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]

[1]Materials adapted from notes provided by a previous Teaching Assistant, Gautier Lenfant.

1 RBC model

For your reference, we provide a description of the Real Business Cycle (RBC) model, which is used in the sample code available on Canvas.

Representative household

The representative household (RH) solves the following problem:

$$\max_{(C_t)_{t\geq 0},(H_t)_{t\geq 0},(I_t)_{t\geq 0},(K_{t+1})_{t\geq 0}} \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \left[\log(C_t) - \chi H_t\right]$$

such that

$$C_t + I_t = R_t K_t + W_t H_t$$

$$K_{t+1} = (1-\delta) K_t + I_t$$

The RH takes $(R_t)_{t\geq 0}, (W_t)_{t\geq 0}$ and K_0 as given.

Remark: Note the difference in notation. Here, labor is denoted by H, whereas in our class model, it is represented as N. The RH has a disutility for labor, as seen from the structure of the utility function. This feature will influence the labor supply decision.

Firm

The firm solves a static profit maximization problem:

$$\max_{K_t, H_t} K_t^{\alpha} \left(X_t H_t\right)^{1-\alpha} - W_t H_t - R_t K_t$$

where the production function is represented by a Cobb-Douglas function with constant returns to scale and labor-augmenting technology.

2 Solving Our Log-linearized Model

Let us recall our model from class. We can rewrite it by moving all variables to the left-hand side:

$$Y_t - A_t K_t^{\alpha} N_t^{1-\alpha} = 0 \tag{1}$$

$$K_{t+1} - (1-\delta)K_t - I_t = 0 \tag{2}$$

$$N_t - (1-\delta_n)N_{t-1} - \chi V_t^{\varepsilon} = 0 \tag{3}$$

$$Y_t - C_t - I_t - \phi V_t = 0 \tag{4}$$

$$1 - \beta \mathbb{E}_t \left[\left(\frac{C_{t+1}}{C_t} \right)^{-\sigma} \left(A_{t+1}\alpha \left(\frac{K_{t+1}}{N_{t+1}} \right)^{\alpha-1} + 1 - \delta \right) \right] = 0 \tag{5}$$

$$\frac{\phi_n}{\varepsilon\chi} V_t^{1-\varepsilon} - A_t(1-\alpha)\left(\frac{K_t}{N_t}\right)^{\alpha} - \beta \mathbb{E}_t \left[\left(\frac{C_{t+1}}{C_t} \right)^{-\sigma} \frac{\phi_n}{\varepsilon\chi} V_{t+1}^{1-\varepsilon}(1-\delta_n) \right] = 0 \tag{6}$$

We can express this system synthetically as:

$$\mathbb{E}_t[F(X_t, Y_t, X_{t+1}, Y_{t+1})] = 0 \tag{7}$$

Our goal is to formulate a linear state-space model where our variables (x_t, y_t) represent log deviations from the steady state. Specifically, we want the system to take the following form:

$$x_{t+1} = h_x x_t + \varepsilon_{t+1}$$

$$y_t = g_x x_t$$

Recall the notations from the last section:

$$\log(Z_t) = \widehat{z}_t$$

$$\log(Z_t) - \log(Z^{ss}) = z_t \iff \widehat{z}_t = z_t + z^{ss}$$

Using these notations, the system of (potentially nonlinear) equilibrium equations (7) can be re-written and Taylor approximated at the first order as follows:

$$\mathbb{E}_t[F(X_t, Y_t, X_{t+1}, Y_{t+1})] = \mathbb{E}_t[F(\exp(\widehat{x}_t), \exp(\widehat{y}_t), \exp(\widehat{x}_{t+1}), \exp(\widehat{y}_{t+1}))] = 0$$

$$\iff \mathbb{E}_t[F(X_t, Y_t, X_{t+1}, Y_{t+1})] = \mathbb{E}_t[\underbrace{F \circ \exp}_{=f}(x_t + x^{ss},\ y_t + y^{ss},\ x_{t+1} + x^{ss},\ y_{t+1} + y^{ss})]$$

$$\approx \underbrace{\mathbb{E}_t[f(x^{ss}, y^{ss}, x^{ss}, y^{ss})]}_{=0} + \mathbb{E}_t[f_x(x^{ss})x_t + f_y(y^{ss})y_t + f_{xp}(x^{ss})x_{t+1} + f_{yp}(y^{ss})y_{t+1}]$$

