

# Econ 202A Macroeconomics: Section 2

---

Kiyea Jin

October 29, 31, 2025

## Section 2

---

## 1. Neoclassical Growth Model (a.k.a., Ramsey-Cass-Koopmans Model)

- Phase Diagram
- Numerical Solution: Finite Difference Method + Newton's Method

## 2. Neoclassical Growth Model in Recursive Representation

- Hamilton-Jacobi-Bellman (HJB) equations

## 3. Numerical Solution: Finite Difference Method

- Explicit Method
- Implicit Method

## **Section 2-1:**

## **Neoclassical Growth Model**

---

# Neoclassical Growth Model Overview

- Two endogenous variables  $c(t)$ ,  $k(t)$

- Two dynamic equations:

$$\frac{\dot{c}(t)}{c(t)} = \frac{f'(k(t)) - \rho - \theta g}{\theta}$$

$$\dot{k}(t) = f(k(t)) - c(t) - (n + g)k(t)$$

- Two boundary conditions:

- $k(0)$  given (initial condition)
- Intertemporal budget constraint with equality (terminal condition)

- It is the fact that dynamic system has a terminal condition (rather than full set of initial conditions) that makes the system “forward looking”.

**Figure 1:** Dynamic System (Steinsson 2024)

# Phase Diagram

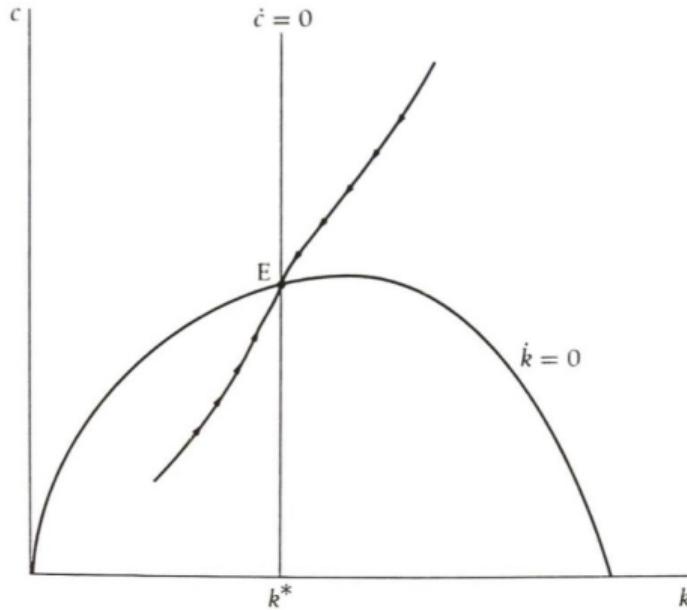


FIGURE 2.5 The saddle path

Source: Romer (2019)

Figure 2: Dynamics of  $c(t)$  and  $k(t)$

- Phase diagram has many paths
  - All of them satisfy the two dynamic equations
  - Which one of these paths will the economy take?
- 
- Answer determined by boundary conditions
    - Boundary condition #1: Initial condition for  $k$
    - But there is no initial condition for  $c$ !!!
    - $c(0)$  is a choice of the household
- 
- So, how do we determine  $c(0)$ ?
    - Boundary condition #2: Intertemporal budget constraint holds with equality

**Figure 3:** Which Path Will the Economy Take? (Steinsson 2024)

# Terminal Boundary Conditions

- Intertemporal budget constraint with equality:

$$\int_0^{\infty} e^{-R(t)} e^{-(n+g)t} c(t) dt = k(0) + \int_0^{\infty} e^{-R(t)} e^{-(n+g)t} [f(k(t)) - (n+g)k(t)] dt \\ - \lim_{t \rightarrow \infty} e^{-R(t)} e^{-(n+g)t} k(t)$$

⇒ When solving models like the Neoclassical Growth Model, numerically with a finite time horizon, imposing that **capital converges to a steady-state level  $k_{ss}$  at the terminal point,  $k(T) = k_{ss}$** , approximates both the transversality condition and the intertemporal budget constraint over an infinite horizon.

## Determination of $c(0)$

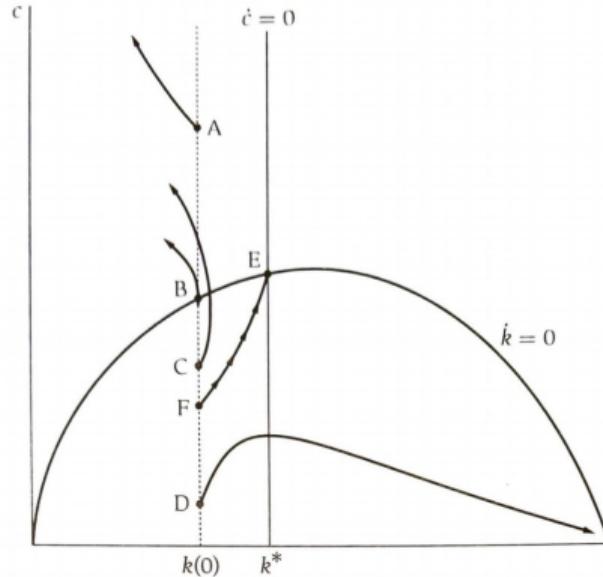


FIGURE 2.4 The behavior of  $c$  and  $k$  for various initial values of  $c$

Source: Romer (2019)

**Figure 4:** Determination of  $c(0)$

## Saddle Path

---

For any positive initial level of  $k$ , there is a unique level of  $c$  that is consistent with households' intertemporal optimization, the dynamics of the capital stock, households' budget constraint, and the requirement that  $k$  not be negative. The function giving this initial  $c$  as a function of  $k$  is known as the saddle path. For any starting value of  $k$ , the initial  $c$  must be the value on the saddle path. (Romer, 2022)

## Exercise: Numerically Solve the Neoclassical Growth Model

Solve the Neoclassical Growth model with the following system of equations:

$$\frac{\dot{c}(t)}{c(t)} = \frac{f'(k(t)) - \rho - \theta g}{\theta} \quad (1)$$

$$\dot{k}(t) = f(k(t)) - c(t) - (n + g)k(t) \quad (2)$$

where the initial condition for capital is  $K_0 = 10$ , and the intertemporal budget constraint with equality is imposed as the terminal condition.

