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 Solution Algorithm 2: Shooting Method

The sample codes provided by Ryan include four MATLAB functions (**model.m**, **model_ss.m**, **parameters.m**, and **resid.m**) and one main program script (**main_prog.m**). Additionally, we will use four helper functions (**declare.m**, **gx_hx_alt.m**, **make_prime.m**, and **passign.m**) in the helper_functions folder.

1.1 Functions in the helper_functions Folder

1.1.1 passign.m

The function takes each field of the struct (a type of data structure that groups variables together) and creates separate variables outside the function. If you call **passign(myStruct)** and **myStruct** has fields **a**, **b**, and **c**, then after running the function, you'll have new variables **a**, **b**, and **c** with the values from **myStruct** in your workspace.

1.1.2 declare.m

The function creates symbolic variables based on the input (either do **declare A B C**, or do **declare("A", "B", "C")**), then puts those symbolic variables into a vector named **V** and also assigns each variable individually in the workspace.

1.1.3 make_prime.m

The function takes a symbolic vector **V** and returns a new symbolic vector **Vp** with the same elements as **V** but with each element's name modified to include a **_p** suffix.

1.1.4 gx_hx_alt.m

The function takes inputs **fy**, **fx**, **fyp**, **fxp** (matrices of derivatives (or Jacobians) from (log-)linearizing the model equations evaluted at the (log) steady state) and outputs **gx** and **hx** (policy matrices).

1.2 parameters.m

The function prepares a set of model parameters as fields in a **param** struct, allowing them to be passed easily to other functions and scripts.

1.3 model_ss.m

The function calculates steady-state values for state variables and jump variables using formulas based on the model's parameters. It returns these values in two vectors, **Xss** and **Yss**, which represent states and jumps in steady state.

1.4 model.m

This function computes the log-linearized version of the RBC model using symbolic calculations in MATLAB. It returns **fxn**, **fyn**, **fxpn**, **fypn**, and **fn**, representing the log-linearized model matrices evaluated at log steady states.

1.5 resid.m (Main Function for Shooting Method)

The function calculates the residuals (differences) of the model equations based on a guessed path for the variables over time. It helps assess how close the guessed paths are to satisfying the model's equations. Let's break down each part of the code.

- **function f = resid(XYv, ss, param, log_var)**: The function **resid** takes four inputs:
  - **XYv**: A matrix of size (# state + # jump, # time period) containing guesses for the paths of variables in log deviations over time. In the sample code, # time period is 500.
  - **ss**: The log steady-state values of these variables.
  - **param**: A struct with model parameters.
  - **log_var**: Indices of variables that should be in log.

  The function returns **f**, a matrix of size (# model equations, # time period) where each row represents the residual of a model equation over time.

- **passign(param)**: Uses **passign** to assign each parameter in **param** as a separate variable, making them accessible by name.

- **XYv = reshape(XYv,[8,numel(XYv)/8]) + ss(:)**: Reshapes **XYv** to an 8-

row matrix (one row per variable) and adds back the log steady-state values **ss**. This creates the variables in log over time.

- Converts the log variables in **XYv** back to levels by exponentiating them. Does the same for the steady-state values in **ss**. We do this because the model questions are in terms of level variables.

- We will create row vectors for states, jumps, next-period states, and next-period jumps, each with a size of (1, # time period). To set this up, we first create auxiliary row vectors of size (1, # time period + 1) for each variable.

  - For the exogenous state that shocks our model, **A**, the first element in the row vector will be the shock value, **exp(log(ss(1))+siga)**. The remaining elements in this vector will be the guessed path for **A**, represented by **XYv(1,:)**.

  - For the endogenous state, **K**, the first element in the row vector will be its steady state value in level, **ss(2)**. The rest of the row vector will be our guessed path for **K**, represented by **XYv(2,:)**.

  - For jumps, the first part of the row vector (in total # time period elements) will contain the guessed path for the jump variable, and the final element will be set to its steady state value in level.

  To access present-time variables, take the first 500 (# time period) elements from each auxiliary vector. For next-period variables, take the last 500 (# time period) elements from each auxiliary vector.

- Finally, we create a matrix of size (# model equations, # time period). Each equation is defined element-wise (using **.***, **./**, etc.) to handle paths over time. Each row of the matrix represents the residuals of a particular equation over time. If all equations are perfectly satisfied, **f** would be zero across all elements.

