Computational Tools for Macroeconomics using MATLAB

Week 10 – Stochastic Dynamics & Business Cycles

Cristiano Cantore

Sapienza University of Rome

Learning Outcomes

By the end of this week, you will be able to:

1. Formulate a stochastic neoclassical growth model.
2. Represent productivity shocks as an AR(1) process.
3. Discretise the shock process using Tauchen's method.
4. Solve the model using policy function iteration (Coleman operator) and the endogenous grid method (EGM).
5. Simulate business cycle dynamics from the solved model.

From Deterministic to Stochastic

- So far, we assumed perfect foresight (no uncertainty).
- But the real world is volatile: GDP fluctuates (Business Cycles).
- We introduce **Aggregate Productivity Shocks** (z_t).
- The Production Function becomes:

$$y_t = e^{z_t} k_t^{\alpha}$$

- z_t evolves randomly over time.

The Shock Process

- We typically model productivity as an AR(1) process in logs:

$$z_{t+1} = \rho z_t + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma_\epsilon^2)$$

- $\rho \in [0, 1)$: Persistence (how long shocks last).
- σ_ϵ: Volatility (size of shocks).
- If $\rho = 0$, shocks are temporary. If $\rho \approx 1$, shocks are very persistent.

The Household's Problem

- Maximize Expected Utility:

$$E_0 \sum_{t=0}^{\infty} \beta^t u(c_t)$$

- Subject to:

$$c_t + k_{t+1} = e^{z_t} k_t^{\alpha} + (1 - \delta) k_t$$

- **Euler Equation** (Intertemporal Optimality):

$$u'(c_t) = \beta E_t \left[u'(c_{t+1}) \left(\underbrace{\alpha e^{z_{t+1}} k_{t+1}^{\alpha-1} + 1 - \delta}_{R_{t+1}} \right) \right]$$

- Note the Expectation Operator E_t: We don't know z_{t+1} yet!

Handling Continuous Shocks

- Computers cannot easily calculate integrals over continuous distributions.
- We "discretize" the shock z_t:
  - Instead of any value in $(-\infty, \infty)$, z_t takes values from a finite grid: $\mathcal{Z} = \{z^1, z^2, \ldots, z^N\}$.
  - Probabilities of moving from z^i to z^j are stored in a Transition Matrix P.
- **Tauchen's Method** (1986) is the standard way to do this.
- It approximates the AR(1) density with a Markov Chain.

Tauchen's Method in MATLAB

- Input: ρ, σ_ϵ, N.
- Output: Grid Z and Matrix P.
- Example ($N = 3$):

$$P = \begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.4 & 0.5 \end{bmatrix}$$

- Row i: Probabilities of going to any state j next period, given we are in state i today.

Understanding `tauchen.m` (1/2)

- Open `tauchen.m`. This helper function creates the grid.
- First, we compute the unconditional standard deviation σ_z:

```
% 1. Calculate the unconditional std dev
sigma_z = sigma_eps / sqrt(1 - rho^2);

% 2. Create the grid Z (+/- m std devs)
z_min = -m * sigma_z;
z_max =  m * sigma_z;
Z = linspace(z_min, z_max, N)';
```

- m determines how wide the grid is (usually $m = 3$).

Understanding `tauchen.m` (2/2)

- Then, we compute probabilities using the Normal CDF.
- We calculate the probability of landing in the interval around Z_j, conditional on starting at Z_i.

```
% 3. Compute P(i, j)
for i = 1:N
    for j = 1:N
        % Probability of being in interval around Z(j)
        % given current state Z(i)
        prob = normcdf(...) - normcdf(...);
        P(i, j) = prob;
    end
end
```

The Policy Function

- We want to find a rule: $k_{t+1} = g(k_t, z_t)$.
- This tells us how much to save for any combination of capital and productivity.
- Since z_t is discrete, we really find N functions:

$$k'(k, z^1), \quad k'(k, z^2), \quad \ldots \quad k'(k, z^N)$$

Method 1: Policy Function Iteration (Coleman)

1. Create a grid for capital K.
2. Guess a policy function $k'_{old}(k, z)$.
3. For each (k, z), find the value k' that satisfies the Euler Equation:

$$u'(c) = \beta \sum_{j=1}^{N} P_{z,j} \cdot u'(c'_j) \cdot R'_j$$

   where c'_j comes from the *guessed* policy $k'_{old}(k', z^j)$.
4. Since k' appears inside $u'(\cdot)$, we need a **root-finder** (like `fzero`) at every point.
5. Update guess k'_{new} and repeat until convergence.