## Key Challenges

1. Solving a **system** of differential equations for both consumption and capital.

## Key Challenges

1. Solving a **system** of differential equations for both consumption and capital.
2. Dealing with both initial and **terminal** boundary conditions.

# Shooting Algorithm

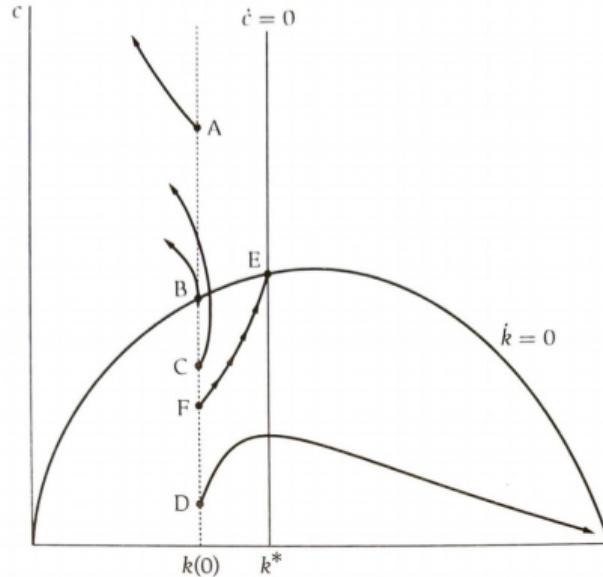


FIGURE 2.4 The behavior of  $c$  and  $k$  for various initial values of  $c$

Source: Romer (2019)

Figure 5: Shooting Algorithm Idea

# Steps of Shooting Algorithm

1. **Guess an Initial Value for Consumption,  $c(0)$ .**

Start by guessing an initial value for consumption,  $c(0)$ . Since  $k(0)$  is already given, you can use both  $k(0)$  and the guessed  $c(0)$  to begin the forward integration.

2. **Solve the System of Equations Forward.**

Using the guessed value of  $c(0)$  and the given  $k(0)$ , solve the system of differential equations (capital accumulation and Euler equation) forward in time from  $t = 0$  to  $t = T$ .

3. **Check the Terminal Condition.**

After the forward integration, check if the computed value of capital  $k(T)$  (or consumption  $c(T)$ ) satisfies the terminal boundary condition.

# Steps of Shooting Algorithm

## 4. Adjust the Guess for $c(0)$ .

If the terminal condition is not satisfied, adjust the initial guess for  $c(0)$  and repeat the process.

Numerical methods like **Newton's method** can be used to iteratively update the guess based on how far the computed terminal value deviates from the desired condition.

## 5. Iterate Until Convergence.

Continue this process of guessing, solving, and checking until the terminal condition is satisfied within a specified tolerance. Once the terminal condition is met, the corresponding value of  $c(0)$  is the correct initial value that leads to a solution where both the initial and terminal conditions are satisfied.

# Finite Difference Approximations

The finite difference approximations are given as:

$$\frac{c(i+1) - c(i)}{\Delta t} = \frac{f'(k(i)) - \rho - \theta g}{\theta} \cdot c(i) \quad (3)$$

$$\frac{k(i+1) - k(i)}{\Delta t} = f(k(i)) - c(i) - (n + g)k(i) \quad (4)$$

where  $i = 1, \dots, l$ ,  $c(i) = c(t_i)$ , and  $k(i) = k(t_i)$  with a uniform time step size  $\Delta t = t_{i+1} - t_i$ .

## Exercise: Analytically Solve the Steady-State Levels of Capital and Consumption

Solve for the steady-state levels of capital and consumption in the Neoclassical Growth Model, given the following system of equations:

$$\frac{\dot{c}(t)}{c(t)} = \frac{f'(k(t)) - \rho - \theta g}{\theta} \quad (1)$$

$$\dot{k}(t) = f(k(t)) - c(t) - (n + g)k(t) \quad (2)$$

assuming the production function follows a Cobb-Douglas form,  $f(k) = AK^\alpha$ .

# Steady States

## Steady-State Levels of Capital and Consumption

Starting from the Euler equation at steady state:

$$0 = \frac{\dot{c}(t)}{c(t)} = \frac{f'(k(t)) - \rho - \theta g}{\theta} = \frac{\alpha A k^{\alpha-1} - \rho - \theta g}{\theta}$$
$$\therefore k_{ss} = \left( \frac{\rho + \theta g}{\alpha A} \right)^{\frac{1}{\alpha-1}} \quad (5)$$

Now, using the capital accumulation equation at steady state:

$$0 = \dot{k}(t) = f(k(t)) - c(t) - (n + g)k(t)$$
$$c_{ss} = f(k_{ss}) - (n + g)k_{ss} = A k_{ss}^{\alpha} - (n + g)k_{ss}$$
$$\therefore c_{ss} = A k_{ss}^{\alpha} - (n + g)k_{ss} \quad (6)$$

## Determination of $c(0)$

---

- We use Newton's method to find the correct initial consumption  $c(0)$  that ensures the terminal capital  $k(T)$  converges to the steady-state capital  $k_{ss}$ .

## Determination of $c(0)$

---

- We use Newton's method to find the correct initial consumption  $c(0)$  that ensures the terminal capital  $k(T)$  converges to the steady-state capital  $k_{ss}$ .
- Newton's method is an iterative numerical technique used to find the roots of a nonlinear function  $f(x) = 0$ .

## Determination of $c(0)$

---

- We use Newton's method to find the correct initial consumption  $c(0)$  that ensures the terminal capital  $k(T)$  converges to the steady-state capital  $k_{ss}$ .
- Newton's method is an iterative numerical technique used to find the roots of a nonlinear function  $f(x) = 0$ .
- In this context:
  - We define a function  $f(c(0)) = k_T(c(0)) - k_{ss}$ , where  $k_T(c(0))$  is the capital at the terminal time given the initial consumption  $c(0)$ .
  - Newton's method is used to iteratively adjust  $c(0)$  so that  $f(c(0)) = 0$ , meaning  $k(T)$  matches the steady-state capital  $k_{ss}$ .

