

# Lecture 8: Value Function Iteration

Zhuotong Liu

University of Edinburgh

Autumn 2025

# Why Do We Need Numerical Models?

- In physics or chemistry, we can run controlled experiments  
*If we want to know which of two balls hits the ground first, we can simply drop them*
- But economics deals with people, firms, and governments we cannot experiment on them in the same way  
*If we want to know how building railways affects a city's development, we cannot add new rail lines to Edinburgh and tear them down in Glasgow just to see what happens*

# Why Do We Need Numerical Models?

- In physics or chemistry, we can run controlled experiments  
*If we want to know which of two balls hits the ground first, we can simply drop them*
- But economics deals with people, firms, and governments we cannot experiment on them in the same way  
*If we want to know how building railways affects a city's development, we cannot add new rail lines to Edinburgh and tear them down in Glasgow just to see what happens*

- So economists take a different approach:

- **Setup:** Use *mathematical functions* to describe relationships between economic variables
- **Solving:** Solve for the *optimal policy rules* that determine how agents behave under different situations

Don't worry if following steps don't make much sense right now. I'll be much more precise about this next week

- **Simulation:** Generate *simulated (fake) data* from the model by randomly drawing shocks and updating agents according to their optimal policy rules
  - **Estimation:** Choose parameters that make the model's simulated data match the observed real-world data
  - **Policy Experiments:** Change policy variables, such as interest rates, taxes, or money supply, to study how the economy would respond
- 
- In other words, numerical models are our laboratories for economics
  - Dynamic programming is the core technique in numerical models

- So economists take a different approach:
  - **Setup:** Use *mathematical functions* to describe relationships between economic variables
  - **Solving:** Solve for the *optimal policy rules* that determine how agents behave under different situations

Don't worry if following steps don't make much sense right now. I'll be much more precise about this next week

- **Simulation:** Generate *simulated (fake) data* from the model by randomly drawing shocks and updating agents according to their optimal policy rules
  - **Estimation:** Choose parameters that make the model's simulated data match the observed real-world data
  - **Policy Experiments:** Change policy variables, such as interest rates, taxes, or money supply, to study how the economy would respond
- 
- In other words, numerical models are our laboratories for economics
  - Dynamic programming is the core technique in numerical models

- So economists take a different approach:

- **Setup:** Use *mathematical functions* to describe relationships between economic variables
- **Solving:** Solve for the *optimal policy rules* that determine how agents behave under different situations

Don't worry if following steps don't make much sense right now. I'll be much more precise about this next week

- **Simulation:** Generate *simulated (fake) data* from the model by randomly drawing shocks and updating agents according to their optimal policy rules
  - **Estimation:** Choose parameters that make the model's simulated data match the observed real-world data
  - **Policy Experiments:** Change policy variables, such as interest rates, taxes, or money supply, to study how the economy would respond
- 
- In other words, numerical models are our laboratories for economics
  - Dynamic programming is the core technique in numerical models

- So economists take a different approach:

- **Setup:** Use *mathematical functions* to describe relationships between economic variables
- **Solving:** Solve for the *optimal policy rules* that determine how agents behave under different situations

Don't worry if following steps don't make much sense right now. I'll be much more precise about this next week

- **Simulation:** Generate *simulated (fake) data* from the model by randomly drawing shocks and updating agents according to their optimal policy rules
  - **Estimation:** Choose parameters that make the model's simulated data match the observed real-world data
  - **Policy Experiments:** Change policy variables, such as interest rates, taxes, or money supply, to study how the economy would respond
- 
- In other words, numerical models are our laboratories for economics
  - Dynamic programming is the core technique in numerical models

- So economists take a different approach:

- **Setup:** Use *mathematical functions* to describe relationships between economic variables
- **Solving:** Solve for the *optimal policy rules* that determine how agents behave under different situations

Don't worry if following steps don't make much sense right now. I'll be much more precise about this next week

- **Simulation:** Generate *simulated (fake) data* from the model by randomly drawing shocks and updating agents according to their optimal policy rules
  - **Estimation:** Choose parameters that make the model's simulated data match the observed real-world data
  - **Policy Experiments:** Change policy variables, such as interest rates, taxes, or money supply, to study how the economy would respond
- 
- In other words, numerical models are our laboratories for economics
  - Dynamic programming is the core technique in numerical models

- So economists take a different approach:
  - **Setup:** Use *mathematical functions* to describe relationships between economic variables
  - **Solving:** Solve for the *optimal policy rules* that determine how agents behave under different situations

Don't worry if following steps don't make much sense right now. I'll be much more precise about this next week

- **Simulation:** Generate *simulated (fake) data* from the model by randomly drawing shocks and updating agents according to their optimal policy rules
  - **Estimation:** Choose parameters that make the model's simulated data match the observed real-world data
  - **Policy Experiments:** Change policy variables, such as interest rates, taxes, or money supply, to study how the economy would respond
- In other words, numerical models are our laboratories for economics
- Dynamic programming is the core technique in numerical models

# Dynamic Programs are Everywhere

- Dynamic problems show up everywhere in Economics
- You find an agent is choosing how to trade off a reward today against waiting for tomorrow. E.g.
  - Saving for tomorrow vs. consuming today
  - How young people make their career decisions (Keane & Wolpin, 1997)
  - Should a bus mechanic repair the engine today, or wait until next month (Rust, 1987)
- The problem is that they are extremely difficult to solve
- In general, closed-form solutions don't exist, and we have to solve them on a computer

## **Section 1**

# Finite Horizon Dynamic Programs

## The neoclassical growth model in 3 periods

- Suppose we take the standard neoclassical growth model with only three periods.
  - Households choose between consuming and investing in the capital stock
  - Capital depreciation at rate  $\delta$ , and initial capital  $k_1$
  - Flow utility  $u(c)$  and production function  $F_t(k) = A_t k^\alpha$
- We can write this problem with a period by period budget constraint:

$$\begin{aligned} & := \max_{c_1, c_2, c_3, k_2, k_3} u(c_1) + \beta u(c_2) + \beta^2 u(c_3) \\ \text{s.t. } & c_1 + k_2 \leq A_1 k_1^\alpha + (1 - \delta)k_1 \quad \text{Period 1 BC} \quad (1) \\ & c_2 + k_3 \leq A_2 k_2^\alpha + (1 - \delta)k_2 \quad \text{Period 2 BC} \\ & c_3 \leq A_3 k_3^\alpha + (1 - \delta)k_3 \quad \text{Period 3 BC} \end{aligned}$$

- If given  $u(c)$ ,  $\delta$ ,  $\beta$  and  $\{A_t\}$ , you know how to put this on a computer and solve it:
  - Sequentially substitute out budget constraints and maximize over the variables  $k_1$  and  $k_2$
  - You did something like this in an earlier problem set
- Call the maximized value  $v_1(k_1)$

## The neoclassical growth model in 3 periods

- Suppose we take the standard neoclassical growth model with only three periods.
  - Households choose between consuming and investing in the capital stock
  - Capital depreciation at rate  $\delta$ , and initial capital  $k_1$
  - Flow utility  $u(c)$  and production function  $F_t(k) = A_t k^\alpha$
- We can write this problem with a period by period budget constraint:

$$\begin{aligned} & := \max_{c_1, c_2, c_3, k_2, k_3} u(c_1) + \beta u(c_2) + \beta^2 u(c_3) \\ \text{s.t. } & c_1 + k_2 \leq A_1 k_1^\alpha + (1 - \delta)k_1 \quad \text{Period 1 BC} \quad (1) \\ & c_2 + k_3 \leq A_2 k_2^\alpha + (1 - \delta)k_2 \quad \text{Period 2 BC} \\ & c_3 \leq A_3 k_3^\alpha + (1 - \delta)k_3 \quad \text{Period 3 BC} \end{aligned}$$

