About TA sections:

TAs: Ekaterina Zubova (ez268@cornell.edu), Zheyang Zhu (zz792@cornell.edu)

Section time and location: 8:40am - 9:55am Uris Hall 262 (section 201), Goldwin Smith Hall 236 (section 202)

Office hours: Tuesdays 5-7 pm in Uris Hall 451 (Ekaterina), Thursdays 5-7 pm in Uris Hall 429 (Zheyang). Other times available by appointment (just send us an email!)

Our plan for today:

1 Impulse Response Functions

An impulse response is the expected difference in a variable, conditional on a shock happening versus not happening. For example, $E[\hat{y}_{t+1}|\varepsilon_t^i = 1] - E[\hat{y}_{t+1}|\varepsilon_t^i = 0]$.

Using the notations we introduced earlier, impulse responses to a one-period temporary shock on the first state variable can be analytically expressed as follows:

$i = 0$ - period when the shock is happening:

$$IR^0(\hat{x}_t) = h\hat{x}_{t-1} + \eta \begin{bmatrix}1\\0\\0\end{bmatrix} - h\hat{x}_{t-1} + \eta \begin{bmatrix}0\\0\\0\end{bmatrix} = \eta \begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$IR^0(\hat{y}_t) = g \left[h\hat{x}_{t-1} + \eta \begin{bmatrix}1\\0\\0\end{bmatrix} - h\hat{x}_{t-1} + \eta \begin{bmatrix}0\\0\\0\end{bmatrix} \right] = g\eta \begin{bmatrix}1\\0\\0\end{bmatrix}$$

Iterating forward:

$i = 1$ - one period after the shock:

$$IR^1(\hat{x}_{t+1}) = h \left[h\hat{x}_{t-1} + \eta \begin{bmatrix}1\\0\\0\end{bmatrix} - h\hat{x}_{t-1} + \eta \begin{bmatrix}0\\0\\0\end{bmatrix} \right] = h\eta \begin{bmatrix}1\\0\\0\end{bmatrix}$$

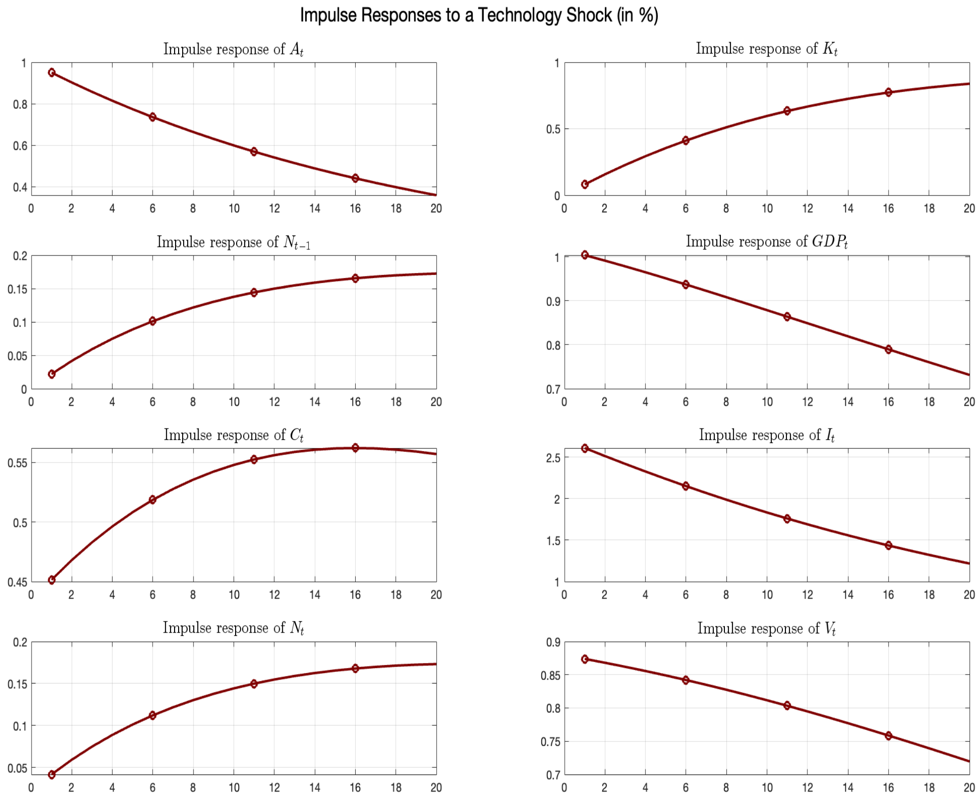
$$IR^1(\hat{y}_{t+1}) = gh \left[h\hat{x}_{t-1} + \eta \begin{bmatrix}1\\0\\0\end{bmatrix} - h\hat{x}_{t-1} + \eta \begin{bmatrix}0\\0\\0\end{bmatrix} \right] = gh\eta \begin{bmatrix}1\\0\\0\end{bmatrix}$$