## Newton's Method

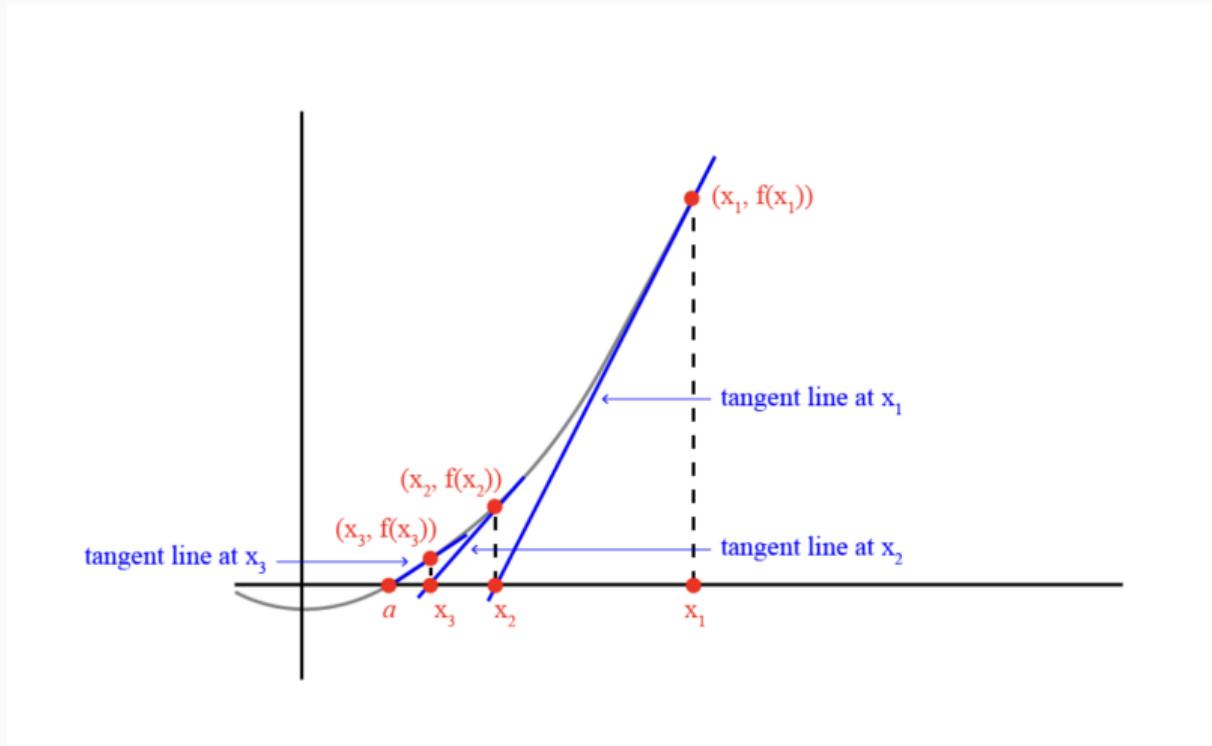
Given an initial guess  $x_0$ , Newton's method iterates according to the following update rule:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

where:

- $x_n$  is the current guess,
- $f(x_n)$  is the function value at  $x_n$ ,
- $f'(x_n)$  is the derivative of the function at  $x_n$ ,
- $x_{n+1}$  is the updated guess.

# Newton's Method



**Figure 6:** Graphical illustration of Newton's method

## Convergence of Newton's Method

- If (i)  $f(x)$  is smooth,  $f \in C^2$  and (ii) the initial guess  $x_0$  is sufficiently close to the true root  $x^*$ , then Newton's method converges quadratically. This means that the error  $|x_n - x^*|$  decreases approximately with the square of the previous error at each iteration, leading to very rapid convergence near the solution.

## Convergence of Newton's Method

- If (i)  $f(x)$  is smooth,  $f \in C^2$  and (ii) the initial guess  $x_0$  is sufficiently close to the true root  $x^*$ , then Newton's method converges quadratically. This means that the error  $|x_n - x^*|$  decreases approximately with the square of the previous error at each iteration, leading to very rapid convergence near the solution.
- If the initial guess is far from the root or if  $f'(x)$  is close to zero, convergence may be slow or fail altogether.

## Convergence of Newton's Method

- If (i)  $f(x)$  is smooth,  $f \in C^2$  and (ii) the initial guess  $x_0$  is sufficiently close to the true root  $x^*$ , then Newton's method converges quadratically. This means that the error  $|x_n - x^*|$  decreases approximately with the square of the previous error at each iteration, leading to very rapid convergence near the solution.
- If the initial guess is far from the root or if  $f'(x)$  is close to zero, convergence may be slow or fail altogether.
- Selecting a well-informed initial guess improves convergence reliability. In economic models, we often use the steady-state value to set an initial guess for  $c(0)$ , as it provides a realistic starting point close to the model's expected long-term equilibrium.

## Exercise: Numerically Solve the Neoclassical Growth Model

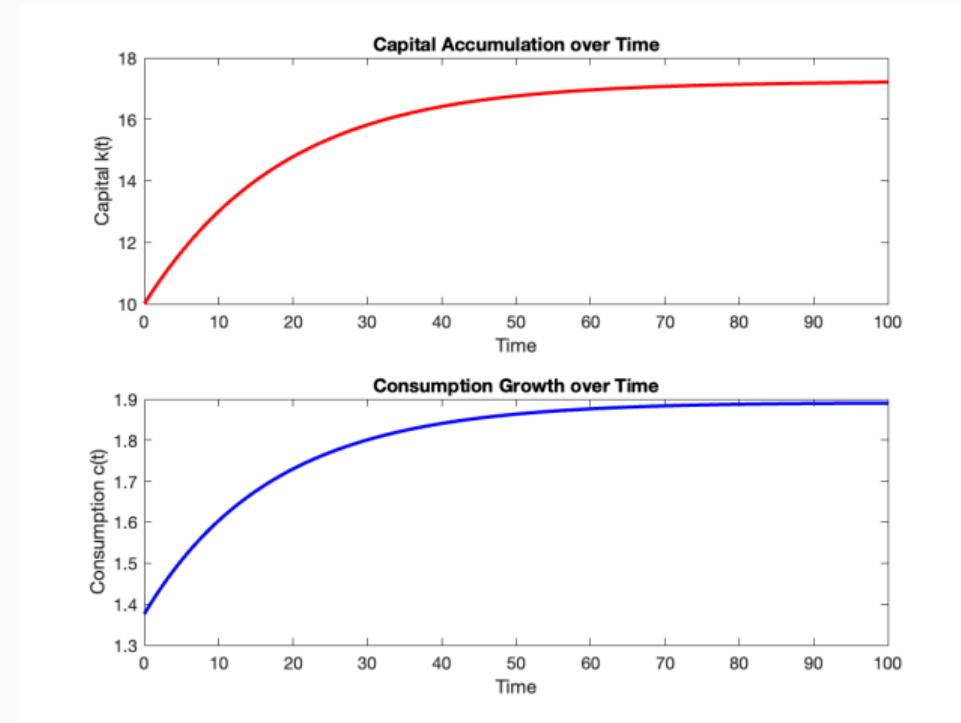
Solve the Neoclassical Growth model with the following system of equations:

