Computational Tools for Macroeconomics using MATLAB

Week 8 – Dynamic Programming & Value Function Iteration

Cristiano Cantore

Sapienza University of Rome

Learning Outcomes

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

1. Understand the Bellman equation formulation for dynamic economic problems.
2. Implement value function iteration (VFI) in MATLAB.
3. Discretise the state space for computation.
4. Simulate optimal decision rules from the computed policy function.
5. Apply VFI to the deterministic neoclassical growth model.

Recap: The Solow Model (Week 7)

- In the Solow model, the savings rate s is **exogenous**.
- Capital evolves mechanically:

$$k_{t+1} = (1-\delta)k_t + sf(k_t)$$

- Consumption is determined residually:

$$c_t = (1-s)f(k_t)$$

- No optimization — households do not choose how much to save.

Question: What if saving were *a choice*?

Introducing Intertemporal Choice

- Agents decide **how to allocate output** between consumption and investment:

$$c_t + k_{t+1} = f(k_t) + (1 - \delta)k_t$$

- Objective: maximize lifetime utility

$$\max_{\{c_t, k_{t+1}\}} \sum_{t=0}^{\infty} \beta^t u(c_t)$$

- Choice variables: c_t, k_{t+1}; state variable: k_t
- Parameter β captures **impatience**.

Introducing Intertemporal Choice

- Agents decide **how to allocate output** between consumption and investment:

$$c_t + k_{t+1} = f(k_t) + (1 - \delta)k_t$$

- Objective: maximize lifetime utility

$$\max_{\{c_t, k_{t+1}\}} \sum_{t=0}^{\infty} \beta^t u(c_t)$$

- Choice variables: c_t, k_{t+1};   state variable: k_t
- Parameter β captures **impatience**.

Trade-off

Higher c_t today $\Rightarrow$ lower k_{t+1} tomorrow $\Rightarrow$ less future output.

Economic Intuition

- Each period: choose between
  - **Consumption** $\Rightarrow$ utility today
  - **Saving (investment)** $\Rightarrow$ utility tomorrow
- Saving transfers resources to the future, but with diminishing returns.
- The optimal choice balances:

$$\text{Marginal utility today} = \beta \times \text{Expected marginal utility tomorrow.}$$

From Fixed s to Optimal s_t

- In the Solow model, s is fixed: households save a constant fraction.
- In the optimization model, s_t is **endogenous**:

$$s_t = 1 - \frac{c_t}{f(k_t)}$$

- The saving rate now depends on preferences (β) and technology (α, δ).
- This leads naturally to a **dynamic optimization** framework.

Next: The Robinson Crusoe model formalizes this idea.

Introducing the Robinson Crusoe Economy

- A single agent (Robinson Crusoe) produces, consumes, and saves.
- He has capital k_t, produces output $f(k_t)$, and decides how much to:
  - **consume** c_t
  - **save / invest** $k_{t+1} - (1-\delta)k_t$
- The resource constraint is:

$$c_t + k_{t+1} = f(k_t) + (1-\delta)k_t$$

- Crusoe values consumption over time according to

$$\max_{\{c_t, k_{t+1}\}} \sum_{t=0}^{\infty} \beta^t u(c_t)$$

Introducing the Robinson Crusoe Economy

- A single agent (Robinson Crusoe) produces, consumes, and saves.
- He has capital k_t, produces output $f(k_t)$, and decides how much to:
  * **consume** c_t
  * **save / invest** $k_{t+1} - (1-\delta)k_t$
- The resource constraint is:

$$c_t + k_{t+1} = f(k_t) + (1-\delta)k_t$$

- Crusoe values consumption over time according to

$$\max_{\{c_t, k_{t+1}\}} \sum_{t=0}^{\infty} \beta^t u(c_t)$$

Goal: find how much to consume today vs save for tomorrow.

The Dynamic Optimization Problem

- The control variables are **current consumption** c_t and **next-period capital** k_{t+1}.
- The state variable is the current capital stock k_t.
- The transition equation links today and tomorrow:

$$k_{t+1} = f(k_t) + (1 - \delta)k_t - c_t$$

- Crusoe chooses $\{c_t, k_{t+1}\}$ to maximize utility subject to this constraint and $k_t \geq 0$.