The **resid** function calculates the residuals for each model equation over a sequence of time steps. By checking these residuals, you can see how close a guessed solution is to satisfying the model's equations dynamically.

1.6 main_prog.m

For the main program script, we skip the explanation for the codes for plotting.

- **param = parameters**: Calls the parameters function to create a structure of model parameters.
- **passign(param)**: Unpacks the param struct, making each parameter accessible as a variable.
- **[Xss, Yss] = model_ss(param)**: Calculates the steady-state values in level Xss (for state variables) and Yss (for jump variables) using the **model_ss** function.
- **[fyn, fxn, fypn, fxpn, fn, log_var] = model(param)**: Calls the **model** function to compute the Jacobian matrices (**fyn**, **fxn**, **fypn**, **fxpn**), which represent the model's log-linearized dynamics.
- **ss(log_var) = log(ss(log_var))**: Converts certain steady-state values to logs.
- **eta = [siga; 0]**: Sets up a shock to **GAM** (the first state variable) with standard deviation **siga**.
- Impulse response computation:
  - Initializes **X** with a size matching the number of state variables, setting the initial shock.
  - Iteratively applies **hx** to compute the path of **X** over time in response to the initial shock.
  - Calculates the path of jump variables (**Y**) based on **X** using the policy matrix **gx**.
- Shooting method:
  - **XYv = [X(:,2:end), zeros(2,1); Y(:,1:end)]**: Sets up the initial guess for X(t+1) and Y(t) in log deviations, based on the log-linearized model's

output. Note that it is fine if your initial guess a just a matrix of random numbers, but a more educated guess saves time for computation.

- **resid0 = resid(XYv, ss, param, log_var)**: Calculates initial residuals to check if the current guess for paths is close to satisfying the model's equations.

- **obj = @(x) resid(x, ss, param, log_var)**: Defines an anonymous function, **obj**, that calculates residuals for any given guess of paths (**x**) fixing **ss**, **param**, and **log_var**.

- **XpYshoot = fsolve(obj, XYv, options)**: Calls **fsolve** to adjust the guess **XYv** until residuals are 0 for all model equations at each time.

- **XYshoot = [X(:,1), XpYshoot(1:2,1:end-1); XpYshoot(3:end,:)]**: Combines the solution from **fsolve** with the initial state for a full path solution, **XYshoot**.

2 Decentralized Economy

2.1 Model Setup

Household's Optimization Problem

Consider a household that chooses consumption and investment in shares of the representative firm. The household's objective is to maximize the following expected life-time utility:

$$\max_{\{C_t,\Gamma_t\}} \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \left\{ \frac{C_t^{1-\sigma}}{1-\sigma} + \lambda_t \left[W_t N_t + \Gamma_{t-1}(D_t + P_t) - C_t - \Gamma_t P_t \right] \right\}$$

where:

- λ_t is the Lagrange multiplier on the budget constraint at time t, representing the shadow value of wealth.
- W_t is the wage rate at time t.
- Γ_t is the amount of shares of the representative firm held by the household at time t.
  Remark: Since we have a representative agent model, note that $\Gamma_t = \Gamma = 1$ (as the representative agent owns the entire firm). However, we see this as one of the market clearing conditions and still take the first-order condition with respect to Γ_t to assign a value to firm ownership.
- D_t is the dividend paid by the representative firm at time t.
- P_t is the price of a share of the representative firm at time t.

Firm's Optimization Problem

Remark: Note the difference from what we saw in the previous part of this course. Recall the neoclassical growth model: we set it up so that the representative household chose how much to invest in capital, while the representative firm solved an unconstrained optimization problem to maximize its profit. Here, the setting is different:

The firm chooses the optimal levels of output, vacancies, investment, employment, and capital stock, and pays a wage w_t. The probability that a vacancy gets filled, Q_t, is taken as **exogenous**.