$$\frac{\dot{c}(t)}{c(t)} = \frac{f'(k(t)) - \rho - \theta g}{\theta} \quad (7)$$

$$\dot{k}(t) = f(k(t)) - c(t) - (n + g)k(t) \quad (8)$$

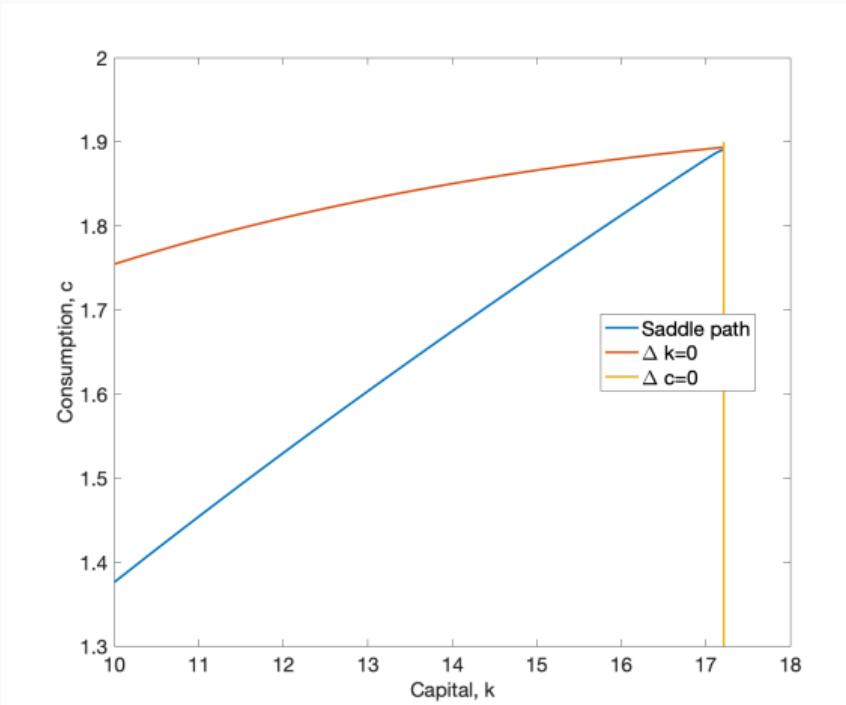
where the initial condition for capital is  $K_0 = 10$ , and the intertemporal budget constraint with equality is imposed as the terminal condition.

# Capital and Consumption Paths Over Time



**Figure 7:** Capital and Consumption Paths Over Time

# Saddle Path Dynamics: Initial Capital Below Steady State



**Figure 8:** Saddle Path Dynamics: Initial Capital Below Steady State

## **Section 2-2:**

# **Neoclassical Growth Model in Recursive Representation**

---

## Neoclassical Growth Model in Discrete-Time Recursive Formulation

We start with the Ramsey growth model in discrete time, assuming  $g = 0$  and  $n = 0$ :

$$\begin{aligned} V(k_t) &= \max_{c_t, k_{t+1}} \{ U(c_t) + (1 - \rho)V(k_{t+1}) \} \\ \text{s.t. } c_t + k_{t+1} &= f(k_t) + (1 - \delta)k_t \\ f(k_t) &= k_t^\alpha \end{aligned} \tag{9}$$

## Discrete to Continuous-Time Transformation

Over a time interval of  $\Delta$  units, the model can be expressed as:

$$V(k_t) = \max_{c_t, k_{t+\Delta}} \{ \Delta U(c_t) + (1 - \Delta\rho)V(k_{t+\Delta}) \}$$

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

$$f(k_t) = k_t^\alpha$$

## Discrete to Continuous-Time Transformation

Subtracting  $V(k_t)$  from both sides and substituting the constraints into  $V(k_{t+\Delta})$ , we get:

$$\begin{aligned} 0 &= \max_{c_t} \left\{ \Delta U(c_t) + (1 - \Delta\rho)V(\Delta k_t^\alpha + (1 - \Delta\delta)k_t - \Delta c_t) - V(k_t) \right\} \\ &= \max_{c_t} \left\{ \Delta U(c_t) + V(k_t + \Delta(k_t^\alpha - \delta k_t - c_t)) - V(k_t) - \Delta\rho V(k_t + \Delta(k_t^\alpha - \delta k_t - c_t)) \right\} \end{aligned}$$

Dividing both sides by  $\Delta$ :

$$0 = \max_{c_t} \left\{ U(c_t) + \frac{V(k_t + \Delta(f(k_t) - \delta k_t - c_t)) - V(k_t)}{\Delta} - \rho V(k_t + \Delta(k_t^\alpha - \delta k_t - c_t)) \right\}$$

Taking the limit as  $\Delta \rightarrow 0$ , we obtain:

$$0 = \max_{c_t} \{ U(c_t) + V'(k_t)(f(k_t) - \delta k_t - c_t) - \rho V(k_t) \}$$

