w_t - (1-\alpha) Z_t \left(\frac{K_{t-1}}{L_t}\right)^{\alpha} \\ Y_t - Z_t K_{t-1}^{\alpha} L_t^{1-\alpha} \\ Y_t - C_t - I_t \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

[1] You can find a formal definition in section 4.2

Simplified H function

$$\begin{bmatrix} C_t^{-\sigma} - \beta(1+r_{t+1})C_{t+1}^{-\sigma} \\ (1-\alpha)Z_t\left(\frac{K_{t-1}}{L_t}\right)^{\alpha} - \varphi L_t^{\nu} C_t^{\sigma} \\ K_t - (1-\delta)K_{t-1} - Y_t + C_t \\ r_t + \delta - \alpha Z_t \left(\frac{K_{t-1}}{L_t}\right)^{\alpha-1} \\ Y_t - Z_t K_{t-1}^{\alpha} L_t^{1-\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

H matrix is smaller, meaning inversion is easier!

We took the equation that, given the state K_{t-1} and K_t determines investment [call this the investment "block"] and subbed it into the resource constraint [a 'target'].

We've also taken the equation that, given L_t determines the implied wage [call this the labor supply "block"], and subbed it into the labor demand equation [another target].

Blocks

What is a 'block'? Blocks are mappings that take in sequences of aggregate inputs and spit out outputs.

For some rough intuition, start by thinking of blocks as 'sectors' of the economy (households, firms, finance, government, monetary policy).

- E.g. households take in wages and interest rates $\{w_t, r_t\}$ at time t and tell you how much they'll work, save, and consume, $\{L_t^S, K_t^S, C_t\}$
- Firms take in wages, interest rates, and aggregate TFP, $\{w_t, r_t, z_t\}$ and tell you how much capital and labor they will employ and therefore how much they'll produce, $\{Y_t, K_t^D, L_t^D\}$
- **Note*** You could also write the firm block as taking "in" equilibrium labor and capital $\{L_t, K_t, z_t\}$ and spitting "out" the prices w_t and r_t that rationalize those quantities.

These 'sector' blocks plus **'target'** blocks (e.g. $Y_t - C_t = 0$) close the model.

Directed Acyclical Graphs

There are 3 types of variables.

1. Exogeneous e.g. Z_t
2. Unknown e.g. r_t, K_t, Y_t, C_t, L_t
3. Intermediate e.g. w_t, I_t

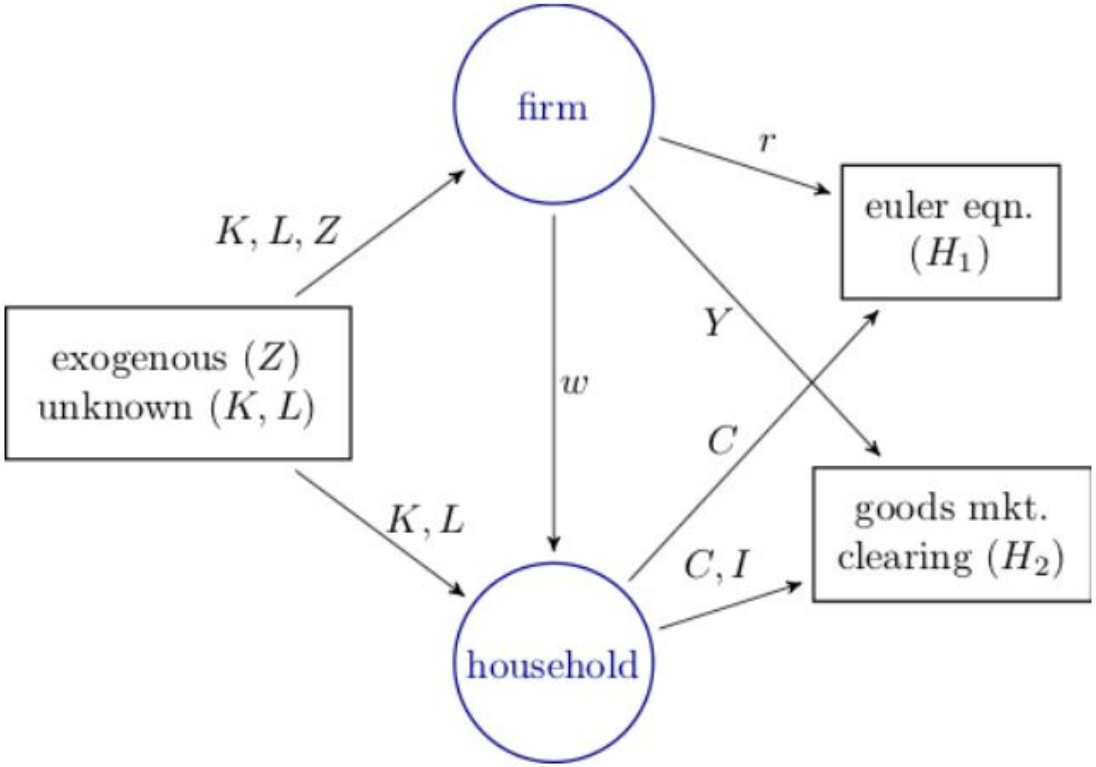
These mapping blocks are organized into DAGs.

