Topics in Macroeconomics

Lectures 4 and 5: Dynamic Programming, Discretization of State Spaces, and Value Function Iterations

Diego de Sousa Rodrigues
de_sousa_rodrigues.diego@uqam.ca

ESG-UQAM

Fall 2025

Road Map

1. Dynamic Programming.
2. Practical Dynamic Programming — VFI (Discrete).
3. Discretizing an AR(1) into a Markov Chain .
4. Practical Dynamic Programming — VFI (Stochastic).

1. Dynamic Programming

Back to Road Map

Infinite Horizon Optimization

1. Time is discrete and the horizon is infinite: $t \in \{0, 1, ...\}$.
2. Single infinitely lived household.
3. Single good representing capital and consumption.
4. Production takes one period.
5. The amount y_t of good produced in period t depends on the stock of capital k_t in this period.

The Model

- We assume that $y_t = F(k_t)$ where $F : \mathbb{R}^+ \to \mathbb{R}^+$ is continuous on $\mathbb{R}^+$, continuously differentiable on $(0, \infty)$, strictly concave, with:

$$F(0) = 0, \quad \forall k > 0, \quad F'(k) > 0, \quad \text{and} \quad \lim_{k \to \infty} F'(k) = 0.$$

The Model

- We assume that $y_t = F(k_t)$ where $F : \mathbb{R}^+ \to \mathbb{R}^+$ is continuous on $\mathbb{R}^+$, continuously differentiable on $(0, \infty)$, strictly concave, with:

$$F(0) = 0, \quad \forall k > 0, \quad F'(k) > 0, \quad \text{and} \quad \lim_{k \to \infty} F'(k) = 0.$$

- In each period t, the output y_t combined with the depreciated capital stock $(1 - \delta)k_t$ can be divided between consumption c_t and new capital stock k_{t+1}.

The Model

- We assume that $y_t = F(k_t)$ where $F : \mathbb{R}^+ \to \mathbb{R}^+$ is continuous on $\mathbb{R}^+$, continuously differentiable on $(0, \infty)$, strictly concave, with:

$$F(0) = 0, \quad \forall k > 0, \quad F'(k) > 0, \quad \text{and} \quad \lim_{k \to \infty} F'(k) = 0.$$

- In each period t, the output y_t combined with the depreciated capital stock $(1 - \delta)k_t$ can be divided between consumption c_t and new capital stock k_{t+1}.
- Implicitly, there is a **one-to-one technology transforming capital in good**.

The Model

- The household's initial stock of capital is $k_0 > 0$.

The Model

- The household's initial stock of capital is $k_0 > 0$.
- His utility from an infinite consumption stream $c = (c_t)_{t \geq 0}$ is:

$$U(c) := \sum_{t=0}^{\infty} \beta^t u(c_t),$$

where $\beta \in (0,1)$ and $u : \mathbb{R}^+ \to \mathbb{R}^+$ is bounded, continuous on $\mathbb{R}^+$, continuously differentiable on $(0, \infty)$, strictly increasing, and strictly concave.

The Set of Alternatives

- We let $f : \mathbb{R}_+ \to \mathbb{R}_+$ be defined by:

$$f(k) := F(k) + (1 - \delta)k.$$

The Set of Alternatives

- We let $f : \mathbb{R}_+ \to \mathbb{R}_+$ be defined by:

$$f(k) := F(k) + (1 - \delta)k.$$

- The household maximizes his discounted utility $U(c)$ among all consumption streams c satisfying the resource feasibility restrictions:

$$c_0 + k_1 \leqslant f(k_0),$$

and for every $t \geqslant 1$,

$$c_t + k_{t+1} \leqslant f(k_t).$$

where $c_t \geqslant 0$ and $k_{t+1} \geqslant 0$.

The Set of Alternatives

- We let $f : \mathbb{R}_+ \to \mathbb{R}_+$ be defined by:

$$f(k) := F(k) + (1 - \delta)k.$$

- The household maximizes his discounted utility $U(c)$ among all consumption streams c satisfying the resource feasibility restrictions:

$$c_0 + k_1 \leqslant f(k_0),$$

and for every $t \geqslant 1$,

$$c_t + k_{t+1} \leqslant f(k_t).$$

where $c_t \geqslant 0$ and $k_{t+1} \geqslant 0$.

- The initial capital stock $k_0 > 0$ is given.
- A key feature in this problem is that it involves **choosing an infinite sequence of consumption or capital stock, one for each t.**

The Set of Alternatives

1. Since the utility function is strictly increasing, we can restrict attention to consumption streams $c = (c_t)\, t \geqslant 0$ satisfying:

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

The Set of Alternatives

1. Since the utility function is strictly increasing, we can restrict attention to consumption streams $c = (c_t)\, t \geqslant 0$ satisfying:

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

2. Therefore, the household's problem is:

$$\max \sum_{t=0}^{\infty} \beta^t u\left(f(k_t) - k_{t+1}\right), \tag{SP}$$

among all capital stock streams $(k_{t+1})_t \geqslant 0$ satisfying $\forall t \geqslant 0, \quad 0 \leqslant k_{t+1} \leqslant f(k_t)$, where k_0 is given.