The Dynamic Optimization Problem

- The control variables are **current consumption** c_t and **next-period capital** k_{t+1}.
- The state variable is the current capital stock k_t.
- The transition equation links today and tomorrow:

$$k_{t+1} = f(k_t) + (1-\delta)k_t - c_t$$

- Crusoe chooses $\{c_t, k_{t+1}\}$ to maximize utility subject to this constraint and $k_t \geq 0$.

Key idea

Today's decision affects tomorrow's possibilities $\Rightarrow$ we need a **recursive** formulation.

Recursive Formulation and the Value Function

- Define the **value function**:

$$V(k_t) = \max_{k_{t+1}} \left\{ u(f(k_t) + (1-\delta)k_t - k_{t+1}) + \beta V(k_{t+1}) \right\}$$

- The term in braces captures two elements:
  - $u(\cdot)$: current utility from consuming c_t
  - $\beta V(k_{t+1})$: discounted future value
- This equation defines a mapping $T[V] = V$ — the **Bellman equation**.

Recursive Formulation and the Value Function

- Define the **value function**:

$$V(k_t) = \max_{k_{t+1}} \{u(f(k_t) + (1-\delta)k_t - k_{t+1}) + \beta V(k_{t+1})\}$$

- The term in braces captures two elements:
  - $u(\cdot)$: current utility from consuming c_t
  - $\beta V(k_{t+1})$: discounted future value
- This equation defines a mapping $T[V] = V$ — the **Bellman equation**.

Principle of Optimality

An optimal plan today must contain an optimal plan for all future periods.

Economic Interpretation of the Bellman Equation

- The value function $V(k)$ represents the **maximum lifetime utility** from starting with capital k.
- We now switch notation from k_{t+1} to k' and from k_t to k.
- The choice variable k' determines:
  - **Consumption today:** $c = f(k) + (1 - \delta)k - k'$
  - **Future wellbeing:** $\beta V(k')$
- The agent chooses k' to balance:

$$\text{marginal utility today} = \beta \times \text{marginal value of capital tomorrow.}$$

From Bellman Equation to Euler Equation

- Start from the Bellman equation:

$$V(k) = \max_{k'} \left\{ u(f(k) + (1-\delta)k - k') + \beta V(k') \right\}$$

- First-order condition with respect to k':

$$-u'(c) + \beta V'(k') = 0 \quad \Rightarrow \quad u'(c) = \beta V'(k')$$

- Using the **envelope theorem** on $V(k)$:

$$V'(k) = u'(c) \left[f'(k) + (1-\delta) \right]$$

- Combining: $V'(k') = u'(c')[f'(k') + (1-\delta)]$

From Bellman Equation to Euler Equation

- Start from the Bellman equation:

$$V(k) = \max_{k'} \{u(f(k) + (1-\delta)k - k') + \beta V(k')\}$$

- First-order condition with respect to k':

$$-u'(c) + \beta V'(k') = 0 \quad \Rightarrow \quad u'(c) = \beta V'(k')$$

- Using the **envelope theorem** on $V(k)$:

$$V'(k) = u'(c)\left[f'(k) + (1-\delta)\right]$$

- Combining: $V'(k') = u'(c')[f'(k') + (1-\delta)]$

Standard Euler Equation

$$u'(c_t) = \beta u'(c_{t+1})\left[f'(k_{t+1}) + (1-\delta)\right]$$

Intuition: Marginal utility today equals discounted marginal utility tomorrow times the return on capital.