The DAGs help you organize the blocks to make sure the following conditions are met:

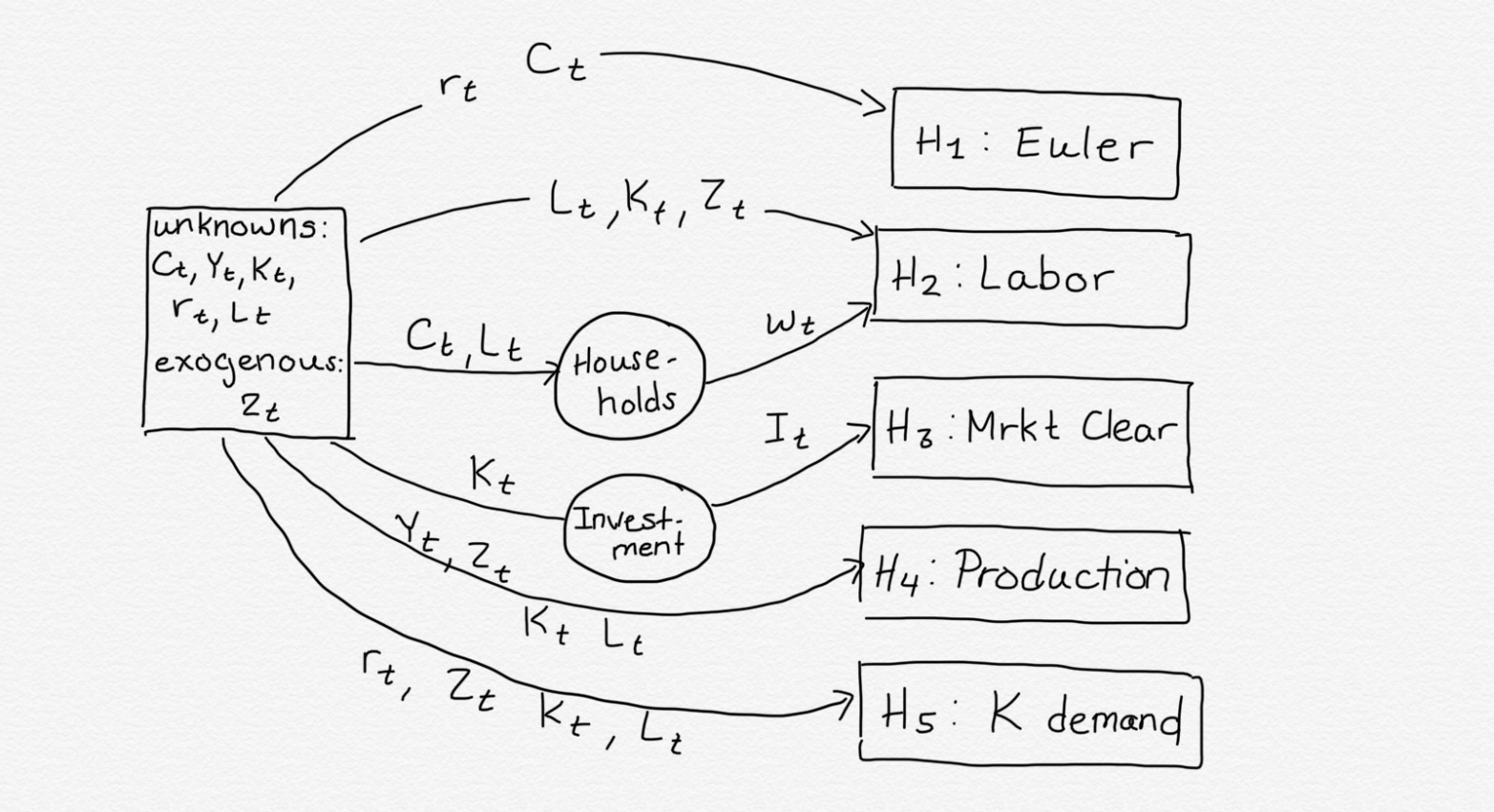
- Every intermediate output variable must be the input to another sector block or target block
- There are the same number of unknowns as targets