For a general period $i = k$ after the shock:

$$IR^k(\hat{x}_{t+k}) = h^k \left[h\hat{x}_{t-1} + \eta \begin{bmatrix}1\\0\\0\end{bmatrix} - h\hat{x}_{t-1} + \eta \begin{bmatrix}0\\0\\0\end{bmatrix} \right] = h^k\eta \begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$IR^k(\hat{y}_{t+k}) = gh^k \left[h\hat{x}_{t-1} + \eta \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - h\hat{x}_{t-1} + \eta \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right] = gh^k \eta \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

An impulse response function (IRF) shows how variables in a model respond over time to a one-time shock. For example, in our model we considered one-period positive shock in technology. By applying log-linearization, which approximates non-linear relationships around the steady state, allowing us to analyze responses in linear terms, we got the following result:



The graphs clearly show that even though a productivity shock was introduced only in period $t = 0$, we shill observe its effect in the following periods. The formulas above provide the analytical basis for why this happens: initially, we observe the direct effect of the shock, which is the immediate, primary response of variables directly affected. In the following periods, the shock's influence persists through the model's dynamic structure, producing indirect effects as the initial response propagates over time.

2 Value Function for Our Model

We can rewrite our model as a value function for the Social Planner in the following form:

$$V(K_t, N_{t-1}, A_t) = \max_{K_{t+1}, N_t} U(C_t) + \beta E_t V(K_{t+1}, N_t, A_{t+1})$$

subject to

$$C_t = Y_t - I_t - \phi_n V_t = A_t K_t^{\alpha} N_t^{1-\alpha} - (K_{t+1} - (1-\delta_k)K_t) - \phi_n (\frac{N_t - (1-\delta_n)N_{t-1}}{\chi})^{1/\varepsilon}$$

and

$$U(C_t) \equiv \frac{C_t^{1-\sigma}}{1-\sigma}$$

Remark: Note that we substituted I_t from the law of motion for capital and V_t from the law of motion for labor and the definition of the matching function, assuming that $S_t = S = 1$.

To determine the optimality conditions, you need to derive the FOCs and Envelop conditions by **carefully** taking derivatives with respect to control variables (FOCs) and **endogenous** states (Envelope).

Combining two FOCs and two Envelope conditions, you will get capital and labor Euler equations:

$$C_t^{-\sigma} = \beta E_t \left[C_{t+1}^{-\sigma} 1 (A_{t+1} \alpha \left(\frac{K_{t+1}}{N_{t+1}} \right)^{\alpha-1} + 1 - \delta_k) \right]$$

$$\frac{\phi_n}{\chi \varepsilon V_t^{\varepsilon-1}} = A_t \left(\frac{K_t}{N_t} \right)^{\alpha} (1-\alpha) + \beta \mathbb{E}_t \left[\frac{C_{t+1}^{-\sigma}}{C_t^{-\sigma}} (1-\delta_n) \frac{\phi_n}{\chi \varepsilon V_{t+1}^{\varepsilon-1}} \right]$$

Just as we saw before!

3 Solution Algorithm 3: Value Function Iteration

The sample codes for value function iteration provided by Ryan include two MATLAB functions (**model_df.m** and **parameters.m**), one MATLAB script (**linear_model.m**) and one main MATLAB script (**solve_vf.m**). Additionally, we will use eight helper functions (**declare.m**, **gx_hx_alt.m**, **make_prime.m**, and **passign.m**, **vec.m**, **struct2array.m**, **make_index.m**, **AR1_rouwen.m**) in the helper_functions folder.

3.1 Functions in the helper_functions Folder

We have previously seen what **declare.m**, **gx_hx_alt.m**, **make_prime.m**, and **passign.m** do. Let's understand the new functions.

3.1.1 struct2array.m and vec.m

The function, **struct2array** (with the help of **vec.m**), takes a struct, **s**, with multiple fields, reshapes each field's data into a column vector, and combines all these column vectors into one single, long column vector **a**.

3.1.2 make_index.m

The function **make_index.m**:

1. Loops through each element in the input **Y**.
2. Creates a variable in the caller workspace for each element in **Y**, with a name based on the element.
3. Sets the value of each variable to the position of the element in **Y**.

For example, if you call **make_index(['A', 'B', 'C'])**, then you would get variables **a_idx**, **b_idx**, and **c_idx** in the workspace, with values 1, 2, and 3, respectively.

3.1.3 AR1_rouwen.m

This function, **AR1_rouwen**, generates a discrete approximation to an AR(1) process based on Kopecky & Suen (2010). The method provides higher accuracy for persistent processes compared to Tauchen's Method that we previously used. The function takes four inputs:

- N: Number of grid points (discrete values) for the AR(1) process.
- rho: The persistence parameter of the AR(1) process.

- mu: The mean of the AR(1) process.
- sige: The standard deviation of the shock in the AR(1) process.