- If given  $u(c)$ ,  $\delta$ ,  $\beta$  and  $\{A_t\}$ , you know how to put this on a computer and solve it:
  - Sequentially substitute out budget constraints and maximize over the variables  $k_1$  and  $k_2$
  - You did something like this in an earlier problem set
- Call the maximized value  $v_1(k_1)$

## The neoclassical growth model in 3 periods

- Suppose we take the standard neoclassical growth model with only three periods.
  - Households choose between consuming and investing in the capital stock
  - Capital depreciation at rate  $\delta$ , and initial capital  $k_1$
  - Flow utility  $u(c)$  and production function  $F_t(k) = A_t k^\alpha$
- We can write this problem with a period by period budget constraint:

$$v_1(k_1) = \max_{c_1, c_2, c_3, k_2, k_3} u(c_1) + \beta u(c_2) + \beta^2 u(c_3)$$

$$\text{s.t. } c_1 + k_2 \leq A_1 k_1^\alpha + (1 - \delta)k_1 \quad \text{Period 1 BC} \quad (1)$$

$$c_2 + k_3 \leq A_2 k_2^\alpha + (1 - \delta)k_2 \quad \text{Period 2 BC}$$

$$c_3 \leq A_3 k_3^\alpha + (1 - \delta)k_3 \quad \text{Period 3 BC}$$

- If given  $u(c)$ ,  $\delta$ ,  $\beta$  and  $\{A_t\}$ , you know how to put this on a computer and solve it:
  - Sequentially substitute out budget constraints and maximize over the variables  $k_1$  and  $k_2$
  - You did something like this in an earlier problem set
- Call the maximized value  $v_1(k_1)$

## Multi-stage budgeting

- This will work for  $T$  periods when  $T$  is small, but it doesn't generalise well...
- Instead, let's split the problem up into multiple stages, each of which looks easier to solve
- Define  $v_3(k_3)$  as the value you get from starting period 3 with a capital stock  $k_3$ :

$$\begin{aligned} v_3(k_3) &= \max_{c_3} u(c_3) \\ \text{s.t. } c_3 &\leq A_3 k_3^\alpha + (1 - \delta)k_3 \end{aligned} \quad (2)$$

- And define  $v_2(k_2)$  as the value you get from starting period 2 with a capital stock  $k_2$ :

$$\begin{aligned} v_2(k_2) &= \max_{c_2, k_3} u(c_2) + \beta v_3(k_3) \\ \text{s.t. } c_2 + k_3 &\leq A_2 k_2^\alpha + (1 - \delta)k_2 \end{aligned} \quad (3)$$

- Notice that we now have a **continuation value**
- Finally, we can rewrite the first period problem as

$$\begin{aligned} v_1(k_1) &= \max_{c_1, k_2} u(c_1) + \beta v_2(k_2) \\ \text{s.t. } c_1 + k_2 &\leq A_1 k_1^\alpha + (1 - \delta)k_1 \end{aligned} \quad (4)$$

## Multi-stage budgeting

- This will work for  $T$  periods when  $T$  is small, but it doesn't generalise well...
- Instead, let's split the problem up into multiple stages, each of which looks easier to solve
- Define  $v_3(k_3)$  as the value you get from starting period 3 with a capital stock  $k_3$ :

$$\begin{aligned} v_3(k_3) &= \max_{c_3} u(c_3) \\ \text{s.t. } c_3 &\leq A_3 k_3^\alpha + (1 - \delta)k_3 \end{aligned} \quad (2)$$

- And define  $v_2(k_2)$  as the value you get from starting period 2 with a capital stock  $k_2$ :

$$\begin{aligned} v_2(k_2) &= \max_{c_2, k_3} u(c_2) + \beta v_3(k_3) \\ \text{s.t. } c_2 + k_3 &\leq A_2 k_2^\alpha + (1 - \delta)k_2 \end{aligned} \quad (3)$$

- Notice that we now have a **continuation value**
- Finally, we can rewrite the first period problem as

$$\begin{aligned} v_1(k_1) &= \max_{c_1, k_2} u(c_1) + \beta v_2(k_2) \\ \text{s.t. } c_1 + k_2 &\leq A_1 k_1^\alpha + (1 - \delta)k_1 \end{aligned} \quad (4)$$

## Multi-stage budgeting

- This will work for  $T$  periods when  $T$  is small, but it doesn't generalise well...
- Instead, let's split the problem up into multiple stages, each of which looks easier to solve
- Define  $v_3(k_3)$  as the value you get from starting period 3 with a capital stock  $k_3$ :

$$\begin{aligned} v_3(k_3) &= \max_{c_3} u(c_3) \\ \text{s.t. } c_3 &\leq A_3 k_3^\alpha + (1 - \delta)k_3 \end{aligned} \quad (2)$$

- And define  $v_2(k_2)$  as the value you get from starting period 2 with a capital stock  $k_2$ :

$$\begin{aligned} v_2(k_2) &= \max_{c_2, k_3} u(c_2) + \beta v_3(k_3) \\ \text{s.t. } c_2 + k_3 &\leq A_2 k_2^\alpha + (1 - \delta)k_2 \end{aligned} \quad (3)$$

- Notice that we now have a **continuation value**
- Finally, we can rewrite the first period problem as

$$\begin{aligned} v_1(k_1) &= \max_{c_1, k_2} u(c_1) + \beta v_2(k_2) \\ \text{s.t. } c_1 + k_2 &\leq A_1 k_1^\alpha + (1 - \delta)k_1 \end{aligned} \quad (4)$$

## Multi-stage budgeting

- This will work for  $T$  periods when  $T$  is small, but it doesn't generalise well...
- Instead, let's split the problem up into multiple stages, each of which looks easier to solve
- Define  $v_3(k_3)$  as the value you get from starting period 3 with a capital stock  $k_3$ :

$$\begin{aligned} v_3(k_3) &= \max_{c_3} u(c_3) \\ \text{s.t. } c_3 &\leq A_3 k_3^\alpha + (1 - \delta)k_3 \end{aligned} \quad (2)$$

- And define  $v_2(k_2)$  as the value you get from starting period 2 with a capital stock  $k_2$ :

$$\begin{aligned} v_2(k_2) &= \max_{c_2, k_3} u(c_2) + \beta v_3(k_3) \\ \text{s.t. } c_2 + k_3 &\leq A_2 k_2^\alpha + (1 - \delta)k_2 \end{aligned} \quad (3)$$

- Notice that we now have a **continuation value**
- Finally, we can rewrite the first period problem as

$$\begin{aligned} v_1(k_1) &= \max_{c_1, k_2} u(c_1) + \beta v_2(k_2) \\ \text{s.t. } c_1 + k_2 &\leq A_1 k_1^\alpha + (1 - \delta)k_1 \end{aligned} \quad (4)$$

## Multi-stage budgeting

- This will work for  $T$  periods when  $T$  is small, but it doesn't generalise well...
- Instead, let's split the problem up into multiple stages, each of which looks easier to solve
- Define  $v_3(k_3)$  as the value you get from starting period 3 with a capital stock  $k_3$ :

$$\begin{aligned} v_3(k_3) &= \max_{c_3} u(c_3) \\ \text{s.t. } c_3 &\leq A_3 k_3^\alpha + (1 - \delta)k_3 \end{aligned} \quad (2)$$