Solving SP

- We want to solve (SP) by identifying the optimal capital stock stream k^* and the corresponding maximum utility:

$$V^*(k_0) = \sum_{t=0}^{\infty} \beta^t u\left(f\left(k_t^*\right) - k_{t+1}^*\right).$$

Solving SP

- We want to solve (SP) by identifying the optimal capital stock stream k^* and the corresponding maximum utility:

$$V^*(k_0) = \sum_{t=0}^{\infty} \beta^t u\left(f\left(k_t^*\right) - k_{t+1}^*\right).$$

- The problem we need to solve is the following one:

$$\underbrace{V(k_0) = \max_{\{k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u\left(f\left(k_t\right) - k_{t+1}\right).}$$

Solving SP

- We want to solve (SP) by identifying the optimal capital stock stream k^* and the corresponding maximum utility:

$$V^*(k_0) = \sum_{t=0}^{\infty} \beta^t u\left(f\left(k_t^*\right) - k_{t+1}^*\right).$$

- The problem we need to solve is the following one:

$$\underbrace{V(k_0) = \max_{\{k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u\left(f\left(k_t\right) - k_{t+1}\right).}_{V(k_0)=\max_{\{k_{t+1}\}_{t=0}^{\infty}}[u(f(k_0)-k_1)+\beta u(f(k_1)-k_2)+\beta^2 u(f(k_2)-k_3)\ldots]}$$

Solving SP

- Observe:

$$V(k_0) = \max_{\{k_{t+1}\}_{t=0}^{\infty}} \left[u(f(k_0) - k_1) + \beta u(f(k_1) - k_2) + \beta^2 u(f(k_2) - k_3) \ldots \right].$$

- Therefore:

$$V(k_0) = \max_{\{0 \leq k_{t+1} \leq f(k_t)\}_{t=0}^{\infty}} \left\{ u(f(k_0) - k_1) + \sum_{t=1}^{\infty} \beta^t u(f(k_t) - k_{t+1}) \right\}.$$

Ideas Behind the Dynamic Programming

- Since k_t for $t \geq 2$ does not appear in $u(f(k_0) - k_1)$, we can rewrite the problem as:

$$V(k_0) = \max_{k_1} \Big\{ u\big(f(k_0) - k_1\big) + \beta \max_{\{k_{t+1}\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} u\big(f(k_t) - k_{t+1}\big) \Big\}.$$

Ideas Behind the Dynamic Programming

- Since k_t for $t \geq 2$ does not appear in $u(f(k_0) - k_1)$, we can rewrite the problem as:

$$V(k_0) = \max_{k_1} \Big\{ u\big(f(k_0) - k_1\big) + \beta \max_{\{k_{t+1}\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} u\big(f(k_t) - k_{t+1}\big) \Big\}.$$

- Observe:

$$\max_{\{k_{t+1}\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} u\big(f(k_t) - k_{t+1}\big) = \max_{\{k_{t+1}\}_{t=1}^{\infty}} \big[u\big(f(k_1) - k_2\big) + \beta u\big(f(k_2) - k_3\big) + \cdots \big] = V(k_1).$$

Ideas Behind the Dynamic Programming

- Since k_t for $t \geq 2$ does not appear in $u(f(k_0) - k_1)$, we can rewrite the problem as:

$$V(k_0) = \max_{k_1} \Big\{ u\big(f(k_0) - k_1\big) + \beta \max_{\{k_{t+1}\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} u\big(f(k_t) - k_{t+1}\big) \Big\}.$$

- Observe:

$$\max_{\{k_{t+1}\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} u\big(f(k_t) - k_{t+1}\big) = \max_{\{k_{t+1}\}_{t=1}^{\infty}} \big[u\big(f(k_1) - k_2\big) + \beta u\big(f(k_2) - k_3\big) + \cdots \big] = V(k_1).$$

- Therefore:

$$V(k_0) = \max_{0 \leq k_1 \leq f(k_0)} \Big\{ u\big(f(k_0) - k_1\big) + \beta V(k_1) \Big\}.$$

Ideas Behind Dynamic Programming

- The dates $t = 0$ and $t = 1$ are arbitrary, so we denote today's capital by k and next period's by k'. Then:

$$\text{(BE)} \quad V(k) = \max_{0 \le k' \le f(k)} \left\{ u\big(f(k) - k'\big) + \beta\, V(k') \right\}.$$

Ideas Behind Dynamic Programming

- The dates $t = 0$ and $t = 1$ are arbitrary, so we denote today's capital by k and next period's by k'. Then:

$$\text{(BE)} \quad V(k) = \max_{0 \le k' \le f(k)} \left\{ u\big(f(k) - k'\big) + \beta\, V(k') \right\}.$$

- Here, $'$ denotes next-period variables.