The function returns three outputs:

- grid: The values that represent the discrete approximation of the AR(1) process.
- theta: The transition probability matrix.
- theta_bar: The stationary distribution of the Markov chain approximation.

3.2 Linearize (NOT log-linearize) the model

To make educated an initial guess for our value function, we'll use solutions from the linearized model to start the value function iteration. The code below provides policy matrices derived from the linearized model, which enable us to find our initial guess for the value function. While starting the value function iteration with a rough guess, like a matrix of zeros, is acceptable, using an educated guess can save considerable time.

3.2.1 linear_model.m

The script essentially combines the functionality of both **model_ss.m** and **model.m**. Key aspects of this script include:

- We are not exponentiating variables; instead, we are solely linearizing the model.
- We include the value function as our jump variable.
- In the second-to-last line, the script generates a new MATLAB function in the current folder: **matlabFunction(fy,fx,fyp,fxp,fv,[Yss,Xss],'vars',pvec,'file', 'model_df.m', 'optimize', false)**:
  - **matlabFunction** is a MATLAB command that converts symbolic expressions into a function file. It allows you to save symbolic expressions in a way that makes them easy to reuse without recalculating them every time.
  - **fy**, **fx**, **fyp**, **fxp**, **fv** are symbolic matrices evaluated at steady-state values. **[Yss, Xss]** is a vector of steady-state values for the state and jump variables.
  - **'vars', {pvec}**: Tells MATLAB that the generated function should take one input, **pvec**. **pvec** is a row vector containing the parameters of the model, so **model_df.m**

will take **pvec** as an input to evaluate **fy**, **fx**, **fyp**, **fxp**, **fv**, **[Yss, Xss]** based on the parameter values.

  – **'file', 'model_df.m'**: Specifies the name of the file to be created. MATLAB will save the function as **model_df.m** in the current directory.

  – **'optimize', false**: Prevents MATLAB from performing additional optimizations on the generated code.

- The script also creates indices for the states and jumps using the **make_index.m** function.

3.2.2 model_df.m

The function is created by the **model_df.m** script using **matlabFunction**. It takes a row vector, **pvec**, as input and outputs **fy**, **fx**, **fyp**, **fxp**, **fv**, and **[Yss, Xss]**.

3.3 Value Function Iteration (solve_vf.m)

Again, we omit the explanation for the plotting codes.

- **addpath('helper_functions')** adds a directory containing helper functions.
- **linear_model** runs the **linear_model.m** script and creates the function, **model_df.m**.
- **rehash** updates the MATLAB list of known files in the folders, so that the newly-created **model_df** function can be read.
- **param = parameters** loads the model parameters, stored in a struct param.
- **pval = struct2array(param)** converts the parameter structure to an array (a column vector) for easy access.
- **model_df(pval')** computes matrices of the linearized model.
- **gx_hx_alt** solves for policy matrices of the linearized model.
- **eta = [0;1]** represents the shock vector.
- Steady-state values are extracted from **yxss**.
- **agrid**: A grid of possible productivity levels A, generated from the **AR1_rouwen** function to represent stochastic productivity. Remember to exponentiate the vector.

- **kgrid** and **hgrid**: Arrays representing possible values for K and H, created as a range around the steady-state values.
- **aagr** and **kkgr** represent combinations of states, A and K.
- **kkgr2** and **hhgr2** represent possible choices of K' and H given different combinations of A and K, i.e. $K'(A,K)$ and $H(A,K)$.
- **kinit**, **hinit**, and **vinit** are the initial guesses for the policy functions, $K'(A,K)$ and $H(A,K)$, and value function, $V(A,K)$, based on the policy matrices from the linearized model.
- **idx** keeps track of the optimal choice indices for each combination of state variables.
- **crit = 1** sets the initial difference between value functions as 1.
- **jj** counts the number of iterations.
- **tic** and **toc** starts and ends the timer for measuring performance.
- **nfix = 25** sets the frequency of full policy updates for computational efficiency.
- The value function iterations continues until **crit** (the convergence criterion) is less than 1e-6 or the iteration count **jj** reaches 1000.
- **vinit_old = vinit**: **vinit_old** stores the value of **vinit** from the previous iteration, so we can check if the value function is converging.
- **EVp** is a matrix that represents expected future values of the value function using the transition matrix **theta**.
- **if mod(jj,nfix) == 0**: This checks if **jj** is a multiple of **nfix**. If true, we recompute the policy functions for each possible combination of states. Otherwise, we skip this and use the previous policy functions.
- In the recompute policy function case, we go through every possible combination of A and K. We record the maximum value of the value function obtained and the index of the choices that maximize the value function.
- In the reuse policy function case, we only update the value function.
- We check if the value function is converging by comparing the difference between **vinit** and

vinit_old. If the maximum difference is small enough, the loop will exit.

- Finally, we recover the optimal choice values from optimal choice index.