- And define  $v_2(k_2)$  as the value you get from starting period 2 with a capital stock  $k_2$ :

$$\begin{aligned} v_2(k_2) &= \max_{c_2, k_3} u(c_2) + \beta v_3(k_3) \\ \text{s.t. } c_2 + k_3 &\leq A_2 k_2^\alpha + (1 - \delta)k_2 \end{aligned} \quad (3)$$

- Notice that we now have a **continuation value**
- Finally, we can rewrite the first period problem as

$$\begin{aligned} v_1(k_1) &= \max_{c_1, k_2} u(c_1) + \beta v_2(k_2) \\ \text{s.t. } c_1 + k_2 &\leq A_1 k_1^\alpha + (1 - \delta)k_1 \end{aligned} \quad (4)$$

## Why does this work?

- For simplicity, set  $\delta = 1$  (full depreciation) and look at our problem again:

$$\begin{aligned}v_1(k_1) &:= \max_{c_1, c_2, c_3, k_2, k_3} u(c_1) + \beta u(c_2) + \beta^2 u(c_3) \\ &\quad c_1 + k_2 \leq A_1 k_1^\alpha \quad \text{Period 1 BC} \\ \text{s.t.} \quad &\quad c_2 + k_3 \leq A_2 k_2^\alpha \quad \text{Period 2 BC} \\ &\quad c_3 \leq A_3 k_3^\alpha \quad \text{Period 3 BC}\end{aligned} \tag{5}$$

- If we substitute in our budget constraints, we see that the payoff at period  $t$  only depends on the capital stock you take into the period, and choices you make later:

$$\begin{aligned}v_1(k_1) &= \max_{k_2, k_3} u(A_1 k_1^\alpha - k_2) + \beta u(A_2 k_2^\alpha - k_3) + \beta^2 u(A_3 k_3^\alpha) \\ &= \max_{k_2} u(A_1 k_1^\alpha - k_2) + \beta \underbrace{\left[ \max_{k_3} u(A_2 k_2^\alpha - k_3) + \beta u(A_3 k_3^\alpha) \right]}_{v_2(k_2)}\end{aligned} \tag{6}$$

- Our problem has a **recursive structure**

## Why does this work?

- For simplicity, set  $\delta = 1$  (full depreciation) and look at our problem again:

$$\begin{aligned}v_1(k_1) &:= \max_{c_1, c_2, c_3, k_2, k_3} u(c_1) + \beta u(c_2) + \beta^2 u(c_3) \\ &\quad c_1 + k_2 \leq A_1 k_1^\alpha \quad \text{Period 1 BC} \\ \text{s.t.} \quad &\quad c_2 + k_3 \leq A_2 k_2^\alpha \quad \text{Period 2 BC} \\ &\quad c_3 \leq A_3 k_3^\alpha \quad \text{Period 3 BC}\end{aligned} \tag{5}$$

- If we substitute in our budget constraints, we see that the payoff at period  $t$  only depends on the capital stock you take into the period, and choices you make later:

$$\begin{aligned}v_1(k_1) &= \max_{k_2, k_3} u(A_1 k_1^\alpha - k_2) + \beta u(A_2 k_2^\alpha - k_3) + \beta^2 u(A_3 k_3^\alpha) \\ &= \max_{k_2} u(A_1 k_1^\alpha - k_2) + \beta \underbrace{\left[ \max_{k_3} u(A_2 k_2^\alpha - k_3) + \beta u(A_3 k_3^\alpha) \right]}_{v_2(k_2)}\end{aligned} \tag{6}$$

- Our problem has a **recursive structure**

## Why does this work?

- For simplicity, set  $\delta = 1$  (full depreciation) and look at our problem again:

$$\begin{aligned} v_1(k_1) &:= \max_{c_1, c_2, c_3, k_2, k_3} u(c_1) + \beta u(c_2) + \beta^2 u(c_3) \\ &\quad c_1 + k_2 \leq A_1 k_1^\alpha \quad \text{Period 1 BC} \\ \text{s.t.} \quad &\quad c_2 + k_3 \leq A_2 k_2^\alpha \quad \text{Period 2 BC} \\ &\quad c_3 \leq A_3 k_3^\alpha \quad \text{Period 3 BC} \end{aligned} \tag{5}$$

- If we substitute in our budget constraints, we see that the payoff at period  $t$  only depends on the capital stock you take into the period, and choices you make later:

$$\begin{aligned} v_1(k_1) &= \max_{k_2, k_3} u(A_1 k_1^\alpha - k_2) + \beta u(A_2 k_2^\alpha - k_3) + \beta^2 u(A_3 k_3^\alpha) \\ &= \max_{k_2} u(A_1 k_1^\alpha - k_2) + \beta \underbrace{\left[ \max_{k_3} u(A_2 k_2^\alpha - k_3) + \beta u(A_3 k_3^\alpha) \right]}_{v_2(k_2)} \end{aligned} \tag{6}$$

- Our problem has a **recursive structure**

## Finite Horizon: General Case

- You can follow this logic through to the general case where we have  $T$  periods
- You've seen the finite horizon problem in its **sequential formulation**:

$$v_0(k_0) = \max_{c_t, k_{t+1}} \sum_{t=0}^T \beta^t u(c_t)$$

s.t.  $c_t + k_{t+1} \leq A_t k_t^\alpha + (1 - \delta)k_t$  for all  $t \geq 0$  (7)

- This can be re-written in a **recursive formulation** as:

$$v_t(k) := \max_{c, k'} [u(c) + \beta v_{t+1}(k')] \quad \text{for all } 0 \leq t \leq T$$

s.t.  $c + k' \leq A_t k^\alpha + (1 - \delta)k, \quad v_{T+1}(k) := 0$  (8)

- This is called a **Bellman equation**
- We've turned a  $T$  dimensional optimization problem into a sequence of  $T$  separate 1-dimensional optimization problems
- Note however: we have to solve eq. (8) for many different values of  $k$

## Finite Horizon: General Case

- You can follow this logic through to the general case where we have  $T$  periods
- You've seen the finite horizon problem in its **sequential formulation**:

$$v_0(k_0) = \max_{c_t, k_{t+1}} \sum_{t=0}^T \beta^t u(c_t)$$

s.t.  $c_t + k_{t+1} \leq A_t k_t^\alpha + (1 - \delta)k_t$  for all  $t \geq 0$  (7)

- This can be re-written in a **recursive formulation** as:

$$v_t(k) := \max_{c, k'} [u(c) + \beta v_{t+1}(k')] \quad \text{for all } 0 \leq t \leq T$$

s.t.  $c + k' \leq A_t k^\alpha + (1 - \delta)k, \quad v_{T+1}(k) := 0$  (8)

- This is called a **Bellman equation**
- We've turned a  $T$  dimensional optimization problem into a sequence of  $T$  separate 1-dimensional optimization problems
- Note however: we have to solve eq. (8) for many different values of  $k$

## Finite Horizon: General Case

- You can follow this logic through to the general case where we have  $T$  periods
- You've seen the finite horizon problem in its **sequential formulation**:

$$v_0(k_0) = \max_{c_t, k_{t+1}} \sum_{t=0}^T \beta^t u(c_t)$$

s.t.  $c_t + k_{t+1} \leq A_t k_t^\alpha + (1 - \delta)k_t$  for all  $t \geq 0$  (7)