Ideas Behind Dynamic Programming

- The dates $t = 0$ and $t = 1$ are arbitrary, so we denote today's capital by k and next period's by k'. Then:

$$\text{(BE)} \quad V(k) = \max_{0 \leq k' \leq f(k)} \left\{ u\big(f(k) - k'\big) + \beta\, V(k') \right\}.$$

- Here, $'$ denotes next-period variables.
- The main idea of dynamic programming is to transform an **intertemporal program** into a **two-period program: today and the future**.

Ideas Behind Dynamic Programming

- The dates $t = 0$ and $t = 1$ are arbitrary, so we denote today's capital by k and next period's by k'. Then:

$$\text{(BE)} \quad V(k) = \max_{0 \le k' \le f(k)} \left\{ u\big(f(k) - k'\big) + \beta\, V(k') \right\}.$$

- Here, $'$ denotes next-period variables.
- The main idea of dynamic programming is to transform an **intertemporal program** into a **two-period program: today and the future**.
- If the value function $V(k)$ were known, we could use (BE) to define a policy function $g(k) = k'$ that attains the maximum in (BE). Given today's capital k, agents choose tomorrow's capital k'.

Ideas Behind the Dynamic Programming

- The main idea was to transform an inter-temporal program into a two period program: **today and the future.**
- Notice that with the Lagrangian approach we had introduced a set of Lagrange Multipliers .
- With the Bellman Equation we introduced a new function, the function V, which is **inter-temporal welfare**.

Recursive structure

- Notice:

$$\text{(BE)} \quad V(k) = \max_{0 \le k' \le f(k)} \Big\{ u\big(f(k) - k'\big) + \beta V(k') \Big\}.$$

Recursive structure

- Notice:

$$\text{(BE)} \quad V(k) = \max_{0 \le k' \le f(k)} \Big\{ u\big(f(k) - k'\big) + \beta V(k') \Big\}.$$

- This determines tomorrow's capital stock as a function of today's capital stock. Specifically:

$$k' = g(k) \in \arg \max_{0 \le x \le f(k)} \Big\{ u\big(f(k) - x\big) + \beta V(x) \Big\}.$$

Recursive structure

- Notice:

$$\text{(BE)} \quad V(k) = \max_{0 \le k' \le f(k)} \Big\{ u\big(f(k) - k'\big) + \beta V(k') \Big\}.$$

- This determines tomorrow's capital stock as a function of today's capital stock. Specifically:

$$k' = g(k) \in \arg\max_{0 \le x \le f(k)} \Big\{ u\big(f(k) - x\big) + \beta V(x) \Big\}.$$

- This function g is known as a **policy function**.

Bellman Equation in Practice

- Suppose further that V is **concave** and **differentiable**. Then the **objective function is the sum of two concave functions and hence is also concave**. So the **optimal policy is characterized by the first order condition**:

$$u'(c) = \beta V\left(k'\right).$$

Bellman Equation in Practice

- Suppose further that V is **concave** and **differentiable**. Then the **objective function is the sum of two concave functions and hence is also concave**. So the **optimal policy is characterized by the first order condition**:

$$u'(c) = \beta V\left(k'\right).$$

- If V were known, we could solve for the policy function by finding the appropriate $k' = g(k)$ for each k such that:

$$u'[f(k) - g(k)] = \beta V'(g(k)).$$

Bellman Equation in Practice

- With the policy function g, we can then construct the entire sequence of capital stocks $k_{t+1} = g(k_t)$ starting with the given initial condition k_0. Then we can back out consumption with $c_t = f(k_t) - g(k_t)$.

Bellman Equation in Practice

- With the policy function g, we can then construct the entire sequence of capital stocks $k_{t+1} = g(k_t)$ starting with the given initial condition k_0. Then we can back out consumption with $c_t = f(k_t) - g(k_t)$.
- Note that **today's capital stock is all that we need to know in order to choose optimal consumption**. In this regard, today's capital stock is the only **state variable** in this problem.

Bellman Equation in Practice

- As already discussed, an optimal policy g(k) is characterized by the FOC:

$$u'[f(k) - g(k)] = \beta V'(g(k)).$$

Bellman Equation in Practice

- As already discussed, an optimal policy g(k) is characterized by the FOC:

$$u'[f(k) - g(k)] = \beta V'(g(k)).$$

- We can make progress by using the **envelope theorem**:

$$V'(k) = u'[f(k) - g(k)]f'(k).$$

  The above is called the envelope condition.