Simplifying the notation, we have:

$$\mathbb{E}_t[F(X_t, Y_t, X_{t+1}, Y_{t+1})] \approx \mathbb{E}_t[f_x x_t + f_y y_t + f_{xp} x_{t+1} + f_{yp} y_{t+1}]$$

In the next step, we need to define the terms f_x, f_y, f_{xp}, and f_{yp}. Recall that the system of equations, once log-linearized, is given by:

$$Yy_t = Ya_t + Y\alpha k_t + (1-\alpha)Yn_t$$

$$Kk_{t+1} = (1-\delta)Kk_t + \delta K i_t$$

$$Nn_t = (1-\delta_n)Nn_{t-1} + \chi\varepsilon V^{\varepsilon} v_t$$

$$Yy_t = Cc_t + Ii_t + \phi V v_t$$

$$0 = \mathbb{E}_t\left[\sigma\left(\alpha\left(\frac{K}{N}\right)^{\alpha-1} + 1 - \delta\right)(c_t - c_{t+1}) + \alpha\left(\frac{K}{N}\right)^{\alpha-1}(a_{t+1} + (\alpha-1)k_{t+1} - (\alpha-1)n_{t+1})\right]$$

$$\frac{\phi_n}{\varepsilon\chi}(1-\varepsilon)V^{1-\varepsilon}v_t = (1-\alpha)\left(\frac{K}{N}\right)^{\alpha}(a_t + \alpha(k_t - n_t)) + \beta\mathbb{E}_t\left(\frac{\phi_n}{\varepsilon\chi}V^{1-\varepsilon}(1-\delta_n)(\sigma(c_t - c_{t+1}) + (1-\varepsilon)v_{t+1})\right)$$

Writing f as:

$$f : (\underbrace{a_t, k_t, n_{t-1}}_{x_t}, \underbrace{y_t, c_t, i_t, n_t, v_t}_{y_t}, \underbrace{a_{t+1}, k_{t+1}, n_t}_{x_{t+1}}, \underbrace{y_{t+1}, c_{t+1}, i_{t+1}, n_{t+1}, v_{t+1}}_{y_{t+1}}) \to$$

$$\begin{cases} \sigma(c_{t+1}-c_t)-\beta\alpha\left(\frac{K}{N}\right)^{\alpha-1}(a_{t+1}+(\alpha-1)k_{t+1}-(\alpha-1)n_{t+1}) \\ Nn_t - Nn_t \\ \frac{\phi_n}{\varepsilon\chi}(1-\varepsilon)V^{1-\varepsilon}v_t-(1-\alpha)\left(\frac{K}{N}\right)^{\alpha}(a_t+\alpha(k_t-n_t))-\beta\frac{\phi_n}{\varepsilon\chi}V^{1-\varepsilon}(1-\delta_n)(\sigma(c_t-c_{t+1})+(1-\varepsilon)v_{t+1}) \\ Kk_{t+1}-K(1-\delta)k_t-K\delta i_t \\ Nn_t-(1-\delta_n)Nn_{t-1}-\chi\varepsilon V^{\varepsilon}v_t \\ Yy_t-Cc_t-Ii_t-\phi Vv_t \\ Yy_t-Ya_t-Y\alpha k_t-(1-\alpha)Yn_t \\ Aa_{t+1}-\rho Aa_t \end{cases} \tag{1}$$

Setting the above linear mapping to 0, you do get $\mathbb{E}_t[f_x x_t+f_y y_t+f_{xp}x_{t+1}+f_{yp}y_{t+1}]=0$.

In matrix form, one can thus write:

$$f_x=\begin{bmatrix} f_{a_t} & f_{k_t} & f_{n_{t-1}} \end{bmatrix}$$

$$\Rightarrow f_x=\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -(1-\alpha)\left(\frac{K}{N}\right)^{\alpha} & -(1-\alpha)\alpha\left(\frac{K}{N}\right)^{\alpha} & 0 \\ 0 & -(1-\delta)K & 0 \\ 0 & 0 & -(1-\delta_n)N \\ 0 & 0 & 0 \\ -Y & -\alpha Y & 0 \\ -\rho & 0 & 0 \end{bmatrix}$$

$$f_y=\begin{bmatrix} f_{y_t} & f_{c_t} & f_{i_t} & f_{n_t} & f_{v_t} \end{bmatrix}$$