- This can be re-written in a **recursive formulation** as:

$$v_t(k) := \max_{c, k'} [u(c) + \beta v_{t+1}(k')] \quad \text{for all } 0 \leq t \leq T$$

s.t.  $c + k' \leq A_t k^\alpha + (1 - \delta)k, \quad v_{T+1}(k) := 0$  (8)

- This is called a **Bellman equation**
- We've turned a  $T$  dimensional optimization problem into a sequence of  $T$  separate 1-dimensional optimization problems
- Note however: we have to solve eq. (8) for many different values of  $k$

## Finite Horizon: General Case

- You can follow this logic through to the general case where we have  $T$  periods
- You've seen the finite horizon problem in its **sequential formulation**:

$$v_0(k_0) = \max_{c_t, k_{t+1}} \sum_{t=0}^T \beta^t u(c_t)$$
$$\text{s.t. } c_t + k_{t+1} \leq A_t k_t^\alpha + (1 - \delta)k_t \quad \text{for all } t \geq 0 \quad (7)$$

- This can be re-written in a **recursive formulation** as:

$$v_t(k) := \max_{c, k'} [u(c) + \beta v_{t+1}(k')] \quad \text{for all } 0 \leq t \leq T$$
$$\text{s.t. } c + k' \leq A_t k^\alpha + (1 - \delta)k, \quad v_{T+1}(k) := 0 \quad (8)$$

- This is called a **Bellman equation**
- We've turned a  $T$  dimensional optimization problem into a sequence of  $T$  separate 1-dimensional optimization problems
- Note however: we have to solve eq. (8) for many different values of  $k$

# Backwards Induction

- This suggests that for finite horizon problems, at least, we can solve the problem backwards
  - 1 For  $t = T$ , solve eq. (8) taking  $v_{T+1}(k)$  as given. Save the results
  - 2 Next, for  $t = T - 1$ , solve eq. (8) taking  $v_{t+1}(k)$  as given (you just solved for it in the previous step)
  - 3 Do the same for  $t = T - 2$ , then  $t = T - 3$ , and so on, until we reach  $t = 0$ .
- This algorithm is called **Backwards Induction**. If  $T$  is finite, it is always well-defined, and will always finish
- How you solve for  $v_t$  depends on your preferences / the properties of the problem
  - You can discretise the problem (a grid of  $k_i$  values, and a grid of  $v_{t,i}$  values)
  - You can use a function approximation technique, and use  $\hat{v}_{t+1,i}$  when you solve at time  $t$
  - If you interpolate, you can use faster optimisation methods on the inside maximization problem

# Backwards Induction

- This suggests that for finite horizon problems, at least, we can solve the problem backwards
  - 1 For  $t = T$ , solve eq. (8) taking  $v_{T+1}(k)$  as given. Save the results
  - 2 Next, for  $t = T - 1$ , solve eq. (8) taking  $v_{t+1}(k)$  as given (you just solved for it in the previous step)
  - 3 Do the same for  $t = T - 2$ , then  $t = T - 3$ , and so on, until we reach  $t = 0$ .
- This algorithm is called **Backwards Induction**. If  $T$  is finite, it is always well-defined, and will always finish
- How you solve for  $v_t$  depends on your preferences / the properties of the problem
  - You can discretise the problem (a grid of  $k_i$  values, and a grid of  $v_{t,i}$  values)
  - You can use a function approximation technique, and use  $\hat{v}_{t+1,i}$  when you solve at time  $t$
  - If you interpolate, you can use faster optimisation methods on the inside maximization problem

# Backwards Induction

- This suggests that for finite horizon problems, at least, we can solve the problem backwards
  - 1 For  $t = T$ , solve eq. (8) taking  $v_{T+1}(k)$  as given. Save the results
  - 2 Next, for  $t = T - 1$ , solve eq. (8) taking  $v_{t+1}(k)$  as given (you just solved for it in the previous step)
  - 3 Do the same for  $t = T - 2$ , then  $t = T - 3$ , and so on, until we reach  $t = 0$ .
- This algorithm is called **Backwards Induction**. If  $T$  is finite, it is always well-defined, and will always finish
- How you solve for  $v_t$  depends on your preferences / the properties of the problem
  - You can discretise the problem (a grid of  $k_i$  values, and a grid of  $v_{t,i}$  values)
  - You can use a function approximation technique, and use  $\hat{v}_{t+1,i}$  when you solve at time  $t$
  - If you interpolate, you can use faster optimisation methods on the inside maximization problem

# Backwards Induction

- This suggests that for finite horizon problems, at least, we can solve the problem backwards
  - 1 For  $t = T$ , solve eq. (8) taking  $v_{T+1}(k)$  as given. Save the results
  - 2 Next, for  $t = T - 1$ , solve eq. (8) taking  $v_{t+1}(k)$  as given (you just solved for it in the previous step)
  - 3 Do the same for  $t = T - 2$ , then  $t = T - 3$ , and so on, until we reach  $t = 0$ .
- This algorithm is called **Backwards Induction**. If  $T$  is finite, it is always well-defined, and will always finish
- How you solve for  $v_t$  depends on your preferences / the properties of the problem
  - You can discretise the problem (a grid of  $k_i$  values, and a grid of  $v_{t,i}$  values)
  - You can use a function approximation technique, and use  $\hat{v}_{t+1,i}$  when you solve at time  $t$
  - If you interpolate, you can use faster optimisation methods on the inside maximization problem

# Backwards Induction

- This suggests that for finite horizon problems, at least, we can solve the problem backwards
  - ① For  $t = T$ , solve eq. (8) taking  $v_{T+1}(k)$  as given. Save the results
  - ② Next, for  $t = T - 1$ , solve eq. (8) taking  $v_{t+1}(k)$  as given (you just solved for it in the previous step)
  - ③ Do the same for  $t = T - 2$ , then  $t = T - 3$ , and so on, until we reach  $t = 0$ .
- This algorithm is called **Backwards Induction**. If  $T$  is finite, it is always well-defined, and will always finish
- How you solve for  $v_t$  depends on your preferences / the properties of the problem
  - You can discretise the problem (a grid of  $k_i$  values, and a grid of  $v_{t,i}$  values)
  - You can use a function approximation technique, and use  $\hat{v}_{t+1,i}$  when you solve at time  $t$
  - If you interpolate, you can use faster optimisation methods on the inside maximization problem

# Backwards Induction

- This suggests that for finite horizon problems, at least, we can solve the problem backwards
  - ① For  $t = T$ , solve eq. (8) taking  $v_{T+1}(k)$  as given. Save the results
  - ② Next, for  $t = T - 1$ , solve eq. (8) taking  $v_{t+1}(k)$  as given (you just solved for it in the previous step)
  - ③ Do the same for  $t = T - 2$ , then  $t = T - 3$ , and so on, until we reach  $t = 0$ .
- This algorithm is called **Backwards Induction**. If  $T$  is finite, it is always well-defined, and will always finish
- How you solve for  $v_t$  depends on your preferences / the properties of the problem
  - You can discretise the problem (a grid of  $k_i$  values, and a grid of  $v_{t,i}$  values)
  - You can use a function approximation technique, and use  $\hat{v}_{t+1,i}$  when you solve at time  $t$
  - If you interpolate, you can use faster optimisation methods on the inside maximization problem