Understanding `coleman.m` (1/2)

- This script implements **Time Iteration** on the Euler Equation.
- The core logic finds k' to set the Euler Residual to zero:

```
% Define residual function
euler_resid = @(k_next) solve_euler(k_next, resource, ...);

% Find root
k_opt = fzero(euler_resid, [lb, ub]);
```

- Notice we use `fzero` inside the loops over K and Z. This is why it's slow!

Understanding `coleman.m` (2/2)

- The residual function calculates:

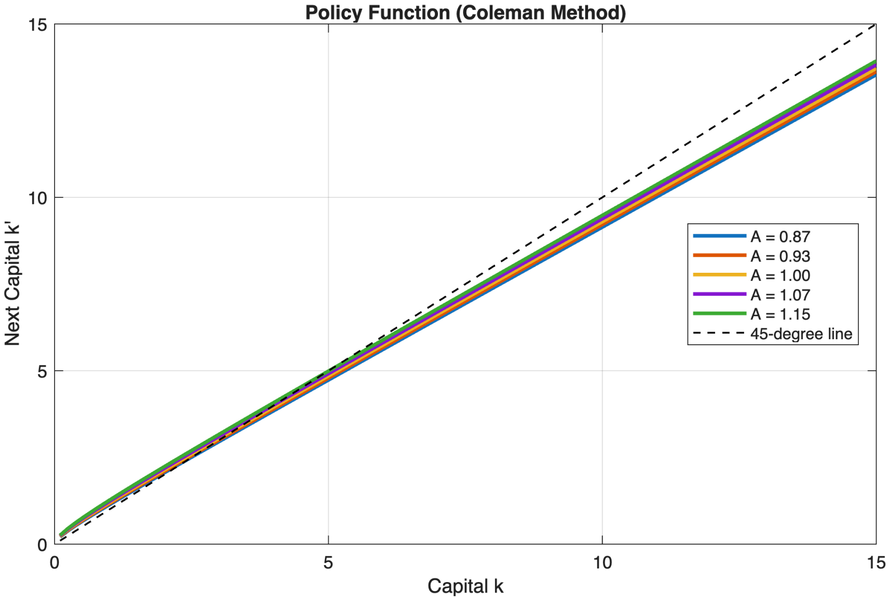
$$\text{Resid} = u'(c) - \beta E[u'(c')R']$$

- To evaluate the expectation $E[\cdot]$, we need the future policy function:

```
% Interpolate future policy k'' = g(k', z')
k_next2 = interp1(K_grid, K_policy_future(:, iz_next), ...
                  k_next, 'linear');
```

- **Action:** Run `coleman.m`. Observe convergence and the Policy Function plot.

Coleman Results



Policy function $k'(k, z)$ for different productivity levels.

Method 2: Endogenous Grid Method (EGM)

- Root-finding is slow. EGM (Carroll, 2006) avoids it.
- **Idea:** Don't fix k and find k'.
- Instead: **Fix k' and find k.**
- Why? The Euler equation is invertible!

$$u'(c) = \text{RHS}(k') \implies c = (u')^{-1}(\text{RHS})$$

- Once we have c, we find k from the budget constraint:

$$k \text{ comes from } e^z k^\alpha + (1-\delta)k = c + k'$$

EGM Algorithm

1. Grid K represents *future* capital k'.
2. Compute expectations (RHS of Euler) for each k'.
3. Invert marginal utility to find implied c.
4. Use resource constraint to find endogenous k today.
5. Interpolate to get $k'(k)$ back on the fixed grid.

Result: 20-50x faster than Coleman method!