Bellman Equation in Practice

- As already discussed, an optimal policy g(k) is characterized by the FOC:

$$u'[f(k) - g(k)] = \beta V'(g(k)).$$

- We can make progress by using the **envelope theorem**:

$$V'(k) = u'[f(k) - g(k)]f'(k).$$

  The above is called the envelope condition.
- This can be written as:

$$V'(g(k)) = u'[f(g(k)) - g(g(k))]f'(g(k)).$$

Bellman Equation in Practice

- Plugging this into the FOC gives:

$$u'[f(k) - g(k)] = \beta u'[f(g(k)) - g(g(k))]f'(g(k)).$$

Bellman Equation in Practice

- Plugging this into the FOC gives:

$$u'[f(k) - g(k)] = \beta u'[f(g(k)) - g(g(k))]f'(g(k)).$$

- The above is just the standard consumption Euler Equation, which is:

$$u'(c_t) = \beta u'(c_{t+1}) f'(k_{t+1}).$$

Bellman Equation in Practice

- Plugging this into the FOC gives:

$$u'[f(k) - g(k)] = \beta u'[f(g(k)) - g(g(k))]f'(g(k)).$$

- The above is just the standard consumption Euler Equation, which is:

$$u'(c_t) = \beta u'(c_{t+1}) f'(k_{t+1}).$$

- We can solve for the steady state of the model without knowing either the value function or the policy function. In the steady state $k' = k$ so that $k^* = g(k^*)$. Therefore, we have:

$$1 = \beta f'(k^*),$$

which we can do without knowing either V or g.

Bellman Equation in Practice

1. **Take the F.O.C.:**

$$u'(c) = \beta V'\left(k'\right).$$

Bellman Equation in Practice

1. **Take the F.O.C.:**

$$u'(c) = \beta V'\left(k'\right).$$

2. **Consider the Envelop condition:**

$$V'(k) = \alpha k^{\alpha-1} u'(c).$$

Bellman Equation in Practice

1. **Take the F.O.C.:**

$$u'(c) = \beta V'\left(k'\right).$$

2. **Consider the Envelop condition:**

$$V'(k) = \alpha k^{\alpha-1} u'(c).$$

3. **Consider the Envelop condition the next period:**

$$V'\left(k'\right) = \alpha k'^{\alpha-1} u'\left(c'\right).$$

Bellman Equation in Practice

1. **Take the F.O.C.:**

$$u'(c) = \beta V'\left(k'\right).$$

2. **Consider the Envelop condition:**

$$V'(k) = \alpha k^{\alpha-1} u'(c).$$

3. **Consider the Envelop condition the next period:**

$$V'\left(k'\right) = \alpha k'^{\alpha-1} u'\left(c'\right).$$

4. **Substitute for V' in the envelop condition, using the F.O.C.:**

$$u'(c) = \beta\alpha k'^{\alpha-1} u'\left(c'\right).$$

and find the Euler Equation. Sequential representation:

$$u'\left(c_t\right) = \beta\alpha k_{t+1}^{\alpha-1} u'\left(c_{t+1}\right).$$

It is always involves these four steps:

Example 1: Saving choice problem

- **Sequential problem:**

$$\begin{array}{ll} \max_{\{c_t, a_{t+1}\}_{t=0}^{\infty}} & \sum_{t=0}^{\infty} \beta^t u(c_t), \\ a_{t+1} + c_t & = a_t (1 + r_t) + w_t. \end{array}$$

Example 1: Saving choice problem

- **Sequential problem:**

$$\begin{array}{ll} \max_{\{c_t, a_{t+1}\}_{t=0}^{\infty}} & \sum_{t=0}^{\infty} \beta^t u\left(c_t\right), \\ a_{t+1} + c_t & = a_t\left(1 + r_t\right) + w_t. \end{array}$$

- **Recursive problem:**

$$\begin{aligned} V(a) =& u(c) + \beta V\left(a'\right), \\ & \text{s.t. } a' + c = a(1 + r) + w. \end{aligned}$$

Example 1: Saving choice problem

1. **Take the F.O.C.:**

2. **Consider the Envelop condition:**

3. **Consider the Envelop condition the next period:**

4. **Substitute for $V^{'}$ in the envelop condition, using the F.O.C.:**

Example 2: Internal habit formation

- Consider the problem: $u\left(c_t - hc_{t-1}\right)$.

Example 2: Internal habit formation

- Consider the problem: $u\left(c_t - hc_{t-1}\right)$.
- Let's write the Bellman equation recursively. Beginning-of-period wealth is a and past consumption is denoted c_{-1}. The Bellman equation is:

$$V\left(a, c_{-1}\right) = \max_{a'} u\left(c - hc_{t-1}\right) + \beta V\left(a', c\right),$$
$$\text{s.t } a' + c = a(1 + r) + w.$$

Example 2: Internal Habit Formation

1. **Take the F.O.C.:**

2. **Consider the Envelop condition:**

3. **Consider the Envelop condition the next period:**

4. **Substitute for $V^{'}$ in the envelop condition, using the F.O.C.:**

Example 3: Stochastic Bellman Equations

$$V(a) = \max_{a'} u(c) + \beta \mathbb{E} V\left(a'\right),$$
$$a' + c = (1 + r)a + w.$$

Calculations are the same as before with an additional $\mathbb{E}$ operator. Find:

$$u'(c) = \beta \mathbb{E}(1 + r)u'\left(c'\right).$$

Ways of Solving for the Value Function

We will discuss two ways of solving dynamic programming problems. The two ways aim to solve the functional equation for V:

1. **Guess and verify:** this method involves a guess of V and verification of the Bellman equation. Inevitably, it assumes the uniqueness of the solution to the equation.