Just X equations (targets) in X unknowns, but the X equations are made up of a bunch of intermediate steps.

DAG: Example


firm
r
euler eqn.
(H_1)
K, L, Z
exogenous (Z)
unknown (K, L)
Y
w
C
K, L
goods mkt.
clearing (H_2)
C, I
household

DAG: The RBC Example



What this looks like in Python

Get Python

Python is useful for this method because functions are treated just like any other object and can be easily used as arguments in other functions or can be 'decorated', meaning modified without changing the underlying function. For example:

```
def plusone(x)
    return x + 1

def plustwo(x)
    return x + 2

@plustwo
def plusone(x)
    return x + 1
```

Simple Blocks

In other words, decorators take the output of the decorated function and do something else with it.

If your 'block' is simple in the sense that it just maps inputs to outputs (we'll get to other types in a second), simply decorate it with the @simple decorator so that the code knows what to do with this output.

```
@simple
def firm(K, L, Z, alpha, delta):
    r = alpha * Z * (K(-1) / L) ** (alpha-1) - delta
    w = (1 - alpha) * Z * (K(-1) / L) ** alpha
    Y = Z * K(-1) ** alpha * L ** (1 - alpha)
    return r, w, Y

@simple
def household(K, L, w, eis, frisch, vphi, delta):
    C = (w / vphi / L ** (1 / frisch)) ** eis
    I = K - (1 - delta) * K(-1)
    return C, I

@simple
def mkt_clearing(r, C, Y, I, K, L, w, eis, beta):
    goods_mkt = Y - C - I
    euler = C ** (-1 / eis) - beta * (1 + r(+1)) * C(+1) ** (-1 / eis)
    walras = C + K - (1 + r) * K(-1) - w * L  # we can the check dynamic version too
    return goods_mkt, euler, walras
```

Solved Blocks

What about when your equilibrium conditions don't generate nice explicit functions? (e.g. firm investment decisions)

They've created a decorator function @solved, which tells python to store/use the solution to a system of equations.

Example:

```
@solved(unknowns=['pi'], targets=['nkpc'])
def pricing(pi, mc, r, Y, kappap, mup):
    nkpc = kappap * (mc - 1/mup) + Y(+1) / Y * np.log(1 + pi(+1)) / (1 + r(+1)) - np.log(1 + pi)
    return nkpc
```

Heterogeneous Agent Blocks

How to handle HA blocks? [More details found in their Jupyter notebook]

- Write a **backward iteration function** representing the Bellman equation. It's best to have it take the derivative of the Value function next period (e.g. $\frac{\partial V_{t+1}}{\partial a_{t+1}}$), the transition matrix of the exogenous state[2], the discrete grids for both state variables, the parameters, and finally the 'inputs' to the block (e.g. w and r)
- It's output will be $\frac{\partial V_t}{\partial a_t}$ and the corresponding policy functions.
- Decorate with the @het decorator and define $\frac{\partial V}{\partial a}$, the transition matrix, and the endogenous state

```
@het(exogenous='Pi', policy='a', backward='Va')
def household(Va_p, Pi_p, a_grid, e_grid, r, w, beta, eis):
    """Single backward iteration step using endogenous gridp
```

[2] They have code to discretize an exogenous process and get this matrix, or you can write your own.