Understanding `egm.m` (1/2)

- The Endogenous Grid Method inverts the Euler equation.
- Step 1: Calculate Expectation of RHS (using future k' grid):

```
% Expected marginal utility for each k' choice
Expected_RHS = Expected_RHS + prob * Mu_old(:, iz_next) .* R_next;
```

- Step 2: Invert Euler to find Consumption today:

```
c_implied = u_inv_prime(beta * Expected_RHS);
```

Understanding `egm.m` (2/2)

- Step 3: Find the endogenous *k* today from budget constraint:

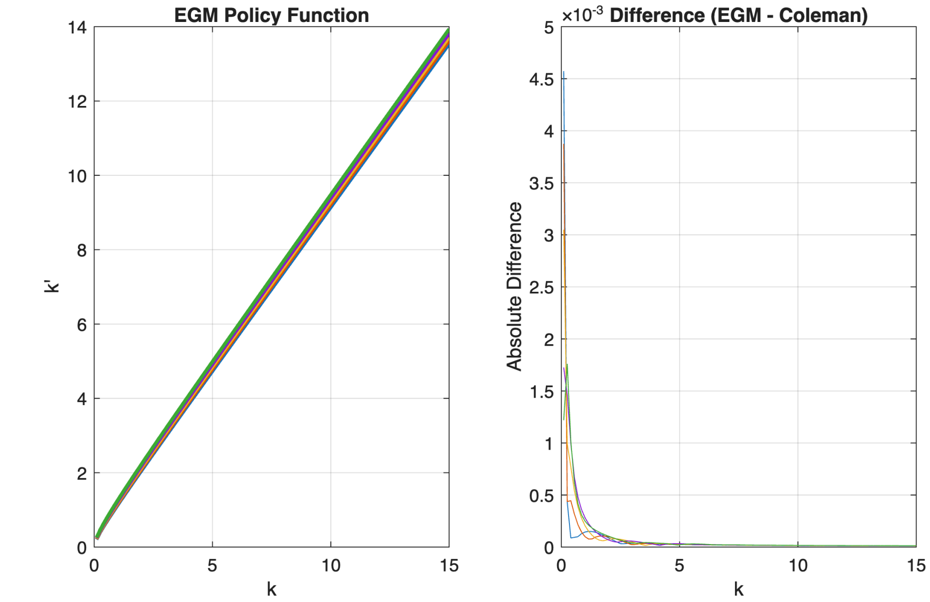
```
Total_Assets_Required = c_implied + K_grid;
% Solve: Resource(k_endo) = Total_Assets
% ... (fzero finding k_endo) ...
```

- Step 4: Interpolate back to fixed grid:

```
K_policy_new(:, iz) = interp1(k_endo, K_grid, K_grid, 'linear');
```

- **Action:** Run `egm.m`. Compare speed with Coleman!

EGM Results



Comparison: EGM vs Coleman (almost identical, but EGM is much faster).

Simulation

- Now we can start with the simulation.
- We will use the policy function to simulate time series.
- We will generate random shocks and feed them into $k'(k, z)$.
- We will plot the time series of Productivity, Output, Consumption, Investment.
- We will calculate the Business Cycle Statistics (std dev, corr).

Simulating the Economy

- Once we have the policy $k'(k, z)$, we can simulate history.
- **Steps:**
  1. Generate a sequence of shocks using P (random numbers).
  2. Start with $k_0 = k_{ss}$.
  3. Iterate: $k_{t+1} = k'(k_t, z_t)$.
  4. Compute y_t, c_t, i_t.
- We can then compute standard deviations and correlations (Business Cycle Statistics) to match real data.

Simulation Code: `simulation_stoch.m` (1/2)

- First, we load the solved policy function and simulate shocks.
- We use the Cumulative Distribution of P to draw the next state.

```
% 1. Load Policy
load('policy_egm.mat'); % or coleman

% 2. Simulate Shocks
P_cum = cumsum(P, 2);
for t = 1:T-1
    r = rand;
    % Find next state index where CDF > random number
    z_idx(t+1) = find(P_cum(z_idx(t), :) >= r, 1);
end
```

Simulation Code: `simulation_stoch.m` (2/2)

- Then we iterate the economy forward using the Policy Function.