Ways of Solving for the Value Function

We will discuss two ways of solving dynamic programming problems. The two ways aim to solve the functional equation for V:

1. **Guess and verify:** this method involves a guess of V and verification of the Bellman equation. Inevitably, it assumes the uniqueness of the solution to the equation.
2. **Value function iteration.**

Guess and Verify

Consider an economy in which the representative consumer lives forever. There is a good in each period that can be consumed or saved as capital as well as labor. The consumer's utility function is:

$$\sum_{t=0}^{\infty} \beta^t \log c_t,$$

where $0 < \beta < 1$. The consumer is endowed with 1 unit of labor in each period and with k_0 units of capital in period 0. Feasible allocations satisfy:

$$c_t + k_{t+1} \leq \theta k_t^{\alpha} l_t^{1-\alpha},$$

where $\theta > 1$ and $0 < \alpha < 1$.

1. Write the appropriate **Bellman Equation.**
2. Guess that the solution takes the form $\boldsymbol{V(k) = a_0 + a_1 \log k}$. By using this guess solve the dynamic programming.

Value Function Iteration

1. Recall the Bellman equation:

$$V(k) = \max_{0 \leq k' \leq f(k)} \left\{ u\left[f(k) - k'\right] + \beta V\left(k'\right) \right\}.$$

Value Function Iteration

1. Recall the Bellman equation:

$$V(k) = \max_{0 \leq k' \leq f(k)} \left\{ u\left[f(k) - k'\right] + \beta V\left(k'\right) \right\}.$$

2. Notice the RHS of this equation defines an operator that maps a given function into a new one.

Value Function Iteration

1. Recall the Bellman equation:

$$V(k) = \max_{0 \leq k' \leq f(k)} \left\{ u\left[f(k) - k'\right] + \beta V\left(k'\right) \right\}.$$

2. Notice the RHS of this equation defines an operator that maps a given function into a new one.
3. This operator is often denoted T and we write TV for the new function. That is,

$$TV(k) \equiv \max_{0 \leq k' \leq f(k)} \left\{ u\left[f(k) - k'\right] + \beta V\left(k'\right) \right\}.$$

Value Function Iteration

1. Recall the Bellman equation:

$$V(k) = \max_{0 \leq k' \leq f(k)} \left\{ u\left[f(k) - k'\right] + \beta V\left(k'\right) \right\}.$$

2. Notice the RHS of this equation defines an operator that maps a given function into a new one.
3. This operator is often denoted T and we write TV for the new function. That is,

$$TV(k) \equiv \max_{0 \leq k' \leq f(k)} \left\{ u\left[f(k) - k'\right] + \beta V\left(k'\right) \right\}.$$

4. Then the Bellman equation implies $V = TV$. The value function V is in fact a fixed point of the operator equation.

Procedure to Value Function Iteration

1. Guess some initial value function, V_0.
2. Use T to construct a new value function $V_1 = TV_0$.

$$V_1(k) = TV_0(k) = \max_{0 \leq k' \leq f(k)} \left\{ u\left[f(k) - k'\right] + \beta V_0\left(k'\right) \right\}.$$

   If V_1 is sufficiently close to V_0 according to some metric, you have got the fixed point and stop here. Otherwise,

3. Keep iterating the value functions $V_{j+1} = TV_j = T^j V_0$. Under suitable regularity conditions, $T^j V_0$ converges to a unique V as $j \to \infty$. The limit is the fixed point and is independent of the initial guess V_0.

Conditions for the Fixed Point

- This iterative procedure works especially well when the operator T is a **contraction mapping.**
- A **contraction mapping** T has the property that for two functions V and W the distance between TV and TW is less than the distance between V and W . That is, using a contracting operator brings functions closer and closer.

Conditions for the Fixed Point

- This iterative procedure works especially well when the operator T is a **contraction mapping.**
- A **contraction mapping** T has the property that for two functions V and W the distance between TV and TW is less than the distance between V and W . That is, using a contracting operator brings functions closer and closer.
- **Sufficient conditions for the operator T to be a contraction mapping are:**
  1. the operator is **monotone**, so if two functions V , W satisfy $V \leq W$, then $TV \leq TW$; and
  2. the operator exhibits **discounting**, so if $a > 0$ is a constant and V is a function then $T(V + a) \leq TV + \beta a$ for some constant $\beta \in (0, 1)$.
- **Is the Bellman equation for the growth model as contraction?**

2. Practical Dynamic Programing — VFI (Discrete)

Back to Road Map

Practical Dynamic Programming:

- Consider our growth model with $u(c) = \log c$ and $f(k) = Ak^{\alpha}$ and define the operator:

$$V_j(k) = \max_{k'} \left\{ \log \left(Ak^{\alpha} - k' \right) + \beta V_{j-1} \left(k' \right) \right] \Bigg\}.$$