# Backwards Induction

- This suggests that for finite horizon problems, at least, we can solve the problem backwards
  - ① For  $t = T$ , solve eq. (8) taking  $v_{T+1}(k)$  as given. Save the results
  - ② Next, for  $t = T - 1$ , solve eq. (8) taking  $v_{t+1}(k)$  as given (you just solved for it in the previous step)
  - ③ Do the same for  $t = T - 2$ , then  $t = T - 3$ , and so on, until we reach  $t = 0$ .
- This algorithm is called **Backwards Induction**. If  $T$  is finite, it is always well-defined, and will always finish
- How you solve for  $v_t$  depends on your preferences / the properties of the problem
  - You can discretise the problem (a grid of  $k_i$  values, and a grid of  $v_{t,i}$  values)
  - You can use a function approximation technique, and use  $\hat{v}_{t+1,i}$  when you solve at time  $t$
  - If you interpolate, you can use faster optimisation methods on the inside maximization problem

## Generalizing to Other Models

- I've shown you this for just the neoclassical growth model, but this all generalizes to a very wide class of models
- It will work for anything that you can write as:

$$\begin{aligned} v_t(s) &= \max_x f(s, x) + \beta v_{t+1}(s') \\ \text{s.t. } s' &= g(s, x), \quad x \in \mathcal{D}(x) \end{aligned} \tag{9}$$

- ▶  $s$  denotes the **state variables** (carried from one period to the next)
  - ▶  $x$  denotes the **control variables** (picked by the decision maker)
  - ▶  $f(s, x)$  denotes the **flow value** (profits, utility, etc.)
  - ▶  $g(s, x)$  denotes the **law of motion** for the state variables
  - ▶  $\mathcal{D}(x)$  denotes the **decision set** (or constraint set) of our decision maker
- The key trick to writing a problem recursively is to think carefully about which variables are control variables, and which ones are state variables

## Section 2

# Infinite Horizon Dynamic Programs

## Infinite horizon case

- Before, you've seen the neoclassical growth model written in its infinite horizon formulation:

$$v(k_0) = \max_{c_t, k_{t+1}} \sum_{t=0}^{\infty} \beta^t u(c_t)$$

s.t.  $c_t + k_{t+1} \leq A_t k_t^\alpha + (1 - \delta)k_t$  for all  $t \geq 0$  (10)

- It turns out the two-stage budgeting logic works here as well:

$$v(k) = \max_{c, k'} u(c) + \beta v(k')$$

s.t.  $c + k' \leq Ak^\alpha + (1 - \delta)k$  (11)

- Note: for simplicity, I've assumed that  $A$  is constant here.

Otherwise, we would need  $A$  to be a state variable, or do something to transform the problem along the balanced growth path.

## Infinite horizon case: Why it works

To see (intuitively) why this works, let  $BC(k)$  encode the budget constraint set with starting capital stock  $k$

$$v(k_0) = \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_t) \quad \text{Rewrite eq. (10)}$$

$$= \max_{(c_t, k_{t+1}) \in BC(k_t)} \left[ u(c_0) + \sum_{t=1}^{\infty} \beta^t u(c_t) \right] \quad \text{Split off } t = 0$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=1}^{\infty} \beta^{t-1} u(c_t) \right) \right] \quad \text{Factor } \beta \text{ and split the max}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_{t+1}) \right) \right] \quad \text{Reindex the sum}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} [u(c_0) + \beta v(k_1)] \quad \text{Substitute def of } v$$

We use Contraction Mapping Theorem to prove that the recursively defined  $v$  agrees with the sequentially defined  $v$ , but it is beyond the scope of this course. Please just trust me.

## Infinite horizon case: Why it works

To see (intuitively) why this works, let  $BC(k)$  encode the budget constraint set with starting capital stock  $k$

$$v(k_0) = \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_t) \quad \text{Rewrite eq. (10)}$$

$$= \max_{(c_t, k_{t+1}) \in BC(k_t)} \left[ u(c_0) + \sum_{t=1}^{\infty} \beta^t u(c_t) \right] \quad \text{Split off } t = 0$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=1}^{\infty} \beta^{t-1} u(c_t) \right) \right] \quad \text{Factor } \beta \text{ and split the max}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_{t+1}) \right) \right] \quad \text{Reindex the sum}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta v(k_1) \right] \quad \text{Substitute def of } v$$

We use Contraction Mapping Theorem to prove that the recursively defined  $v$  agrees with the sequentially defined  $v$ , but it is beyond the scope of this course. Please just trust me.

## Infinite horizon case: Why it works

To see (intuitively) why this works, let  $BC(k)$  encode the budget constraint set with starting capital stock  $k$

$$v(k_0) = \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_t) \quad \text{Rewrite eq. (10)}$$

$$= \max_{(c_t, k_{t+1}) \in BC(k_t)} \left[ u(c_0) + \sum_{t=1}^{\infty} \beta^t u(c_t) \right] \quad \text{Split off } t = 0$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=1}^{\infty} \beta^{t-1} u(c_t) \right) \right] \quad \text{Factor } \beta \text{ and split the max}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_{t+1}) \right) \right] \quad \text{Reindex the sum}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta v(k_1) \right] \quad \text{Substitute def of } v$$

We use Contraction Mapping Theorem to prove that the recursively defined  $v$  agrees with the sequentially defined  $v$ , but it is beyond the scope of this course. Please just trust me.

## Infinite horizon case: Why it works

To see (intuitively) why this works, let  $BC(k)$  encode the budget constraint set with starting capital stock  $k$

$$v(k_0) = \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_t) \quad \text{Rewrite eq. (10)}$$

$$= \max_{(c_t, k_{t+1}) \in BC(k_t)} \left[ u(c_0) + \sum_{t=1}^{\infty} \beta^t u(c_t) \right] \quad \text{Split off } t = 0$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=1}^{\infty} \beta^{t-1} u(c_t) \right) \right] \quad \text{Factor } \beta \text{ and split the max}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_{t+1}) \right) \right] \quad \text{Reindex the sum}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} [u(c_0) + \beta v(k_1)] \quad \text{Substitute def of } v$$

We use Contraction Mapping Theorem to prove that the recursively defined  $v$  agrees with the sequentially defined  $v$ , but it is beyond the scope of this course. Please just trust me.

## Infinite horizon case: Why it works

To see (intuitively) why this works, let  $BC(k)$  encode the budget constraint set with starting capital stock  $k$

$$v(k_0) = \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_t) \quad \text{Rewrite eq. (10)}$$

$$= \max_{(c_t, k_{t+1}) \in BC(k_t)} \left[ u(c_0) + \sum_{t=1}^{\infty} \beta^t u(c_t) \right] \quad \text{Split off } t = 0$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=1}^{\infty} \beta^{t-1} u(c_t) \right) \right] \quad \text{Factor } \beta \text{ and split the max}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} \left[ u(c_0) + \beta \left( \max_{(c_t, k_{t+1}) \in BC(k_t)} \sum_{t=0}^{\infty} \beta^t u(c_{t+1}) \right) \right] \quad \text{Reindex the sum}$$

$$= \max_{(c_0, k_1) \in BC(k_0)} [u(c_0) + \beta v(k_1)] \quad \text{Substitute def of } v$$

We use Contraction Mapping Theorem to prove that the recursively defined  $v$  agrees with the sequentially defined  $v$ , but it is beyond the scope of this course. Please just trust me.