## Hamilton-Jacobi-Bellman (HJB) Equation

Rearranging terms and dropping time notation leads to the HJB equation:

$$\rho V(k) = \max_c \{ U(c) + V'(k) \cdot (f(k) - \delta k - c) \} \quad (10)$$

## Hamilton-Jacobi-Bellman (HJB) Equation

Rearranging terms and dropping time notation leads to the HJB equation:

$$\rho V(k) = \max_c \{ U(c) + V'(k) \cdot (f(k) - \delta k - c) \} \quad (10)$$

The optimal consumption,  $c = c(k)$ , is derived from the first-order condition:

$$U'(c) = V'(k) \quad (11)$$

## Hamilton-Jacobi-Bellman (HJB) Equation

Rearranging terms and dropping time notation leads to the HJB equation:

$$\rho V(k) = \max_c \{ U(c) + V'(k) \cdot (f(k) - \delta k - c) \} \quad (10)$$

The optimal consumption,  $c = c(k)$ , is derived from the first-order condition:

$$U'(c) = V'(k) \quad (11)$$

Denote  $s(k) = f(k) - \delta k - c = k^\alpha - \delta k - c$ , which represents optimal savings (investment).

## **Section 2-3:**

## **Numerical Solution:**

## **Finite Difference Method**

---

## Finite Difference Approximation

The finite difference approximations to HJB equation (10), associated with the FOC (11) is:

$$\begin{aligned}\rho V_i &= U(c_i) + V'_i \cdot (k_i^\alpha - \delta k_i - c_i) \\ \text{with } c_i &= (U')^{-1}(V'_i)\end{aligned}\tag{12}$$

where  $i = 1, \dots, I$ ,  $V_i = V(k_i)$  with a uniform step size  $\Delta k = k_{i+1} - k_i$ .

## Key Challenges

---

1. Approximating the **derivative** of the value function,  $V'_i$ .

1. Approximating the **derivative** of the value function,  $V'_i$ .

- Mixed Method
- Upwind Scheme (Next section)

## Key Challenges

---

1. Approximating the **derivative** of the value function,  $V'_i$ .
  - Mixed Method
  - Upwind Scheme (Next section)
2. Solving the system, which is highly **non-linear**, requires iterative schemes.

# Key Challenges

---

1. Approximating the **derivative** of the value function,  $V'_i$ .
  - Mixed Method
  - Upwind Scheme (Next section)
2. Solving the system, which is highly **non-linear**, requires iterative schemes.
  - Explicit Method
  - Implicit Method

# Key Challenges

1. Approximating the derivative of the value function,  $V'_i$ .
  - Mixed Method
  - **Upwind Scheme** (Next section)
2. Solving the system, which is highly non-linear, requires iterative schemes.
  - Explicit Method
  - **Implicit Method**

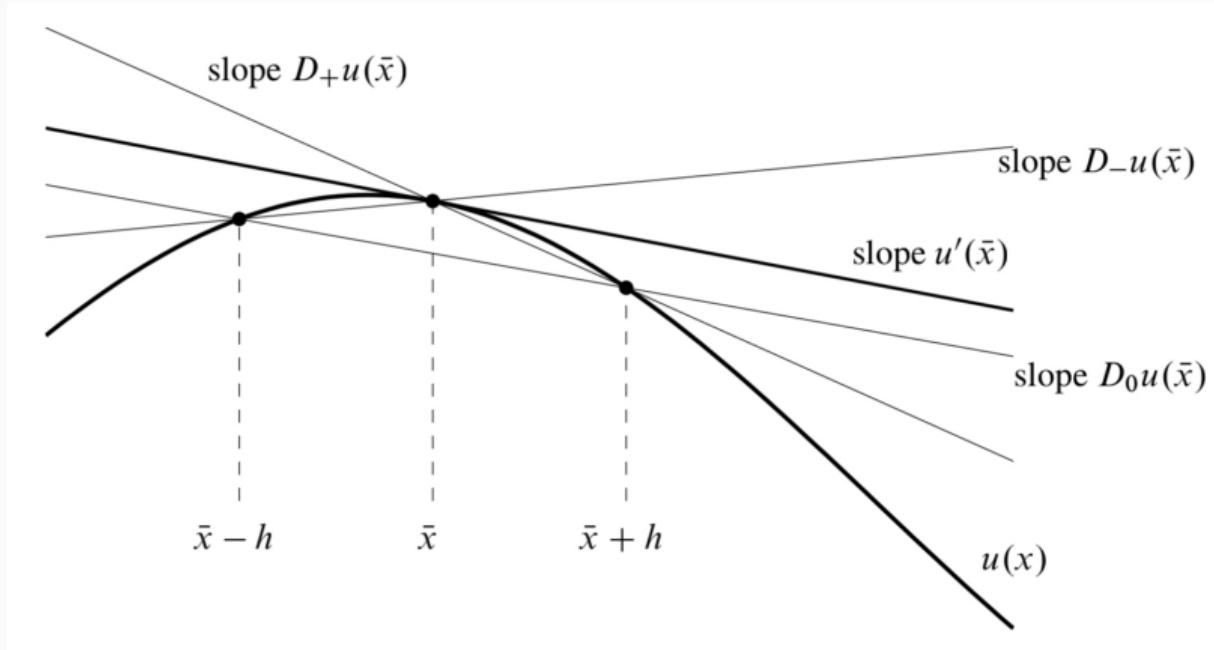
## Algorithm Sketch

1. Construct  $I$  discrete grid points for  $k$ , denoted as  $k_i$  where  $i = 1, \dots, I$ , and let  $V_i = V(k_i)$ .
2. For each  $k_i$  on the grid, guess for a value of  $\mathbf{V}^0 = (V_1^0, V_2^0, \dots, V_I^0)$ .

*For iterations  $n = 0, 1, 2, \dots$ ,*

3. Compute  $(V_i^n)'$  using a mixed method or upwind scheme.
4. Compute  $\mathbf{c}^n$  from  $c_i^n = (U')^{-1}[(V_i^n)']$ .
5. Find  $V_i^{n+1}$  using the update rule of an explicit or implicit method.
6. If  $\mathbf{V}^{n+1}$  is close enough to  $\mathbf{V}^n$ : stop. Otherwise, go to step 3.