The firm's objective function represents the discounted present value of profits, where profits are defined as revenue minus costs:

$$\begin{aligned}\max_{\{Y_t,I_t,V_t,K_{t+1},N_{t+1}\}} \mathbb{E}_0 \sum_{t=1}^{\infty} \beta^t \Bigg\{ & \frac{\lambda_t}{\lambda_0} \left[Y_t - W_t N_t - I_t - \phi_n V_t \right] \\ & + \frac{1}{\lambda_0} \Gamma_{1,t} \left[(1-\delta_n) N_{t-1} + V_t Q_t - N_t \right] \\ & + \frac{1}{\lambda_0} \Gamma_{2,t} \left[(1-\delta_k) K_t + I_t - K_{t+1} \right] \\ & + \frac{1}{\lambda_0} \Gamma_{3,t} \left[A_t K_t^{\alpha} N_t^{1-\alpha} - Y_t \right] \Bigg\}\end{aligned}$$

where:

- $\Gamma_{1,t}$, $\Gamma_{2,t}$, $\Gamma_{3,t}$: Lagrange multipliers associated with the firm's constraints. **Have nothing to do with Γ_t representing firm shares**!

- Q_t: Probability that a vacancy gets filled, **assumed exogenous** (so we do not substitute the formula for Q just yet!).

- λ_t/λ_0: Stochastic discount factor.
  Remark: λ_0 and λ_t are the same as in the household problem, representing the shadow value of wealth in terms of marginal utility.
  Intuition: Since the household owns the firm, it values the income flow generated by the firm's profits as part of its total wealth. The shadow value of this income flow is, therefore, identical to the shadow value of wealth for the household. Thus, the firm's per-period profits are discounted accordingly.

2.2 Key Equations

Optimality conditions for the representative household (follow from the FOCs):

$$C_t^{-\sigma} = \lambda_t$$

Intuition: This condition states that the Lagrange multiplier λ_t on the household's budget constraint is equal to the marginal utility of consumption, $C_t^{-\sigma}$ (we have seen this result many times).

$$P_t = \beta \mathbb{E}_t \left[\underbrace{\left(\frac{C_{t+1}}{C_t} \right)^{-\sigma}}_{\text{Stochastic Discount Factor}} (P_{t+1} + D_{t+1}) \right]$$

Intuition: This equation describes the optimal condition for holding shares. The price of a share today, P_t, equals the discounted expected future payoff (next period's price P_{t+1} plus dividend D_{t+1}), adjusted by the *stochastic discount factor* $\left(\frac{C_{t+1}}{C_t} \right)^{-\sigma}$.

Optimality conditions for the representative firm (follow from the FOCs):

$$1 = \beta \mathbb{E}_t \left[\frac{\lambda_{t+1}}{\lambda_t} A_{t+1} \alpha \left(\frac{K_{t+1}}{N_{t+1}} \right)^{\alpha - 1} + 1 - \delta_k \right]$$

Intuition: This condition (Euler equation for capital) represents the firm's optimal choice for capital - just the same as what we saw in the SPP. The left side represents the "cost" of one unit of capital. The right side is the expected marginal benefit, discounted by β and adjusted by the ratio of future to current marginal utility $\frac{\lambda_{t+1}}{\lambda_t}$.

$$\frac{\phi_n}{Q_t} = \underbrace{A_t \left(\frac{K_t}{N_t} \right)^{\alpha} (1 - \alpha) - W_t}_{\text{Net Benefit of Hiring a Worker}} + \underbrace{\beta \mathbb{E}_t \left[\frac{\lambda_{t+1}}{\lambda_t} (1 - \delta_n) \frac{\phi_n}{Q_{t+1}} \right]}_{\text{Continuation Value of Having This Worker}}$$

Intuition: This equation (Euler equation for labor) describes the firm's optimal choice for vacancies. The left side represents the cost per posting a vacancy, adjusted

for the probability of successfully filling it. The right side represents the net marginal benefit of filling this vacancy, which includes the net flow from the matched worker (the marginal product of labor minus the wage) and the continuation value, representing the savings from not having to hire this worker again in the next period.

2.3 Differences from SPP

The key differences between the solutions to the decentralized economy and social planner problem are in the Euler equation for labor:

$$\frac{\phi_n}{Q_t} = A_t \left(\frac{K_t}{N_t}\right)^{\alpha} (1-\alpha) - W_t + \beta \mathbb{E}_t \left[\frac{\lambda_{t+1}}{\lambda_t}(1-\delta_n)\frac{\phi_n}{Q_{t+1}}\right] \quad \text{(Decentralized)}$$

$$\frac{\phi_n}{M_v(\cdot)} = A_t \left(\frac{K_t}{N_t}\right)^{\alpha} (1-\alpha) + \beta \mathbb{E}_t \left[\frac{\lambda_{t+1}}{\lambda_t}(1-\delta_n)\frac{\phi_n}{M_v(\cdot)}\right] \quad \text{(Social Planner)}$$

- In the **Decentralized** economy, the term Q_t represents the probability that a vacancy is filled in the market. This reflects the vacancy-filling process in a decentralized labor market where firms consider the exogenous probability Q_t when posting vacancies (think of it as an externality).