Example: Log Utility and Cobb–Douglas Production

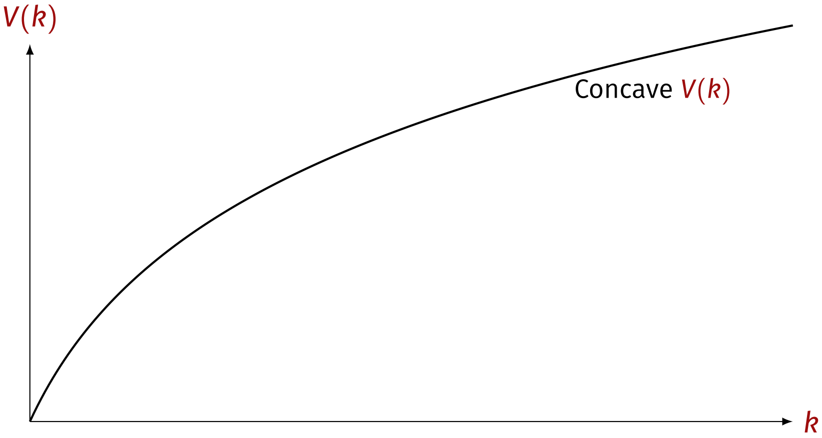
- Let $u(c) = \ln c, \quad f(k) = k^{\alpha}$.
- Parameters: $\alpha = 0.4$, $\beta = 0.95$, $\delta = 0.1$.
- The recursive problem becomes:

$$V(k) = \max_{k'} \left\{ \ln\left(k^{\alpha} + (1-\delta)k - k'\right) + \beta V(k') \right\}.$$

- Analytical solution is hard $\Rightarrow$ we will solve it **numerically**.

Next: visual intuition for the Bellman equation.

Visualizing the Value Function



Intuition: more capital increases lifetime utility, but with diminishing returns.

The Bellman Equation as a Recursive Problem

- Any dynamic optimization problem can be written as:

$$V(k) = \max_{k'} \left\{ u(f(k) - k') + \beta V(k') \right\}$$

- This is a **fixed point** problem:

$$V = T(V),$$

  where T is the **Bellman operator**.
- Iterating on T will eventually reach the true value function V^*.

The Bellman Equation as a Recursive Problem

- Any dynamic optimization problem can be written as:

$$V(k) = \max_{k'} \left\{ u(f(k) - k') + \beta V(k') \right\}$$

- This is a **fixed point** problem:

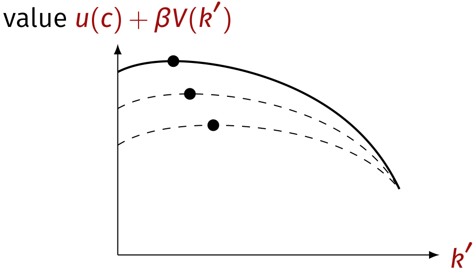
$$V = T(V),$$

  where T is the **Bellman operator**.
- Iterating on T will eventually reach the true value function V^*.

Intuition

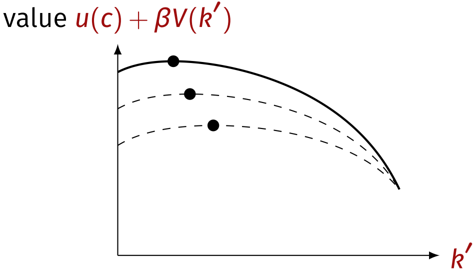
Each iteration updates our guess of how much future value matters relative to current utility.

Graphical Intuition: The Bellman Equation



- **Each curve** shows the objective function $u(c) + \beta V(k')$ for a **fixed** current capital k.
- For each k, the agent chooses k' to maximize this curve.
- **The trade-off:** Higher k' means lower consumption today ($c = f(k) + (1 - \delta)k - k'$) but higher future value $\beta V(k')$.
- **The dots** mark the optimal choice $k'^*(k)$ for each k.
- **The envelope** of these maxima gives the value function $V(k)$.

Graphical Intuition: The Bellman Equation



- **Each curve** shows the objective function $u(c) + \beta V(k')$ for a **fixed** current capital k.
- For each k, the agent chooses k' to maximize this curve.
- **The trade-off:** Higher k' means lower consumption today ($c = f(k) + (1 - \delta)k - k'$) but higher future value $\beta V(k')$.
- **The dots** mark the optimal choice $k'^*(k)$ for each k.
- **The envelope** of these maxima gives the value function $V(k)$.

Different current capital levels (k) shift the trade-off, leading to different optimal savings choices (k').