# Finite Difference Approximation



**Figure 9:** Various approximations to  $u'(\bar{x})$  interpreted as the slope of secant lines.

# Finite Difference Approximations

Suppose we have values  $V_i = V(k_i)$  on a uniformly spaced grid of  $k$ , denoted as  $K = \{k_1, \dots, k_I\}$ , with step size  $\Delta k = k_{i+1} - k_i$ .

- The forward difference approximation of  $V'$  at  $k_i$  is:

$$V'_i \approx \frac{V_{i+1} - V_i}{\Delta k} \equiv V'_{i,F}$$

- The backward difference approximation of  $V'$  at  $k_i$  is:

$$V'_i \approx \frac{V_i - V_{i-1}}{\Delta k} \equiv V'_{i,B}$$

- The central difference approximation of  $V'$  at  $k_i$  is:

$$V'_i \approx \frac{V_{i+1} - V_{i-1}}{2\Delta k} \equiv V'_{i,C}$$

## Mixed Method

The mixed method approximation for  $V'_i$  is defined as:

$$V'_i \simeq \begin{cases} V'_{i,F} = \frac{V_{i+1} - V_i}{\Delta k}, & i = 1 \\ V'_{i,C} = \frac{V_{i+1} - V_{i-1}}{2\Delta k}, & i \in \{2, 3, \dots, I-1\} \\ V'_{i,B} = \frac{V_i - V_{i-1}}{\Delta k}, & i = I \end{cases} \quad (13)$$

# Implementation in MATLAB

- Use MATLAB's `gradient` function to compute the numerical derivative:

$$\mathbf{V}'(k) = \text{gradient}(\mathbf{V})/dk$$

## Algorithms

`gradient` calculates the *central difference* for interior data points. For example, consider a matrix with unit-spaced data,  $A$ , that has horizontal gradient  $G = \text{gradient}(A)$ . The interior gradient values,  $G(:, j)$ , are

$$G(:, j) = 0.5 * (A(:, j+1) - A(:, j-1));$$

The subscript  $j$  varies between 2 and  $N-1$ , with  $N = \text{size}(A, 2)$ .

`gradient` calculates values along the edges of the matrix with *single-sided differences*:

$$\begin{aligned} G(:, 1) &= A(:, 2) - A(:, 1); \\ G(:, N) &= A(:, N) - A(:, N-1); \end{aligned}$$

## Implementation in MATLAB

- Use MATLAB's gradient function to compute the numerical derivative:

$$\mathbf{V}'(k) = \text{gradient}(\mathbf{V})/dk$$

- Construct the  $I \times I$  matrix  $\mathbf{D}$  (finite-difference operator), so that the derivative  $\mathbf{V}'(k)$  can be approximated by:

$$\mathbf{V}'(k) \simeq \mathbf{D} \times \mathbf{V}(k)$$

The matrix  $\mathbf{D}$  is defined as:

$$\mathbf{D} = \begin{pmatrix} -1/dk & 1/dk & 0 & \cdots & 0 \\ -0.5/dk & 0 & 0.5/dk & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \vdots & -0.5/dk & 0 & 0.5/dk \\ 0 & 0 & 0 & -1/dk & 1/dk \end{pmatrix} \quad (14)$$

## Explicit Method

---

The value function is updated iteratively for  $n = 1, \dots$ , according to:

$$\rho V_i^{n+1} = U(c_i^n) + (V_i^n)' \cdot (k_i^\alpha - \delta k_i - c_i^n)$$

with  $c_i^n = (U')^{-1}[(V_i^n)']$

## Explicit Method

The value function is updated iteratively for  $n = 1, \dots$ , according to:

$$\rho V_i^{n+1} = U(c_i^n) + (V_i^n)' \cdot (k_i^\alpha - \delta k_i - c_i^n)$$

with  $c_i^n = (U')^{-1}[(V_i^n)']$

While the explicit method is straightforward to implement, it can be less reliable in numerical implementations.

## Explicit Method

To improve convergence, the value function is updated iteratively for  $n = 1, \dots$ , according to:

$$\frac{V_i^{n+1} - V_i^n}{\Delta} + \rho V_i^n = U(c_i^n) + (V_i^n)' \cdot (k_i^\alpha - \delta k_i - c_i^n) \quad (15)$$

with  $c_i^n = (U')^{-1}[(V_i^n)']$

## Explicit Method

To improve convergence, the value function is updated iteratively for  $n = 1, \dots$ , according to:

$$\frac{V_i^{n+1} - V_i^n}{\Delta} + \rho V_i^n = U(c_i^n) + (V_i^n)' \cdot (k_i^\alpha - \delta k_i - c_i^n) \quad (15)$$

with  $c_i^n = (U')^{-1}[(V_i^n)']$

The parameter  $\Delta$  is the step size of the explicit method. It can be shown that the explicit method only converges if  $\Delta$  is sufficiently (prohibitively) small. (Candler, 1999)

## Explicit Method Algorithm

---

1. Construct  $I$  discrete grid points for  $k$ , denoted as  $k_i$  where  $i = 1, \dots, I$ , and let  $V_i = V(k_i)$ .
2. For each  $k_i$  on the grid, guess for a value of  $\mathbf{V}^0 = (V_1^0, V_2^0, \dots, V_I^0)$ .

*For iterations  $n = 0, 1, 2, \dots$ ,*

3. Compute  $(V_i^n)'$  using (13).
4. Compute  $\mathbf{c}^n$  from  $c_i^n = (U')^{-1}[(V_i^n)']$ .
5. Find  $V_i^{n+1}$  from (15).
6. If  $\mathbf{V}^{n+1}$  is close enough to  $\mathbf{V}^n$ : stop. Otherwise, go to step 3.