Practical Dynamic Programming:

- Consider our growth model with $u(c) = \log c$ and $f(k) = Ak^{\alpha}$ and define the operator:

$$V_j(k) = \max_{k'} \left\{ \log \left(Ak^{\alpha} - k' \right) + \beta V_{j-1} \left(k' \right) \right] \Bigg\}.$$

- We can find the solution to this problem **analytically**:
  1. Start with $V_0(k) = 0$. Solve one period problem. $V_1(k) = \max_{k'} \ln \left(Ak^{\alpha} - k' \right)$. Solution, $k' = 0$ and $c = Ak^{\alpha}$. So, $V_1(k) = \ln A + \alpha \ln k$.

Practical Dynamic Programming:

- Consider our growth model with $u(c) = \log c$ and $f(k) = Ak^{\alpha}$ and define the operator:

$$V_j(k) = \max_{k'} \left\{ \log\left(Ak^{\alpha} - k'\right) + \beta V_{j-1}\left(k'\right) \right] \Bigg\}.$$

- We can find the solution to this problem **analytically**:
  1. Start with $V_0(k) = 0$. Solve one period problem. $V_1(k) = \max_{k'} \ln\left(Ak^{\alpha} - k'\right)$. Solution, $k' = 0$ and $c = Ak^{\alpha}$. So, $V_1(k) = \ln A + \alpha \ln k$.
  2. $V_2(k) = \max_{k'} \left\{ \ln\left(Ak^{\alpha} - k'\right) + \beta\left(\ln A + \alpha \ln k'\right) \right\}$. Solution: $k' = \frac{\beta\alpha}{1+\beta\alpha} Ak^{\alpha}$, and $V_2(k) = (1+\beta)\ln A + \beta\alpha \ln \beta\alpha A - (1+\beta\alpha)\ln(1+\beta\alpha) + \alpha(1+\beta\alpha)\ln k$.
  3. By keeping iterating until the limit we reach $k' = \alpha\beta Ak^{\alpha}$.

Practical Dynamic Programming

The problem is not always that we can found an analytical solution, so we need a computer.

How to write a computer code

- Discretize the state space: $k_1 < k_2 < \ldots < k_n$; $V_j(k) = \max_{k'}$:

$$\begin{bmatrix} \log\left(Ak_1^{\alpha} - k_1'\right) + \beta V_{j-1}\left(k_1'\right) & \log\left(Ak_1^{\alpha} - k_2'\right) + \beta V_{j-1}\left(k_2'\right) & \ldots & \log\left(Ak_1^{\alpha} - k_n'\right) + \beta V_{j-1}\left(k_n'\right) \\ \log\left(Ak_2^{\alpha} - k_1'\right) + \beta V_{j-1}\left(k_1'\right) & \log\left(Ak_2^{\alpha} - k_2'\right) + \beta V_{j-1}\left(k_2'\right) & \ldots & \log\left(Ak_2^{\alpha} - k_n'\right) + \beta V_{j-1}\left(k_n'\right) \\ \log\left(Ak_n^{\alpha} - k_1'\right) + \beta V_{j-1}\left(k_1'\right) & \log\left(Ak_n^{\alpha} - k_2'\right) + \beta V_{j-1}\left(k_2'\right) & \ldots & \log\left(Ak_n^{\alpha} - k_n'\right) + \beta V_{j-1}\left(k_n'\right) \end{bmatrix}.$$

Practical Dynamic Programming

The problem is not always that we can found an analytical solution, so we need a computer.

How to write a computer code

- Discretize the state space: $k_1 < k_2 < \ldots < k_n$; $V_j(k) = \max_{k'}$:

$$\begin{bmatrix} \log\left(Ak_1^{\alpha} - k_1'\right) + \beta V_{j-1}\left(k_1'\right) & \log\left(Ak_1^{\alpha} - k_2'\right) + \beta V_{j-1}\left(k_2'\right) & \ldots & \log\left(Ak_1^{\alpha} - k_\eta'\right) + \beta V_{j-1}\left(k_\eta'\right) \\ \log\left(Ak_2^{\alpha} - k_1'\right) + \beta V_{j-1}\left(k_1'\right) & \log\left(Ak_2^{\alpha} - k_2'\right) + \beta V_{j-1}\left(k_2'\right) & \ldots & \log\left(Ak_2^{\alpha} - k_\eta'\right) + \beta V_{j-1}\left(k_\eta'\right) \\ \log\left(Ak_n^{\alpha} - k_1'\right) + \beta V_{j-1}\left(k_1'\right) & \log\left(Ak_n^{\alpha} - k_2'\right) + \beta V_{j-1}\left(k_2'\right) & \ldots & \log\left(Ak_n^{\alpha} - k_n'\right) + \beta V_{j-1}\left(k_n'\right) \end{bmatrix}.$$

- Start with $V_0(k) = 0$.
- Find $V_1(k)$.Check if $|V_1(k) - V_0(k)| < \varepsilon$;
- If not ,find $V_2(k)$. Keep iterating until $|V_n(k) - V_{n-1}(k)| < \varepsilon$.