- In the **Social Planner** setting, the term $M_v(\cdot)$ represents a planner-controlled matching function that determines how vacancies and job seekers are matched. This allows the planner to directly control the vacancy matching process, potentially making it more efficient than the decentralized market.

- The decentralized equation includes the wage term W_t, which represents the wage firms must pay workers in a competitive labor market. In the decentralized economy, firms take this wage as given and consider it when deciding on vacancy postings.

- The social planner's equation does not include a wage term, as the planner internalizes the labor market dynamics and does not need to consider wages explicitly. Instead, the planner directly optimizes the allocation of labor and capital without facing a market wage constraint.

We can rewrite these equations substituting Q_t and $M_c(\cdot)$:

$$Q_t = \frac{\text{Matches}}{\text{Vacancies}} = \frac{\chi V_t^{\epsilon} S_t^{1-\epsilon}}{V_t} = \chi \left(\frac{S_t}{V_t}\right)^{1-\epsilon}$$

$$M_v(\cdot) = \chi\epsilon \left(\frac{S_t}{V_t}\right)^{1-\epsilon}$$

Remark: Remember that we make this substitution in equilibrium **after** we have already computed the FOCs, not before!

After the substitution:

$$\frac{\phi_n}{\chi\left(\frac{S_t}{V_t}\right)^{1-\epsilon}} = A_t \left(\frac{K_t}{N_t}\right)^{\alpha} (1-\alpha) - W_t + \beta \mathbb{E}_t \left[\frac{\lambda_{t+1}}{\lambda_t}(1-\delta_n)\frac{\phi_n}{\chi\left(\frac{S_{t+1}}{V_{t+1}}\right)^{1-\epsilon}}\right] \quad \text{(Decentralized)}$$

$$\frac{\phi_n}{\chi\epsilon\left(\frac{S_t}{V_t}\right)^{1-\epsilon}} = A_t \left(\frac{K_t}{N_t}\right)^{\alpha} (1-\alpha) + \beta \mathbb{E}_t \left[\frac{\lambda_{t+1}}{\lambda_t}(1-\delta_n)\frac{\phi_n}{\chi\epsilon\left(\frac{S_{t+1}}{V_{t+1}}\right)^{1-\epsilon}}\right] \quad \text{(Social Planner)}$$

In a decentralized economy, individual firms do not consider the broader impact of their hiring decisions on the labor market. Specifically:

- **Crowding in the Labor Market**: When a firm in a decentralized economy posts an additional vacancy, it doesn't account for how this increases competition for workers, making it harder for each vacancy to be filled. As more firms post vacancies, the probability of successfully filling any given vacancy decreases, leading to a "crowded" labor market. However, because individual firms do not internalize this effect, they perceive the cost of posting vacancies to be lower than it truly is at a societal level. By contrast, the social planner recognizes this effect and internalizes it, effectively perceiving a higher cost of hiring. This is reflected in the planner's condition, where $\epsilon < 1$ captures the diminishing returns in matching efficiency as the labor market becomes more saturated.

- **Private vs. Social Benefits of Labor**: In the decentralized economy, the firm's private benefit from hiring additional labor (i.e., marginal product of labor

- MPL) is reduced by the wage it must pay, which limits its incentive to hire relative to the socially optimal level. The social planner, however, considers the full marginal product of labor, without needing to "compensate" for wages, since it views labor from a collective welfare perspective. As a result, the planner values additional labor more highly, recognizing the full social benefit of hiring.

The Social Planner's allocation is **efficient**, whereas the decentralized economy is not, except in one particular case if

$$W_t = (1 - \epsilon)\,\text{MPL}_t$$

i.e., when the firm's perceived marginal benefit of labor equals the social marginal benefit.

In this scenario, the decentralized firm would effectively "internalize" the social benefit of hiring additional labor, as if it were considering the welfare impact of each hiring decision on the entire economy. How to achieve this? Read the section "**Nash Bargaining of Wages**" in the notes uploaded on Canvas.