Writing the Backward Iteration Function

To write this function, you'll need to solve for the choice of a_{t+1} (and therefore c_t, and possibly n_t) that maximize utility given $V(a_{t+1})$. What's the best way to do this?

The **endogenous gridpoint method**.

The consumer's Bellman will look something like:

$$V(e_t, a_t) = max_{c_t, a_{t+1}} u(c_t) + \beta \sum_{e_t} V(e_{t+1}, a_{t+1}) P(e_t, e_{t+1})$$

Logic: Start with a 2D grid representing possible values for (e_{t+1}, a_{t+1}) and a guess for $V_{a_{t+1}}(a_{t+1}, e_{t+1})$

- Use this and transition matrix to get $c_t(e_t, a_{t+1})$
- Use budget to back out a_t on grid
- Assume monotonicity and interpolate

Writing the Backward Iteration Function

Take the derivative with respect to a_{t+1}, to get:
$0 = (-1)u'(c_t) + \beta \sum_{e_t} V_{a_{t+1}}(e_{t+1}, a_{t+1})$

$$u'(c_t) = \beta V_{a_{t+1}} \mathbf{P}$$

So with a (vector) guess for $V_{a_{t+1}}(e_{t+1}, a_{t+1})$, can get guess for $u'(c_t)$

Straightforwardly, the guess for $u'(c_t)$ associated with every e_t and a_{t+1} delivers a guess for a grid for c_t associated with every e_t and a_{t+1}. Then use the budget constraint to get a_t associated with these points.

Use interpolation and the assumption that the policy function is *monotonic* to get the policy function for a_t.

Finally, taking the derivative of Bellman w.r.t. a_t this time, gets you $V_{a_t}(e_t, a_t)$

Getting H_U

Now that we have a sense of what a block is and how they help reduce the dimension of the **H(U,Z)** function, we can think about how to actually find the elements of H_U and H_Z.

Recall we've just done fancy variable substitution, so finding elements of H_U (say for example, the derivative of the resource constraint with respect to K_t) you'll need a bunch of partial derivatives/partial equilibrium Jacobians (e.g. $\frac{\partial I}{\partial K_t}$).

Assuming you've already found the stationary equilibrium, taking the derivative of the simple and solved blocks is relatively straightforward. Simple numerical differentiation.

However, finding the block Jacobians for the HA blocks is a bit more complicated.

HA Problems using the SSJ Method

Assume that you've decomposed the set of equilibrium equations into blocks and condensed the size of the targets.

Example, let H_1 be the goods market clearing condition
$C_t(r_t(K_t), w_t(L_t)) + I_t(K_t) - Y_t(K_t, L_t) = 0$

Let K_t and L_t be your unknowns, U_t

To get $\frac{\partial H_{1t}}{\partial K_s} = J^{C,r} J^{r,K} + J^{I,K} - J^{Y,K}$ you'll need the partial equilibrium or block Jacobian, $J^{C,r} = \frac{\partial C_t}{\partial r_s}$

In this case, $C(r_t, w_t)$ is part of the HA component. I.e.
$C_t = \int c_t(e_t, k_t) dD(e_t, k_t)$

An efficient algorithm for solving for HA block jacobians is among the main contributions of the paper. Fake News Algorithm

Summing Up So Far

The Reiter method kept the HA problem in the "state space", discretizes the high dimensional state, and linearizes around stationary equilibrium w.r.t. aggregate shocks.

The Boppart, Krusell, Mitman method writes the system in aggregate sequence space and solves this system fully non-linearly. Assuming perfect foresight (MIT shocks).

This method also writes the problem in aggregate sequence space, but linearizes the system (takes derivatives of equilibrium equations w.r.t. elements of aggregate *sequences* evaluated at stationary equilibrium), then combines these linearized equations to get impulse response functions.

- Their algorithm that quickly solves for the derivative of HA variables w.r.t. aggregates is a big part of why this works Fake News Algorithm
- Decomposing the model into blocks both speeds up the process and makes models modular/easy to customize

More Python!

So let's assume you've:

1. Organized your model into a DAG of blocks
2. Coded these blocks up (simple, solved, or het blocks)
3. **Pro tip: I find its best to write my HA function in a separate file with a function that will also solve for the stationary equilibrium

Now what?

Solve for the stationary equilibrium (we'll just call this the steady state).