## How to solve for $v$ : finite horizon logic

- We now have this **recursive formulation** of our problem:

$$\begin{aligned} v(k) &= \max_{c, k'} u(c) + \beta v(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (12)$$

- but how do we actually solve it?
- Let's go back to the finite horizon problem, and imagine that  $T$  is really large:

$$\begin{aligned} v_t(k) &= \max_{c, k'} u(c) + \beta v_{t+1}(k') \quad \text{for all } 0 \leq t \leq T \\ \text{s.t. } c + k' &\leq A_t k^\alpha + (1 - \delta)k, \quad v_{T+1}(k) = h(k) \end{aligned} \quad (13)$$

- How important is the terminal (boundary) condition  $V_{T+1}$  to the solution at  $t = 0$ ?
- Notice that it gets discounted by  $\beta$  every period. If  $\beta < 1$ ,

$$\lim_{T \rightarrow \infty} \beta^{T+1} V_{T+1}(k) = 0 \quad (14)$$

- As  $T$  gets large, the finite horizon problem (at  $t = 0$ ) looks more and more like the infinite horizon problem

## How to solve for $v$ : finite horizon logic

- We now have this **recursive formulation** of our problem:

$$\begin{aligned} v(k) &= \max_{c, k'} u(c) + \beta v(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (12)$$

- but how do we actually solve it?
- Let's go back to the finite horizon problem, and imagine that  $T$  is really large:

$$\begin{aligned} v_t(k) &= \max_{c, k'} u(c) + \beta v_{t+1}(k') \quad \text{for all } 0 \leq t \leq T \\ \text{s.t. } c + k' &\leq A_t k^\alpha + (1 - \delta)k, \quad v_{T+1}(k) = h(k) \end{aligned} \quad (13)$$

- How important is the terminal (boundary) condition  $V_{T+1}$  to the solution at  $t = 0$ ?
- Notice that it gets discounted by  $\beta$  every period. If  $\beta < 1$ ,

$$\lim_{T \rightarrow \infty} \beta^{T+1} V_{T+1}(k) = 0 \quad (14)$$

- As  $T$  gets large, the finite horizon problem (at  $t = 0$ ) looks more and more like the infinite horizon problem

## How to solve for $v$ : finite horizon logic

- We now have this **recursive formulation** of our problem:

$$\begin{aligned} v(k) &= \max_{c, k'} u(c) + \beta v(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (12)$$

- but how do we actually solve it?
- Let's go back to the finite horizon problem, and imagine that  $T$  is really large:

$$\begin{aligned} v_t(k) &= \max_{c, k'} u(c) + \beta v_{t+1}(k') \quad \text{for all } 0 \leq t \leq T \\ \text{s.t. } c + k' &\leq A_t k^\alpha + (1 - \delta)k, \quad v_{T+1}(k) = h(k) \end{aligned} \quad (13)$$

- How important is the terminal (boundary) condition  $V_{T+1}$  to the solution at  $t = 0$ ?
- Notice that it gets discounted by  $\beta$  every period. If  $\beta < 1$ ,

$$\lim_{T \rightarrow \infty} \beta^{T+1} V_{T+1}(k) = 0 \quad (14)$$

- As  $T$  gets large, the finite horizon problem (at  $t = 0$ ) looks more and more like the infinite horizon problem

## How to solve for $v$ : finite horizon logic

- We now have this **recursive formulation** of our problem:

$$\begin{aligned} v(k) &= \max_{c, k'} u(c) + \beta v(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (12)$$

- but how do we actually solve it?
- Let's go back to the finite horizon problem, and imagine that  $T$  is really large:

$$\begin{aligned} v_t(k) &= \max_{c, k'} u(c) + \beta v_{t+1}(k') \quad \text{for all } 0 \leq t \leq T \\ \text{s.t. } c + k' &\leq A_t k^\alpha + (1 - \delta)k, \quad v_{T+1}(k) = h(k) \end{aligned} \quad (13)$$

- How important is the terminal (boundary) condition  $V_{T+1}$  to the solution at  $t = 0$ ?
- Notice that it gets discounted by  $\beta$  every period. If  $\beta < 1$ ,

$$\lim_{T \rightarrow \infty} \beta^{T+1} V_{T+1}(k) = 0 \quad (14)$$

- As  $T$  gets large, the finite horizon problem (at  $t = 0$ ) looks more and more like the infinite horizon problem

## Value Function Iteration: Algorithm

- This suggests a simple approach: define

$$\begin{aligned} v_s(k) &= \max_{c, k'} u(c) + \beta v_{s-1}(k') \\ \text{s.t. } & c + k' \leq Ak^\alpha + (1 - \delta)k \end{aligned} \tag{15}$$

- 1 Start from any “terminal” condition  $v_0(k) = h(k)$  you like
- 2 Solve the model “backwards”, just like when we did backwards induction on the finite horizon problem. That is, for each iteration  $s$ , solve eq. (15) with  $v_{s-1}$  from the previous step
- 3 Stop when  $\|v_s - v_{s-1}\| < \varepsilon$  for some pre-set tolerance level

Note: we’re indexing our iterations forward instead of backwards here, so our boundary condition is at  $s = 0$  not  $t = T$

- This algorithm is called **Value Function Iteration**

# Value Function Iteration

## Convergence and Uniqueness

$$\begin{aligned} v(k) &= \max_{c, k'} u(c) + \beta v(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \tag{12}$$

- You can show that most functions defined this way have a unique solution

The details are complicated, but this will basically be true anytime you have a max operator on the LHS, a well-behaved constraint set, and a discount rate  $\beta < 1$ .

I will not ever ask you about problems where the recursive formulation doesn't yield a unique solution, or where value function iteration fails to converge.

- Moreover, value function iteration converges geometrically to the true solution at a rate proportional to  $\beta$ 
  - When  $\beta$  is close to 1, the problem converges more slowly
- This means that for appropriately defined problems, you can *always* use value function iteration, and it will *always* converge to the **unique solution**.

## Value Function Iteration is Slow

- Usually requires several hundred (or more) iterations to converge
- Inside each iteration, we have to repeatedly solve a costly maximization problem
- Like all the methods we'll see here, suffers badly from the **curse of dimensionality**:
  - Suppose your state space is multi-dimensional. You need to put a grid of values on each dimension.
  - If you have  $n$  dimensions, and your grid  $G$  is

$$G = G_1 \times G_2 \times \cdots \times G_n$$

then the total number of grid points is

$$|G| = \prod_{i=1}^n |G_i|$$

- If  $n$  is 6, and  $|G_i| = 10$  (a coarse grid), then we have to solve 1 million maximization problems at every iteration. The computational burden grows **exponentially**

## Speeding it up: Policy Function Iteration

$$\begin{aligned} v_s(k) &= \max_{c, k'} u(c) + \beta v_{s-1}(k') \\ \text{s.t. } & c + k' \leq Ak^\alpha + (1 - \delta)k \end{aligned} \tag{15}$$

- We started with eq. (15), however, it is very costly to compute the maximisation step
- When we are close to the true solution, the optimal policy will not be changing very much
- **Key Idea:** What if we skipped the maximization step, and just used the optimal  $c^*, k'^*$  from the previous iteration?

## Speeding it up: Policy Function Iteration

$$\begin{aligned} v_s(k) &= \max_{c, k'} u(c) + \beta v_{s-1}(k') \\ \text{s.t. } & c + k' \leq Ak^\alpha + (1 - \delta)k \end{aligned} \tag{15}$$

- We started with eq. (15), however, it is very costly to compute the maximisation step
- When we are close to the true solution, the optimal policy will not be changing very much
- **Key Idea:** What if we skipped the maximization step, and just used the optimal  $c^*, k'^*$  from the previous iteration?