$$\Rightarrow f_y = \begin{bmatrix} 0 & -\sigma & 0 & 0 & 0 \\ 0 & 0 & 0 & -N & 0 \\ 0 & -\beta\frac{\phi_n}{\varepsilon\chi}V^{1-\varepsilon} & 0 & \alpha(1-\alpha)\left(\frac{K}{N}\right)^{\alpha} & (1-\varepsilon)\frac{\phi_n}{\varepsilon\chi}V^{1-\varepsilon} \\ 0 & 0 & -K\delta & 0 & 0 \\ 0 & 0 & 0 & N & -\chi\varepsilon V^{\varepsilon} \\ Y & -C & -I & 0 & -\phi V \\ -Y & 0 & 0 & -Y(1-\alpha) & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$f_{xp} = \begin{bmatrix} f_{a_{t+1}} & f_{k_{t+1}} & f_{n_t} \end{bmatrix}$$

$$\Rightarrow f_{x_p} = \begin{bmatrix} -\beta\alpha\left(\frac{K}{N}\right)^{\alpha-1} & \beta\alpha(1-\alpha)\left(\frac{K}{N}\right)^{\alpha-1} & 0 \\ 0 & 0 & N \\ 0 & 0 & 0 \\ 0 & K & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$f_{y_p} = \begin{bmatrix} f_{y_{t+1}} & f_{c_{t+1}} & f_{i_{t+1}} & f_{n_{t+1}} & f_{v_{t+1}} \end{bmatrix}$$

$$\Rightarrow f_{y_p} = \begin{bmatrix} 0 & \sigma & 0 & -\beta\alpha(1-\alpha)\left(\frac{K}{N}\right)^{\alpha-1} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma\beta\frac{\phi_n}{\chi\varepsilon}(1-\delta_n)V^{1-\varepsilon} & 0 & 0 & -\beta\frac{\phi_n}{\chi\varepsilon}(1-\delta_n)(1-\varepsilon)V^{1-\varepsilon} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3 Solution Algorithm 1: Log-Linearization

Let's talk about what our sample code is doing. The sample code on canvas is based on the RBC model. The setup of the RBC model can be found in the first part of this section note.

- **parameters.m**: This code creates a set of parameters for a model by storing specific values (like numbers that the model will use) into a structure called **param**. Think of it like a mini-database.

- **model_ss.m**: This code calculates the steady state of the model.

  - **function [ss, param] = model_ss(param)** means this function is called model_ss, and it takes in one input (**param**) and gives two outputs (**ss** and **param**).

  - The first few lines take values from the param structure (created in the last function) and store them in simpler names like **bet**, **gam**, **chi**, etc.

  - Each steady-state variable is calculated with a formula that uses these parameters.

  - The **yy** vector collects jump variables and the **xx** vector collects the state variables. Finally, **ss** = **[yy xx]** puts these values into one steady-state vector which is the output.

- **model.m**: This code creates a linearized version of the RBC model.

  - **function [fyn, fxn, fypn, fxpn, fn] = model(param)** defines a function called model, which takes in param structure and outputs five different matrices related to the model.

  - The line **[ss, param] = model_ss(param)** computes the steady state of the model using the function **model_ss**, which we've seen before. This returns the steady-state values for later calculations.

  - We then retrieve the parameters from **param**, like **bet**, **gam**, **chi**, etc., which were defined in parameters.m.

  - Using **syms**, the code creates symbolic variables (e.g., **C**, **K**, **W**, **R**) to represent the main model variables both in the current period and in the future period (denoted with **_p** for "prime").
  - **X** and **XP** are vectors of state variables in terms of symbolic variables, with **XP** representing the next period's variables. **Y** and **YP** are vectors of jump variables in terms of symbolic variables, with **YP** representing the next period's variables.
  - Then we set up equations in **f** that define the relationships among variables in the model.
  - We substitute the steady-state values into the equations to ensure everything balances out as expected in steady state.
  - **f** is redefined using exponential functions for the variables in **var_list**.
  - We take derivatives (jacobians) of the model equations with respect to **X**, **Y**, **XP**, and **YP**. These derivatives form matrices in terms of symbolic variables.
  - Finally, the function evaluates the symbolic derivatives at the steady-state values and stores them as **fxn**, **fyn**, **fxpn**, **fypn**, and **fn**, representing the linearized model matrices with numerical values.

- **gx_hx_alt.m**: This function finds the policy matrices (**gx** and **hx**).
  - The function **gx_hx_alt** takes inputs **fy**, **fx**, **fyp**, **fxp** (matrices of derivatives from linearizing the model equations) and outputs **gx** and **hx** (policy matrices).

4 Matching the Data

When dealing with macro models, we often want to calibrate them to match real-world data. There are several strategies to ensure the model reflects key economic features. Any ideas?