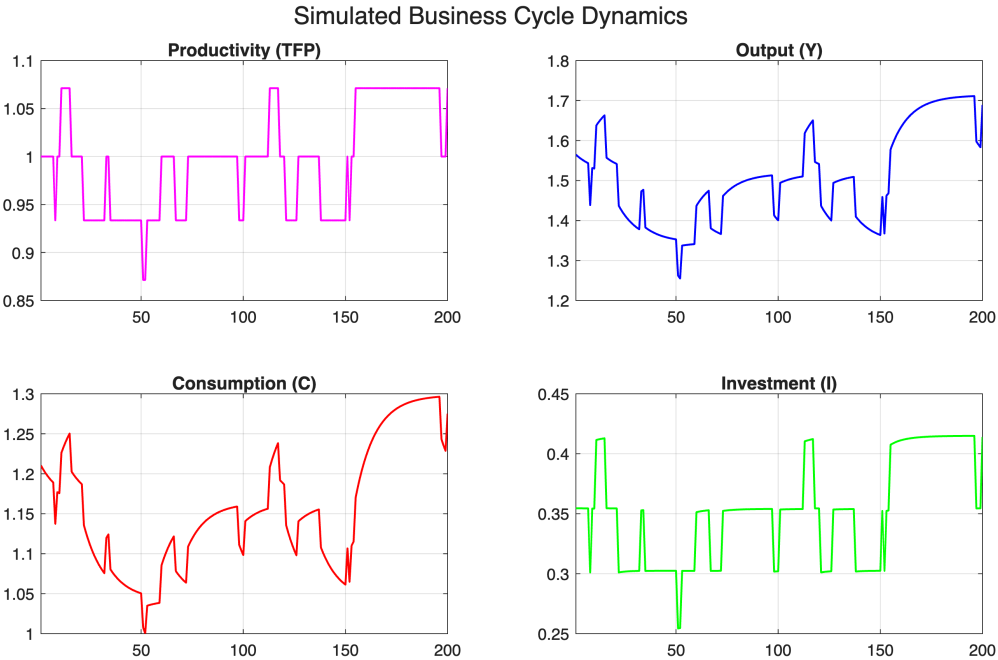
```
for t = 1:T
    % Get Policy k' = g(k, z)
    k_next = interp1(K_grid, K_policy(:, z_curr), ...
                     k_curr, 'linear');

    % Calculate other variables
    Y_sim(t) = A_curr * k_curr^alpha;
    I_sim(t) = k_next - (1-delta)*k_curr;
    C_sim(t) = Y_sim(t) - I_sim(t);

    % Update State
    K_sim(t+1) = k_next;
end
```

- **Action:** Run `simulation_stoch.m` and inspect the plots and statistics.

Simulation Results



Challenge: Precautionary Savings

Question

Does uncertainty make households save more or less?

- **Task:**
  1. Solve the model with NO uncertainty ($\sigma_\epsilon = 0$).
  2. Solve the model with HIGH uncertainty ($\sigma_\epsilon = 0.1$).
  3. Plot the policy function $k'(k)$ for the mean state ($z = 0$) for both cases.
- Use the `egm.m` file as a starting point.
- What do you observe?

Homework: Precautionary Savings Analysis

- **Task 1: Quantifying Precautionary Savings**
  - Solve the model with $\sigma_\epsilon \approx 0$ (deterministic) and $\sigma_\epsilon = 0.04$ (moderate uncertainty).
  - Simulate both economies for 1000 periods (with 100 periods of burn-in).
  - Calculate the long-run average capital stock $\bar{k}$ for each case.
  - Compute the "Precautionary Savings Premium": $\frac{\bar{k}_{stoch} - \bar{k}_{det}}{\bar{k}_{det}} \times 100\%$.
- **Task 2: The Role of Risk Aversion**
  - Fix $\sigma_\epsilon = 0.04$ and solve for $\sigma = 1$ (log utility) and $\sigma = 5$ (high risk aversion).
  - Plot both policy functions $k'(k)$ for the mean productivity state on the same graph.
  - Simulate both and compare mean capital stocks.
  - Discuss: How does risk aversion amplify precautionary savings?
- **Submission:** A single ZIP file with your code and Figures.