## Speeding it up: Policy Function Iteration

$$\begin{aligned} v_s(k) &= \max_{c, k'} u(c) + \beta v_{s-1}(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (15)$$

- **Key Idea:** What if we skipped the maximization step, and just used the optimal  $c^*, k'^*$  from the previous iteration?
- **Algorithm:**
  - 1 Start from any “terminal” condition  $v_0(k) = h(k)$  you like
  - 2 Solve the model “backwards”, just like when we did backwards induction on the finite horizon problem. I.e., for each iteration  $s$ , solve eq. (15) with  $v_{s-1}$  from the previous step, but **save the optimal policy**  $(c_s^*(k), k_s'^*(k))$  and the solution as  $v_s^0(k)$
  - 3 For  $i = 1, \dots, n$ , set  $v_s^i(k) := u(c_s^*(k)) + \beta v_s^{i-1}(k_s'^*(k))$
  - 4 Set  $v_s(k) := v_s^n(k)$
  - 5 Stop when  $\|v_s - v_{s-1}\| < \varepsilon$  for some preset tolerance level

In general, this tends to be much faster than value function iteration, although you have to be careful. Convergence is *not* guaranteed.

## Speeding it up: Policy Function Iteration

$$\begin{aligned} v_s(k) &= \max_{c, k'} u(c) + \beta v_{s-1}(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (15)$$

- **Key Idea:** What if we skipped the maximization step, and just used the optimal  $c^*, k'^*$  from the previous iteration?
- **Algorithm:**
  - 1 Start from any “terminal” condition  $v_0(k) = h(k)$  you like
  - 2 Solve the model “backwards”, just like when we did backwards induction on the finite horizon problem. I.e., for each iteration  $s$ , solve eq. (15) with  $v_{s-1}$  from the previous step, but **save the optimal policy**  $(c_s^*(k), k_s'^*(k))$  and the solution as  $v_s^0(k)$
  - 3 For  $i = 1, \dots, n$ , set  $v_s^i(k) := u(c_s^*(k)) + \beta v_s^{i-1}(k_s'^*(k))$
  - 4 Set  $v_s(k) := v_s^n(k)$
  - 5 Stop when  $\|v_s - v_{s-1}\| < \varepsilon$  for some preset tolerance level

In general, this tends to be much faster than value function iteration, although you have to be careful. Convergence is *not* guaranteed.

## Speeding it up: Policy Function Iteration

$$\begin{aligned} v_s(k) &= \max_{c, k'} u(c) + \beta v_{s-1}(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (15)$$

- **Key Idea:** What if we skipped the maximization step, and just used the optimal  $c^*, k'^*$  from the previous iteration?
- **Algorithm:**
  - 1 Start from any “terminal” condition  $v_0(k) = h(k)$  you like
  - 2 Solve the model “backwards”, just like when we did backwards induction on the finite horizon problem. I.e., for each iteration  $s$ , solve eq. (15) with  $v_{s-1}$  from the previous step, but **save the optimal policy**  $(c_s^*(k), k_s'^*(k))$  and the solution as  $v_s^0(k)$
  - 3 For  $i = 1, \dots, n$ , set  $v_s^i(k) := u(c_s^*(k)) + \beta v_s^{i-1}(k_s'^*(k))$
  - 4 Set  $v_s(k) := v_s^n(k)$
  - 5 Stop when  $\|v_s - v_{s-1}\| < \varepsilon$  for some preset tolerance level

In general, this tends to be much faster than value function iteration, although you have to be careful. Convergence is *not* guaranteed.

## Speeding it up: Policy Function Iteration

$$\begin{aligned} v_s(k) &= \max_{c, k'} u(c) + \beta v_{s-1}(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (15)$$

- **Key Idea:** What if we skipped the maximization step, and just used the optimal  $c^*, k'^*$  from the previous iteration?
- **Algorithm:**
  - 1 Start from any “terminal” condition  $v_0(k) = h(k)$  you like
  - 2 Solve the model “backwards”, just like when we did backwards induction on the finite horizon problem. I.e., for each iteration  $s$ , solve eq. (15) with  $v_{s-1}$  from the previous step, but **save the optimal policy**  $(c_s^*(k), k_s'^*(k))$  and the solution as  $v_s^0(k)$
  - 3 For  $i = 1, \dots, n$ , set  $v_s^i(k) := u(c_s^*(k)) + \beta v_s^{i-1}(k_s'^*(k))$
  - 4 Set  $v_s(k) := v_s^n(k)$
  - 5 Stop when  $\|v_s - v_{s-1}\| < \varepsilon$  for some preset tolerance level

In general, this tends to be much faster than value function iteration, although you have to be careful. Convergence is *not guaranteed*.

## Speeding it up: Policy Function Iteration

$$\begin{aligned} v_s(k) &= \max_{c, k'} u(c) + \beta v_{s-1}(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (15)$$

- **Key Idea:** What if we skipped the maximization step, and just used the optimal  $c^*, k'^*$  from the previous iteration?
- **Algorithm:**
  - 1 Start from any “terminal” condition  $v_0(k) = h(k)$  you like
  - 2 Solve the model “backwards”, just like when we did backwards induction on the finite horizon problem. I.e., for each iteration  $s$ , solve eq. (15) with  $v_{s-1}$  from the previous step, but **save the optimal policy**  $(c_s^*(k), k_s'^*(k))$  and the solution as  $v_s^0(k)$
  - 3 For  $i = 1, \dots, n$ , set  $v_s^i(k) := u(c_s^*(k)) + \beta v_s^{i-1}(k_s'^*(k))$
  - 4 Set  $v_s(k) := v_s^n(k)$
  - 5 Stop when  $\|v_s - v_{s-1}\| < \varepsilon$  for some preset tolerance level

In general, this tends to be much faster than value function iteration, although you have to be careful. Convergence is *not guaranteed*.

## Speeding it up: Policy Function Iteration

$$\begin{aligned} v_s(k) &= \max_{c, k'} u(c) + \beta v_{s-1}(k') \\ \text{s.t. } c + k' &\leq Ak^\alpha + (1 - \delta)k \end{aligned} \quad (15)$$

- **Key Idea:** What if we skipped the maximization step, and just used the optimal  $c^*, k'^*$  from the previous iteration?
- **Algorithm:**
  - 1 Start from any “terminal” condition  $v_0(k) = h(k)$  you like
  - 2 Solve the model “backwards”, just like when we did backwards induction on the finite horizon problem. I.e., for each iteration  $s$ , solve eq. (15) with  $v_{s-1}$  from the previous step, but **save the optimal policy**  $(c_s^*(k), k_s'^*(k))$  and the solution as  $v_s^0(k)$
  - 3 For  $i = 1, \dots, n$ , set  $v_s^i(k) := u(c_s^*(k)) + \beta v_s^{i-1}(k_s'^*(k))$
  - 4 Set  $v_s(k) := v_s^n(k)$
  - 5 Stop when  $\|v_s - v_{s-1}\| < \varepsilon$  for some preset tolerance level

In general, this tends to be much faster than value function iteration, although you have to be careful. Convergence is *not guaranteed*.

## **Section 3**

# Extensions

# Stochastic Productivity

- Consider the neoclassical growth model, but where  $A$  is a persistent, log-normal shock

$$\begin{aligned} v(k, A) &= \max_{c, k'} u(c) + \beta \mathbb{E}[v(k', A') | A] \\ c + k' &\leq Ak^\alpha + (1 - \delta)k \\ \text{s.t.} \quad \log(A') &= \rho \log(A) + \varepsilon \\ \varepsilon &\sim \mathcal{N}(0, \sigma) \end{aligned} \tag{16}$$

- Key Difference:**  $A$  is a state variable, and we have to take expectations over  $A'$  tomorrow.
- How do we handle the expectations operator?