## Implicit Method

$V^{n+1}$  is now *implicitly* defined by:

$$\frac{V_i^{n+1} - V_i^n}{\Delta} + \rho V_i^{n+1} = U(c_i^n) + (V_i^{n+1})' \cdot (k_i^\alpha - \delta k_i - c_i^n) \quad (16)$$

with  $c_i^n = (U')^{-1}[(V_i^n)']$

## Implicit Method

$V^{n+1}$  is now *implicitly* defined by:

$$\frac{V_i^{n+1} - V_i^n}{\Delta} + \rho V_i^{n+1} = U(c_i^n) + (V_i^{n+1})' \cdot (k_i^\alpha - \delta k_i - c_i^n) \quad (17)$$

with  $c_i^n = (U')^{-1}[(V_i^n)']$

The step size  $\Delta$  can be arbitrarily large. (Achdou et al., 2022)

## Implicit Method Algorithm

1. Construct  $I$  discrete grid points for  $k$ , denoted as  $k_i$  where  $i = 1, \dots, I$ , and let  $V_i = V(k_i)$ .
2. For each  $k_i$  on the grid, guess for a value of  $\mathbf{V}^0 = (V_1^0, V_2^0, \dots, V_I^0)$ .

*For iterations  $n = 0, 1, 2, \dots$ ,*

3. Compute  $(V_i^n)'$  using (13).
4. Compute  $\mathbf{c}^n$  from  $c_i^n = (U')^{-1}[(V_i^n)']$ .
5. Find  $V_i^{n+1}$  from (17).
6. If  $\mathbf{V}^{n+1}$  is close enough to  $\mathbf{V}^n$ : stop. Otherwise, go to step 3.

## Implicit Method: Matrix Representation

1. Define  $I$  discrete grid points for  $k$ , denoted as  $k_i$  for  $i = 1, \dots, I$ , and form an  $I \times 1$  vector  $\mathbf{k} = [k_1, k_2, \dots, k_I]'$ .
2. Let  $V_i = V(k_i)$ . For each  $k_i$  on the grid, make an initial guess for the value function as an  $I \times 1$  vector  $\mathbf{V}^0 = [V_1^0, V_2^0, \dots, V_I^0]'$ .  
*For iterations  $n = 0, 1, 2, \dots$*
3. Compute the derivative of the value function as an  $I \times 1$  vector  $(\mathbf{V}^n)'$  using an  $I \times I$  finite-difference operator  $\mathbf{D}$  such that  $\mathbf{DV}^n \simeq (\mathbf{V}^n)'$ .
4. Compute the optimal consumption as an  $I \times 1$  vector  $\mathbf{c}^n$  from  $\mathbf{c}^n = (U')^{-1}(\mathbf{DV}^n)$ .
5. Compute the optimal savings as an  $I \times 1$  vector  $\mathbf{s}^n$  from  $\mathbf{s}^n = f(\mathbf{k}) - \delta\mathbf{k} - \mathbf{c}^n$ .
6. Find  $\mathbf{V}^{n+1}$  from:  
$$\frac{1}{\Delta}(\mathbf{V}^{n+1} - \mathbf{V}^n) + \rho\mathbf{V}^{n+1} = U(\mathbf{c}^n) + (\mathbf{DV}^{n+1}) \cdot \mathbf{s}^n$$
where the dot indicates element-wise multiplication.
7. If  $\mathbf{V}^{n+1}$  is close enough to  $\mathbf{V}^n$ : stop. Otherwise, go to step 3.

# Matrix Representation

Alternative matrix formulation:

$$\frac{1}{\Delta}(\mathbf{V}^{n+1} - \mathbf{V}^n) + \rho \mathbf{V}^{n+1} = U(\mathbf{c}^n) + \mathbf{S}^n \mathbf{D} \mathbf{V}^{n+1}$$

where  $\mathbf{S}^n = \text{diag}(\mathbf{s}^n)$  is an  $I \times I$  diagonal matrix with diagonals  $\mathbf{s}^n = \{s_1^n, \dots, s_I^n\}$ .

Equivalently, solve the linear system:

$$\mathbf{V}^{n+1} = \left( (\rho + \frac{1}{\Delta}) \mathbf{I} - \mathbf{S}^n \mathbf{D} \right)^{-1} \left[ U(\mathbf{c}^n) + \frac{1}{\Delta} \mathbf{V}^n \right] \quad (18)$$

## Steady-State Conditions and Capital Grid Setup

---

To capture dynamics near the steady state accurately, set the grid for the state variable  $k$  within a range that includes the steady-state level  $k_{ss}$ .

# Steady-State Conditions and Capital Grid Setup

To capture dynamics near the steady state accurately, set the grid for the state variable  $k$  within a range that includes the steady-state level  $k_{ss}$ .

## Exercise: Solve for the Steady-State Level of Capital

Solve for the steady-state level of capital in the following equation:

$$\rho V(k) = \max_c \{ U(c) + V'(k) \cdot (f(k) - \delta k - c) \}$$

with  $U'(c) = V'(k)$

# Steady-State Conditions and Capital Grid Setup

## Steady-State Level of Capital

In the steady state:

$$\dot{k} = f(k) - \delta k - c = 0$$

Therefore, the following conditions hold:

$$\rho V'(k) = U'(c) \cdot (f'(k) - \delta)$$

Applying the FOC,  $V'(k) = U'(c)$ :