How Value Function Iteration Works (Step-by-Step)

1. Start from a guess $V_0(k)$.
2. For each k_i, compute

$$V_1(k_i) = \max_{k'} \{u(f(k_i) - k') + \beta V_0(k')\}.$$

3. Repeat:

$$V_{t+1}(k_i) = \max_{k'} \{u(f(k_i) - k') + \beta V_t(k')\}.$$

4. Stop when $\|V_{t+1} - V_t\| < \varepsilon$.

How Value Function Iteration Works (Step-by-Step)

1. Start from a guess $V_0(k)$.
2. For each k_i, compute

$$V_1(k_i) = \max_{k'} \{u(f(k_i) - k') + \beta V_0(k')\}.$$

3. Repeat:

$$V_{t+1}(k_i) = \max_{k'} \{u(f(k_i) - k') + \beta V_t(k')\}.$$

4. Stop when $||V_{t+1} - V_t|| < \varepsilon$.

Contraction Mapping

$T[V]$ is a contraction $\Rightarrow$ convergence to a unique V^*.

Why the Value Function is Concave

- Utility $u(c)$ is concave $\Rightarrow$ diminishing marginal utility.
- Production $f(k)$ is concave $\Rightarrow$ diminishing returns.
- The combination implies $V(k)$ is also concave and increasing.

Why the Value Function is Concave

- Utility $u(c)$ is concave $\Rightarrow$ diminishing marginal utility.
- Production $f(k)$ is concave $\Rightarrow$ diminishing returns.
- The combination implies $V(k)$ is also concave and increasing.

Economic Meaning

As capital grows, each extra unit of k adds less to lifetime welfare.

Takeaways from the Graphical Approach

- The Bellman equation expresses the same economics as the intertemporal Euler equation—but recursively.
- $V(k)$ summarizes the best lifetime utility from each k.
- Value Function Iteration (VFI):
  - starts from a guess V_0
  - applies the Bellman operator
  - converges to V^*
- Once V^* is known, we can extract the **policy function** $k'(k)$.

Next: Implement VFI step-by-step in MATLAB.

Numerical Setup for the Growth Model

- We solve numerically the Bellman equation:

$$V(k) = \max_{k'} \{ u(f(k) + (1-\delta)k - k') + \beta V(k') \}$$

- Functional forms:

$$f(k) = k^{\alpha}, \quad u(c) = \ln(c)$$

- Parameters (example):

$$\alpha = 0.4, \quad \beta = 0.95, \quad \delta = 0.1$$

- State space: grid for $k \in [k_{\min}, k_{\max}]$ with N points.

Numerical Setup for the Growth Model

- We solve numerically the Bellman equation:

$$V(k) = \max_{k'} \{ u(f(k) + (1-\delta)k - k') + \beta V(k') \}$$

- Functional forms:

$$f(k) = k^{\alpha}, \quad u(c) = \ln(c)$$

- Parameters (example):

$$\alpha = 0.4, \quad \beta = 0.95, \quad \delta = 0.1$$

- State space: grid for $k \in [k_{\min}, k_{\max}]$ with N points.

Goal: compute $V(k)$ and the policy rule $k'(k)$.

Algorithm Outline

1. Define parameter values and create the grid.
2. Compute all feasible consumption levels:

$$c_{ij} = f(k_i) + (1 - \delta)k_i - k_j'$$

   If $c_{ij} \leq 0 \Rightarrow u(c_{ij}) = -\infty$.
3. Initialize value function $V_0(k_i) = 0$.
4. Apply the Bellman operator:

$$V_{t+1}(k_i) = \max_j \{u(c_{ij}) + \beta V_t(k_j)\}$$

5. Iterate until convergence.

Algorithm Outline

1. Define parameter values and create the grid.
2. Compute all feasible consumption levels:

$$c_{ij} = f(k_i) + (1-\delta)k_i - k_j'$$

   If $c_{ij} \leq 0 \Rightarrow u(c_{ij}) = -\infty$.
3. Initialize value function $V_0(k_i) = 0$.
4. Apply the Bellman operator:

$$V_{t+1}(k_i) = \max_j \{u(c_{ij}) + \beta V_t(k_j)\}$$

5. Iterate until convergence.

Outputs

Optimal $k'(k)$ and the corresponding value function $V^*(k)$.