- Naive Approach:** simply replace expectations with an integral and calculate it numerically in every function call:

$$\mathbb{E}[v(k', A') | A] = \int_{-\infty}^{\infty} v(k', \exp(\rho \log A + \varepsilon)) f(\varepsilon) d\varepsilon \tag{17}$$

where  $f$  is the pdf of  $\varepsilon$

- This will work, but that integral is costly to compute and you will have to calculate it *many, many* times

## Stochastic Productivity

- Consider the neoclassical growth model, but where  $A$  is a persistent, log-normal shock

$$\begin{aligned} v(k, A) &= \max_{c, k'} u(c) + \beta \mathbb{E}[v(k', A') | A] \\ c + k' &\leq Ak^\alpha + (1 - \delta)k \\ \text{s.t.} \quad \log(A') &= \rho \log(A) + \varepsilon \\ \varepsilon &\sim \mathcal{N}(0, \sigma) \end{aligned} \tag{16}$$

- Key Difference:**  $A$  is a state variable, and we have to take expectations over  $A'$  tomorrow.
- How do we handle the expectations operator?
  - Naive Approach:** simply replace expectations with an integral and calculate it numerically in every function call:

$$\mathbb{E}[v(k', A') | A] = \int_{-\infty}^{\infty} v(k', \exp(\rho \log A + \varepsilon)) f(\varepsilon) d\varepsilon \tag{17}$$

where  $f$  is the pdf of  $\varepsilon$

- This will work, but that integral is costly to compute and you will have to calculate it *many, many* times

# Expectations Operator: Better Approach

## Discretise the AR(1)

- Remember from Week 4 that we can discretise an AR(1) process. I.e., we find a grid of  $A_i$   $i=1$  to  $N_A$  and a Markov transition  $P$  matrix such that

$$\Pr(A' = A_i | A_j) = P_{ij}$$

is a good discrete approximation of our process. You can use Tauchen's Method to find this

- Note that I've defined this such that the columns of  $P$  sum to 1 (make sure you check this, otherwise you need to use the transpose of  $P$ )
- Now we can write our expectations operator as

$$\mathbb{E}[v(k', A') | A = A_j] = \sum_{i=1}^{N_A} \Pr(A' = A_i | A_j) v(k', A_i) = \sum_{i=1}^{N_A} v(k', A_i) P_{ij} \quad (18)$$

- Whenever you see a sum like this, you should be thinking about matrix multiplication

# Expectations Operator: Better Approach

## Discretise the AR(1)

- Remember from Week 4 that we can discretise an AR(1) process. I.e., we find a grid of  $A_i$   $i=1$  to  $N_A$  and a Markov transition  $P$  matrix such that

$$\Pr(A' = A_i | A_j) = P_{ij}$$

is a good discrete approximation of our process. You can use Tauchen's Method to find this

- Note that I've defined this such that the columns of  $P$  sum to 1 (make sure you check this, otherwise you need to use the transpose of  $P$ )
- Now we can write our expectations operator as

$$\mathbb{E}[v(k', A') | A = A_j] = \sum_{i=1}^{N_A} \Pr(A' = A_i | A_j) v(k', A_i) = \sum_{i=1}^{N_A} v(k', A_i) P_{ij} \quad (18)$$

- Whenever you see a sum like this, you should be thinking about matrix multiplication

## Expectations as a Matrix Product

- Suppose we want to calculate this expectation for a vector of  $\{k_s\}_{s=1}^{N_k}$
- If we stack them up in a matrix:  $V_{sj} = v(k_s, A_j)$  then we can compute

$$EV := \underbrace{\begin{bmatrix} v(k_1, A_1) & v(k_1, A_2) & \cdots & v(k_1, A_{N_A}) \\ v(k_2, A_1) & v(k_2, A_2) & \cdots & v(k_2, A_{N_A}) \\ \vdots & \vdots & \ddots & \vdots \\ v(k_{N_k}, A_1) & v(k_{N_k}, A_2) & \cdots & v(k_{N_k}, A_{N_A}) \end{bmatrix}}_V \underbrace{\begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,N_A} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,N_A} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N_A,1} & P_{N_A,2} & \cdots & P_{N_A,N_A} \end{bmatrix}}_P$$

- You can check that

$$(EV)_{sj} = \sum_{i=1}^{N_A} v(k_s, A_i) P_{ij} = \sum_{i=1}^{N_A} v(k_s, A_i) \Pr(A' = A_i | A_j) = \mathbb{E}[v(k_s, A') | A_j].$$

- In other words, we can calculate our expectations for all the relevant values of  $k$  just *once* per value function iteration loop
- To evaluate  $k$  off-grid, we can use interpolation *once* on  $EV$  instead of on  $V$
- In general, this delivers huge speed gains

## Expectations as a Matrix Product

- Suppose we want to calculate this expectation for a vector of  $\{k_s\}_{s=1}^{N_k}$
- If we stack them up in a matrix:  $V_{sj} = v(k_s, A_j)$  then we can compute

$$EV := \underbrace{\begin{bmatrix} v(k_1, A_1) & v(k_1, A_2) & \cdots & v(k_1, A_{N_A}) \\ v(k_2, A_1) & v(k_2, A_2) & \cdots & v(k_2, A_{N_A}) \\ \vdots & \vdots & \ddots & \vdots \\ v(k_{N_k}, A_1) & v(k_{N_k}, A_2) & \cdots & v(k_{N_k}, A_{N_A}) \end{bmatrix}}_V \underbrace{\begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,N_A} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,N_A} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N_A,1} & P_{N_A,2} & \cdots & P_{N_A,N_A} \end{bmatrix}}_P$$

- You can check that

$$(EV)_{sj} = \sum_{i=1}^{N_A} v(k_s, A_i) P_{ij} = \sum_{i=1}^{N_A} v(k_s, A_i) \Pr(A' = A_i | A_j) = \mathbb{E}[v(k_s, A') | A_j].$$

- In other words, we can calculate our expectations for all the relevant values of  $k$  just *once* per value function iteration loop
- To evaluate  $k$  off-grid, we can use interpolation *once* on  $EV$  instead of on  $V$
- In general, this delivers huge speed gains

## Expectations as a Matrix Product

- Suppose we want to calculate this expectation for a vector of  $\{k_s\}_{s=1}^{N_k}$
- If we stack them up in a matrix:  $V_{sj} = v(k_s, A_j)$  then we can compute

$$EV := \underbrace{\begin{bmatrix} v(k_1, A_1) & v(k_1, A_2) & \cdots & v(k_1, A_{N_A}) \\ v(k_2, A_1) & v(k_2, A_2) & \cdots & v(k_2, A_{N_A}) \\ \vdots & \vdots & \ddots & \vdots \\ v(k_{N_k}, A_1) & v(k_{N_k}, A_2) & \cdots & v(k_{N_k}, A_{N_A}) \end{bmatrix}}_V \underbrace{\begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,N_A} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,N_A} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N_A,1} & P_{N_A,2} & \cdots & P_{N_A,N_A} \end{bmatrix}}_P$$

- You can check that

$$(EV)_{sj} = \sum_{i=1}^{N_A} v(k_s, A_i) P_{ij} = \sum_{i=1}^{N_A} v(k_s, A_i) \Pr(A' = A_i | A_j) = \mathbb{E}[v(k_s, A') | A_j].$$

- In other words, we can calculate our expectations for all the relevant values of  $k$  just *once* per value function iteration loop
- To evaluate  $k$  off-grid, we can use interpolation *once* on  $EV$  instead of on  $V$
- In general, this delivers huge speed gains