$$f'(k_{ss}) = \rho + \delta$$

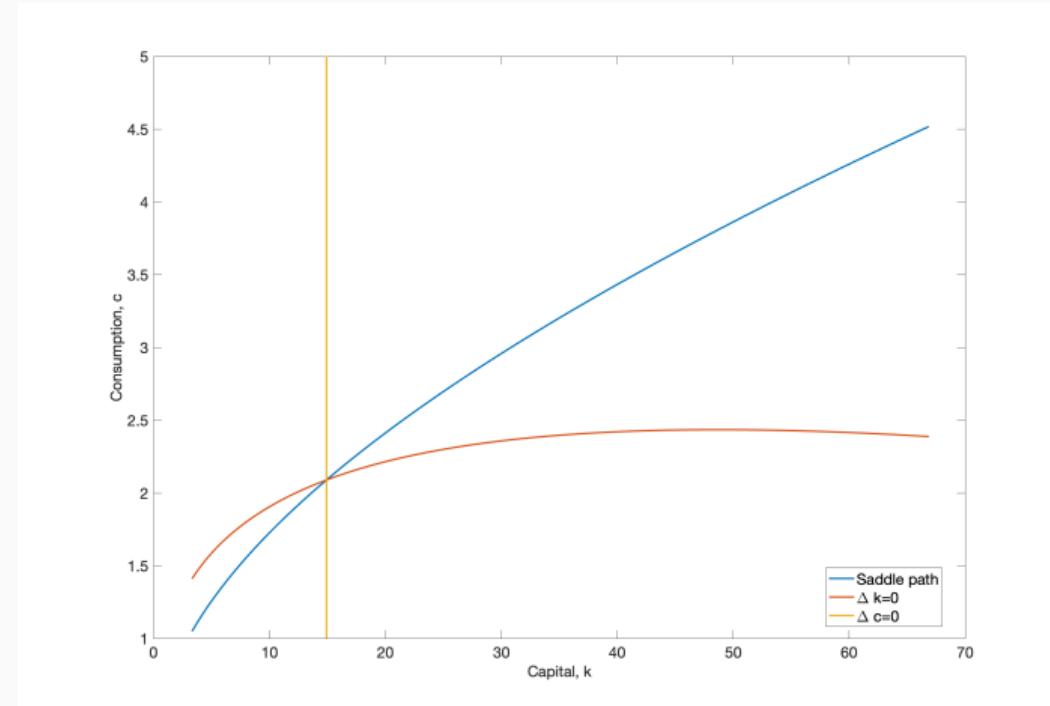
$$\therefore k_{ss} = \left( \frac{\rho + \delta}{\alpha A} \right)^{\frac{1}{\alpha - 1}}$$

## Initial Guess for the Value Function

We use the following initial guess for the value function:

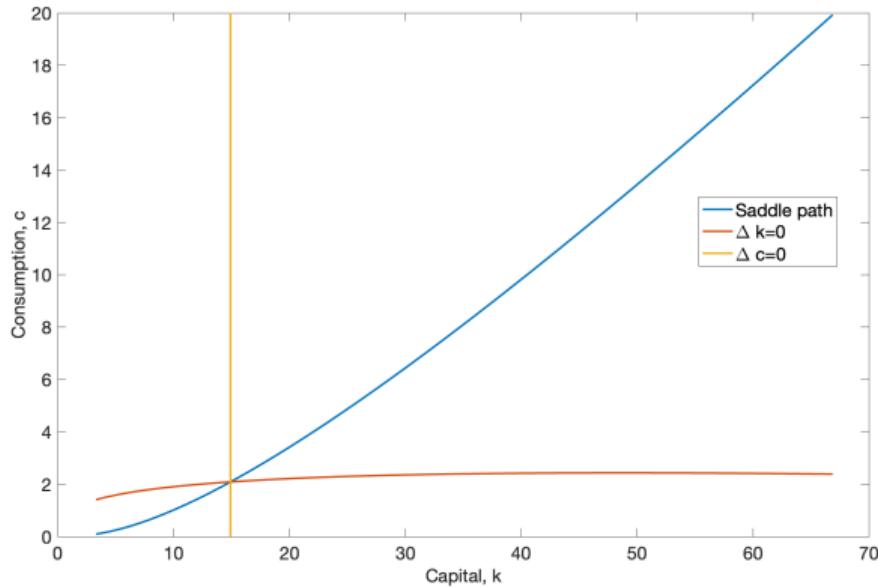
$$V_i^0 = \frac{U(f(k_i))}{\rho}, \quad i = 1, \dots, I.$$

# Saddle Path Dynamics



**Figure 10:** Saddle Path Dynamics

# Saddle Path Dynamics with Low $\theta$



**Figure 11:** Saddle Path Dynamics with Low  $\theta$

## Saddle Path with Different $\theta$

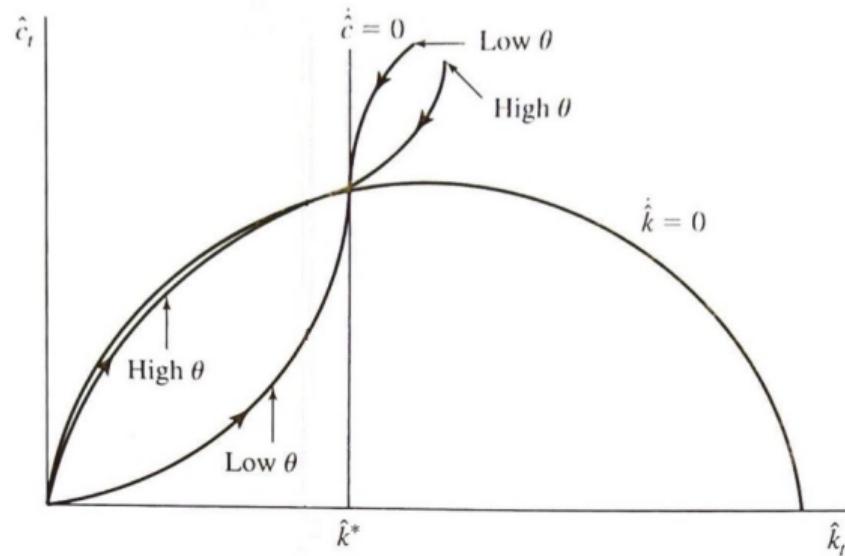


Figure 2.2

Source: Barro and Sala-i-Martin (2004)

Figure 12: Saddle Path with Different  $\theta$

## Shape of the Saddle Path

- The saddle path gives  $c(k)$  (called the policy function)
- What is the shape of this path?
- Consider different values of  $\theta$
- Recall that  $1/\theta$  is the intertemporal elasticity of substitution
- High  $\theta$  (low  $1/\theta$ ) implies strong desire to smooth consumption  
Household will try to shift consumption from the future  
Saddle path will be close to  $\dot{k}(t) = 0$  locus

**Figure 13:** Shape of the Saddle Path (Steinsson 2024)

## References

---

Achdou, Y., J. Han, J.-M. Lasry, P.-L. Lions, and B. Moll (2022). Income and wealth distribution in macroeconomics: A continuous-time approach. The review of economic studies 89(1), 45–86.

Candler, G. V. (1999). Finite-difference methods for dynamic programming problems. Computational Methods for the Study of Dynamic Economies.

Romer, D. (2022). Advanced Macroeconomics (6th ed.). New York: McGraw-Hill Education.