Pseudocode for VFI

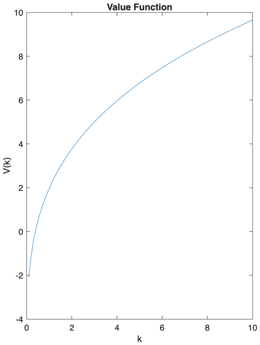
```
% 1. Parameters and grid
alpha = 0.4; beta = 0.95; delta = 0.1;
kmin = 0.1; kmax = 10; N = 100;
kgrid = linspace(kmin,kmax,N)';

% 2. Utility matrix
f = kgrid.^alpha;
cons = f + (1-delta)*kgrid - kgrid';  % matrix c_{ij}
util = -inf(N);
feas = cons > 0;
util(feas) = log(cons(feas));

% 3. Value function iteration
V = zeros(N,1); diff = 1;
while diff > 1e-6
    Vnew = zeros(N,1);
    for i = 1:N
        [Vnew(i), pol_ind(i)] = max(util(i,:) + beta*V');
    end
    diff = max(abs(Vnew - V));
    V = Vnew;
end
```

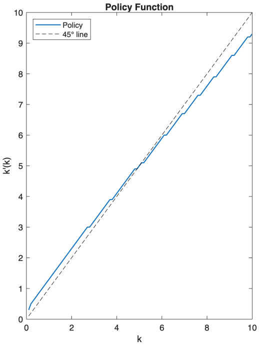
Extracting Policy and Plotting Results

```
% 4. Policy function and value function
k_policy = kgrid(pol_ind);

% 5. Plot
figure;
subplot(1,2,1)
plot(kgrid, V)
title('Value Function'); xlabel('k'); ylabel('V(k)')

subplot(1,2,2)
plot(kgrid, k_policy, 'LineWidth',1.2)
hold on; plot(kgrid, kgrid, '--k')
title('Policy Function'); xlabel('k'); ylabel('k''(k)')
legend('Policy','45° line','Location','NorthWest')
```

Interpretation: the intersection with the 45° line gives the steady state.

Results from Value Function Iteration





Value Function (left):

- Concave and increasing
- Negative at low k (tight consumption constraint)
- Diminishing marginal value of capital

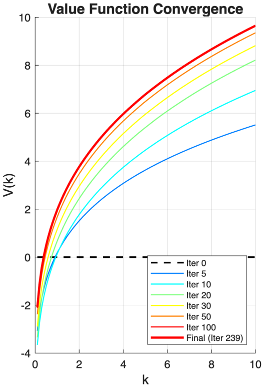
Policy Function (right):

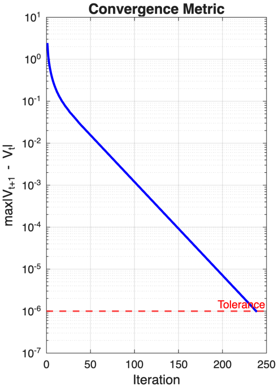
- Crosses 45° line at steady state $k^* \approx 5$
- Below k^*: $k'(k) > k$ → capital accumulates
- Above k^*: $k'(k) < k$ → capital decumulates
- Global stability toward k^*

Code: Visualizing Convergence

```
% Store value functions at selected iterations
V = zeros(N,1); diff = 1; iter = 0;
V_history = zeros(N,10); % store every 10th iteration
iter_save = 1;

while diff > 1e-6
    Vnew = zeros(N,1);
    for i = 1:N
        [Vnew(i), ~] = max(util(i,:) + beta*V');
    end
    diff = max(abs(Vnew - V));
    V = Vnew;
    iter = iter + 1;

    % Save every 10 iterations
    if mod(iter,10)==0 && iter_save <= 10
        V_history(:,iter_save) = V;
        iter_save = iter_save + 1;
    end
end
```

Visualizing Convergence





Observations:

- Value function starts at $V_0(k) = 0$
- Early iterations refine quickly
- Convergence slows near solution
- All curves converge to $V^*(k)$
- Typically converges in 50-100 iterations