- Create a steady state function that takes as inputs: 1) all of your parameters, 2) calibration targets, 3) Guesses for any parameters you're solving for as part of the calibration
- Solve for whatever you can analytically!
- Solve for those remaining parameters using the het blocks - more on this in the next slide (e.g. target r $= 2\%$, K $= \frac{I}{\delta}$, so what β makes this work?)

Stationary Equilibrium

You've written the backward iteration functions for the het blocks (let's say you've named it "household") and decorated it with @het so the code knows what to do with it.

What it "does" with it is recognize that its a certain "class" or type of object. In python you can (and they do in their code package) introduce functions that are specific to/belong to certain objects. These are called 'methods'. (See Python tutorial)

household.ss() takes the function you've written, and all the inputs inside the () and finds the value function, policy functions, and aggregate choice variables at those values.

So just use a numerical solver (they provide a few for you!) to solve for remaining parameters (e.g. β). Finally, store everything in a dictionary which will be the output of your steady state function.

Solving the Model

To recap, you've created a file that contains:

- your HA block backward iteration function(s)
- a steady state function that takes in parameters, calibration targets, and guesses for any flexible parameters as inputs and returns all the elements of your stationary equilibrium

Now create a second file containing all of your other blocks with the steady state equal to the one solved for in your first file.

Declare the components of your model (T, block list, exogenous vars, unknowns, targets)

Solve first for the Jacobians (H_U and H_Z) by calling their ready-made functions. Then calculate impulse response functions!

Getting Started

Directions can be found on the authors' github page:
https://github.com/shade-econ/sequence-jacobian

To get started, just download Anaconda

- Super easy to use Python distribution platform.
- bunch of the most popular/useful packages
- Comes with Spyder, and IDE which is how you'll write, edit, debug, and compile and run your python code

Then simply download and unzip their code, making sure you store it in a location that Python can find (use Python path manager).

Go Back

Fake News Algorithm

Represent the HA block as a collection of:

- Inputs X_t (e.g. r_t and w_t)
- Outputs Y_t (e.g. C_t, A_t, N_t)
- A Distribution over individual states, D_t discretized into g points
- A value function v_t
- functions v, y, and Λ such that:

$$\begin{aligned} v_t &= v(v_{t+1}, X_t) \\ D_t &= \Lambda(v_{t+1}, X_t)' D_{t+1} \\ Y_t &= y(v_{t+1}, X_t)' D_t \end{aligned}$$

If these functions exist, then if we have a steady state input X_{ss}, can get the steady state v_{ss}, and thus the steady state D_{ss} and y_{ss}.

Fake News Algorithm

So assuming (as in BKM) the system starts and by T will end up at the steady state, these three equations define a mapping from the sequence of inputs $\{X_1, X_2, X_3, ...\}$ to the sequence of outputs $\{Y_1, Y_2, Y_3...\}$

To see this, just note that if you have $v_T = v_{ss}$ and you have X_{T-1} then you have v_{T-1}, and with X_{T-2} you have v_{T-2}, and so on ...

This immediately means I have all of the D_t terms, which finally gives me the Y_t terms. So we have: $\mathbf{Y}$=h($\mathbf{X}$). We need the Jacobian of h, J.

Suppose a shock happens at time s. I.e. there is a change to the sequence of inputs such that X_s increases by ϵ.

First insight: the response of the *individual* output variable/policy function y_t^s, will be the same as $y_{T-1-(s-t)}^{T-1}$ after. In other words, agents care about the distance to the shock, not the calendar time.

Fake News Algorithm

The other big result (for *small* dx):

Define $F_{t,s}dx$ as $dY_t^s - dY_{t-1}^{s-1}$ and D_t^w as the distribution at time t given a shock at time s.

... we have that $dy_t^s - dy_{t-1}^{s-1} = 0$, but $F_{t,s}$ isn't exactly 0 because $D_t^s \neq D_{t-1}^s$

They prove that $F_{t,s}dx = y'_{ss}(\Lambda'_{ss})dD_1^s$

Intuition: "To first order, this difference in initial distribution affects aggregates at all dates as if agents followed their steady state behavior"

TLDR: A single backward iteration starting from a shock at T-1 is all you need to calculate the whole Jacobian.

Go Back