How to solve the iterations process using computer

The goal is to translate the **Dynamic Programming into a computer code**:

1. Set parameters α, β and A and compute the steady state $\bar{k} = [\alpha\beta A]^{\frac{1}{1-\alpha}}$.
2. Grid the vector $k : k \in [k_1 < k_2 < \ldots < k_m]$ with $k_1 > 0$.

How to solve the iterations process using computer

The goal is to translate the **Dynamic Programming into a computer code**:

1. Set parameters α, β and A and compute the steady state $\bar{k} = [\alpha\beta A]^{\frac{1}{1-\alpha}}$.
2. Grid the vector $k : k \in [k_1 < k_2 < \ldots < k_m]$ with $k_1 > 0$.
3. Given the grid for k,compute the grid for $c = Ak^{\alpha} - k$. Matrix, with dimensions mXm:

$$\begin{bmatrix} c_{11} & \cdots & c_{1j} & \cdots & c_{1m} \\ \vdots & & & & \vdots \\ c_{i1} & & c_{ij} & & c_{im} \\ \vdots & & & & \vdots \\ c_{m1} & \cdots & c_{mj} & \cdots & c_{mm} \end{bmatrix}.$$

The rows represent k and the columns k', such that:

$$c_{ij} = Ak_i^{\alpha} - k_j^{'}$$

Solving with iterations

To build the previous matrix we can do a loop using any computer language:

Given the consumption matrix c, create the utility matrix, U, of dimensions mXm.

Value Function Iteration

1. Initialize the value function, V, by defining a vector of zeros, with dimensions $mX1$.
2. Compute the first period value function and call it TV as follows:

$$\begin{bmatrix} TV(k_1) \\ TV(k_2) \\ \vdots \\ TV(k_m) \end{bmatrix} = \left(\max_{k'} \left\{ \begin{bmatrix} U_{11} & U_{12} & & U_{1m} \\ U_{21} & U_{22} & & \\ & & \ddots & \\ U_{m1} & U_{m2} & & U_{mm} \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1] \end{bmatrix} \right\} \right).$$

Given a k_i (i.e. k) we want to find k_j (i.e. k') that maximizes the expression in the curly brackets. I.e. we choose the column number j, that maximizes the expression for row i. In the code this will look like: V = zeros(m, 1);
o = ones(1, m);
TV = (max(U + beta * V * o));

Value Function Iteration

3. Create a **loop** to iterate on the value function:
   - Set V to be the TV computed from the previous step.
   - Compute a matrix that contains all possible values for $\log c + \beta V$, then of all the elements of the matrix choose the max and call it TV (like before).
   - Then compare your old value function V with the new value function TV for the stopping rule.
   - Stop the loop using a stopping criterion and call the final V^*.

while looping you may wanna to show the number of iterations until convergence.

How to find the Policy Function?

1. Previously we found the value for k' that solves:

$$\boldsymbol{TV} = \max \left(U + \beta V^* \left(k' \right) \right).$$

2. For each column we can save the index where we found the maximum.
3. To find the policy function, use the index and the grid of capital to find the value for k'.

Value Function Iteration

TO MAKE THINGS CLEAR LET'S GIVE A LOOK IN A CODE THAT SOLVES THE VALUE FUNCTION ITERATION.

Practical Dynamic Programming

Now observe the following facts:

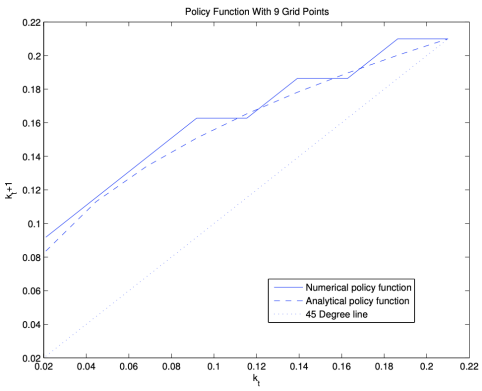

Figure: 9 Grid Points

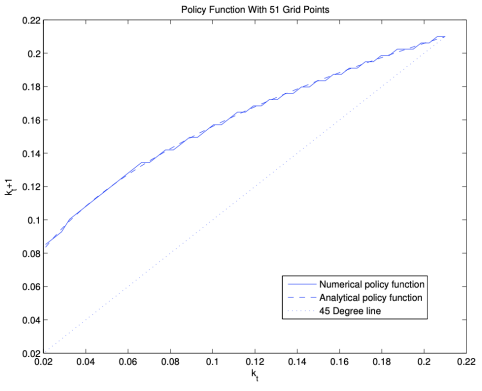

Figure: 51 Grid Points

Stochastic Dynamic Programming

Suppose consumer maximizes:

$$V(k_0, z_0) = \max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} = E_0 \left[\sum_{t=0}^{\infty} \beta^t u(c_t) \right], 0 < \beta < 1,$$

s.t.

$$k_{t+1} + c_t \leq z_t f(k_t),$$

$$z_{t+1} = \rho z_t + \epsilon_{t+1} \quad , \rho \in (0,1), \epsilon \sim F\left(0, \sigma_\epsilon^2\right),$$

$$k_0 > 0 \quad \text{given, and}$$

$$c_t \geq 0.$$

- Note: u(.) and f(.) satisfy standard properties.
- At period t agents know the realization of the shock z_t.