Convergence criterion:

$$\max_i |V_{t+1}(k_i) - V_t(k_i)| < 10^{-6}$$

Economic Interpretation

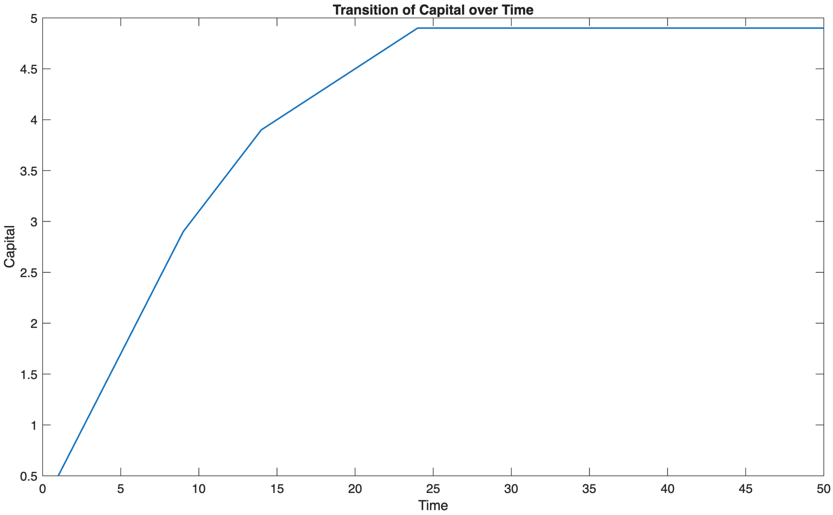
- Policy function $k'(k)$ tells how much to save for each k.
- Where $k'(k) = k \Rightarrow$ steady state capital k^*.
- Comparative statics:
    - Higher $\beta \rightarrow$ more patient $\rightarrow$ higher k^*.
    - Lower $\alpha \rightarrow$ lower returns to capital $\rightarrow$ smaller k^*.
- The shape of $V(k)$ reflects lifetime well-being across capital levels.

Next: simulate dynamics using the computed policy rule.

Simulating the Capital Path

```
% Simulate transition dynamics
T = 50;                         % number of periods
kpath = zeros(T,1);
kpath(1) = 0.5;                 % initial capital

for t = 1:T-1
    % interpolate policy to get next capital
    kpath(t+1) = interp1(kgrid, k_policy, kpath(t));
end


% Plot transition
figure;
plot(1:T, kpath, 'LineWidth',1.2)
xlabel('Time'); ylabel('Capital')
title('Transition of Capital over Time')
```

Capital Path from Value Function Iteration



Economic Interpretation of the Simulation

- Starting from low k_0, Crusoe saves more to reach k^*.
- Once near steady state, $k' \approx k$ and consumption stabilizes.
- If β increases (more patience):
  - Policy shifts upward $\Rightarrow$ higher steady-state k^*.
- If α decreases (lower productivity):
  - Policy flattens $\Rightarrow$ lower steady-state k^*.

Key Takeaways from Week 8

- Dynamic programming transforms multi-period optimization into a recursive **Bellman equation**.
- The **Value Function Iteration (VFI)** algorithm finds the fixed point $V^*(k)$ and the optimal decision rule $k'(k)$.
- The **policy function** encodes saving behaviour for each capital level.
- Simulating k_t using $k'(k)$ shows convergence to steady state k^*.

Next: extend the simulation and interpret results.

In-Class Challenge: Simulating Dynamics

Objective

Use the policy function computed in class to **simulate and interpret** the transition of the economy.

- Start from $k_0 = 0.5$ and use $k'(k_t)$ to simulate c_t for $T = 50$ periods:

$$k_{t+1} = k'(k_t), \quad c_t = f(k_t) + (1 - \delta)k_t - k_{t+1}.$$

- Plot k_t and c_t over time.
- Verify that k_t converges to the steady state k^*.

In-Class Challenge: different calibration

- Test how parameters affect dynamics:

$$\beta \in \{0.90, 0.95, 0.98\}, \qquad \alpha \in \{0.25, 0.30, 0.40\}.$$

- For each case:
  1. Run the simulation.
  2. Plot the new policy function and capital path.
  3. Comment briefly on the economic implications.
- Add comments in your script explaining intuition: e.g. "Higher β → more saving → higher steady state."

Deliverable: figures + short comments in your MATLAB file.

Homework: From Aggregate to Per-Worker Variables

- Aggregate technology: $Y_t = F(K_t) = K_t^{\alpha}$, $\alpha \in (0, 1)$.
- Capital depreciates at rate δ, population grows at rate $n > 0$:

$$L_{t+1} = (1 + n)\, L_t.$$

- Per-worker variables: $k_t \equiv K_t/L_t$, $c_t \equiv C_t/L_t$, $y_t \equiv Y_t/L_t = f(k_t) = k_t^{\alpha}$.
- Per-worker resource constraint:

$$c_t + (1 + n)k_{t+1} = f(k_t) + (1 - \delta)k_t.$$

Interpretation: with population growth, tomorrow's capital per worker must be *scaled up* by $(1 + n)$.

Dynamic Problem and Bellman Equation

- Planner's problem (per worker):

$$\max_{\{c_t, k_{t+1}\}_{t\geq 0}} \sum_{t=0}^{\infty} \beta^t u(c_t), \quad u(c) = \ln c$$

$$\text{s.t.} \quad c_t + (1+n)k_{t+1} = f(k_t) + (1-\delta)k_t, \quad k_0 \text{ given.}$$

- Bellman equation (state k, control k'):

$$V(k) = \max_{k'} \Big\{ u\big(f(k) + (1-\delta)k - (1+n)k'\big) + \beta V(k') \Big\}.$$

- Feasibility: $c = f(k) + (1-\delta)k - (1+n)k' > 0$.

Euler Equation and Economic Intuition

- First-order condition (Euler equation) under interior solutions:

$$u'(c_t) = \beta\, u'(c_{t+1})\, \frac{f'(k_{t+1}) + 1 - \delta}{1+n}.$$

- With $u(c) = \ln c \Rightarrow u'(c) = 1/c$:

$$\frac{1}{c_t} = \beta\, \frac{1}{c_{t+1}}\, \frac{f'(k_{t+1}) + 1 - \delta}{1+n}.$$

- **Steady state** ($k_{t+1} = k_t = k^*$ and $c_{t+1} = c_t = c^*$):

$$1 = \beta\, \frac{f'(k^*) + 1 - \delta}{1+n} \;\Rightarrow\; f'(k^*) = \frac{1+n}{\beta} - (1-\delta).$$

- For $f(k) = k^\alpha$: $\alpha\, {k^*}^{\alpha-1} = \dfrac{1+n}{\beta} - (1-\delta)$.

Euler Equation and Economic Intuition

- First-order condition (Euler equation) under interior solutions:

$$u'(c_t) \;=\; \beta\, u'(c_{t+1})\, \frac{f'(k_{t+1}) + 1 - \delta}{1+n}.$$

- With $u(c) = \ln c \Rightarrow u'(c) = 1/c$:

$$\frac{1}{c_t} \;=\; \beta\, \frac{1}{c_{t+1}}\, \frac{f'(k_{t+1}) + 1 - \delta}{1+n}.$$

- **Steady state** ($k_{t+1} = k_t = k^*$ and $c_{t+1} = c_t = c^*$):

$$1 \;=\; \beta\, \frac{f'(k^*) + 1 - \delta}{1+n} \quad\Rightarrow\quad f'(k^*) \;=\; \frac{1+n}{\beta} - (1-\delta).$$

- For $f(k) = k^\alpha$: $\alpha\, {k^*}^{\alpha-1} = \dfrac{1+n}{\beta} - (1-\delta)$.

Comparative statics (per worker)

$\uparrow n \Rightarrow$ higher dilution $\Rightarrow$ lower k^*. $\uparrow \beta \Rightarrow$ more patience $\Rightarrow$ higher k^*. $\uparrow \delta \Rightarrow$ more depreciation $\Rightarrow$ lower k^*.

What Changes Computationally (vs. Class Code)

- Only the **consumption mapping** changes:

$$c_{ij} = f(k_i) + (1-\delta)k_i - (1+n)\,k'_j.$$

- Bellman update and the rest of VFI are unchanged.
- Simulation uses the same resource constraint:

$$k_{t+1} = k'(k_t), \qquad c_t = f(k_t) + (1-\delta)k_t - (1+n)\,k_{t+1}.$$

- Use the same grid $[k_{\min}, k_{\max}]$ to **compare** with $n = 0$.

Deliverables: value/policy plots, simulated paths, parameter experiments, comments.