Optimal Growth Problem with Uncertainty

- We can show that:

$$V(k_0, z_0) = \max_{0 \leq k_1 \leq z_0 f(k_0)} \{u(z_0 f(k_0) - k_1) + \beta E_0 [V(k_1, z_1)]\}.$$

Optimal Growth Problem with Uncertainty

- We can show that:

$$V(k_0, z_0) = \max_{0 \leq k_1 \leq z_0 f(k_0)} \{u(z_0 f(k_0) - k_1) + \beta E_0 [V(k_1, z_1)]\}.$$

- Since the problem repeats itself in every period we have:

$$V(k, z) = \max_{0 \leq k' \leq z f(k)} \left\{u\left(z f(k) - k'\right) + \beta E\left[V\left(k', z'\right)\right]\right\}.$$

Optimal Growth Problem with Uncertainty

- We can show that:

$$V(k_0, z_0) = \max_{0 \leq k_1 \leq z_0 f(k_0)} \{u(z_0 f(k_0) - k_1) + \beta E_0 [V(k_1, z_1)]\}.$$

- Since the problem repeats itself in every period we have:

$$V(k, z) = \max_{0 \leq k' \leq z f(k)} \left\{u\left(z f(k) - k'\right) + \beta E\left[V\left(k', z'\right)\right]\right\}.$$

- Notice that z is not our choice variable, since it is exogenous. So, we again have the **stochastic version of the Euler Equation**:

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

Conclusions

- Bellman equations : powerful tool for dynamic economies.
- Structure like two-period economies.
- In complicated environment, the issue is to find the correct state space.
- The interpretation is then simple. It can be computed in more complex environment.

3. Discretizing an AR(1) into a Markov Chain

Back to Road Map

Discretizing an AR(1) with a Finite Markov Chain

In numerical work it is sometimes convenient to replace a continuous model with a discrete one. In particular, Markov chains are routinely generated as discrete approximations to AR(1) processes of the form:

$$y_{t+1} = \mu(1-\rho) + \rho y_t + \varepsilon_{t+1},$$

where ε_{t+1} is assumed to be i.i.d. $\mathcal{N}(0, \sigma^2)$.
Tauchen's method is the most common to approximate this continuous process with a finite Markov chain.[1]

Objective: Discretize the range of y_t into points $y_i,\ i = 1, \ldots, N$ and assign transition probabilities π_{ij}. The transition matrix $\Pi = [\pi_{ij}]$ characterizes the Markov chain.

[1] Tauchen, G. (1986). Finite state markov-chain approximations to univariate and vector autoregressions. Economics Letters, 20(2), 177–181.

Step 1: State-Space Grid

Define two bounds using the unconditional variance $\frac{\sigma^2}{1-\rho^2}$:

$$y_1 = \mu - m\sqrt{\frac{\sigma^2}{1-\rho^2}}, \qquad y_N = \mu + m\sqrt{\frac{\sigma^2}{1-\rho^2}},$$

where m is a scaling parameter (typically $m = 3$ or 4).

Then construct an equispaced grid:

$$\Delta = \frac{y_N - y_1}{N-1}, \qquad y_i = y_1 + (i-1)\Delta, \quad i = 1, \ldots, N.$$

Step 2: Interval Midpoints

Now create the borders of each interval $[y_i, y_{i+1}]$:

$$m_i = \frac{y_{i+1} + y_i}{2} = y_1 + (2i-1)\frac{\Delta}{2} = y_i + \frac{\Delta}{2}, \qquad i = 1, \ldots, N-1.$$

Partition of the real line:

$$y_i \in \begin{cases} (-\infty, m_1] & \text{if } i = 1, \\ (m_{i-1}, m_i] & \text{if } 1 < i < N, \\ (m_{N-1}, \infty) & \text{if } i = N. \end{cases}$$

Step 3: Transition Probabilities (Case $j = 2, \ldots, N-1$)

For $j = 2, \ldots, N-1$:

$$\begin{aligned}
\pi_{ij} &= \Pr\left(y_{t+1} = y_j \,|\, y_t = y_i\right) \\
&= \Pr\left(\mu(1-\rho) + \rho y_i + \varepsilon_{t+1} = y_j\right) \\
&\approx \Pr\left(m_{j-1} \leq \mu(1-\rho) + \rho y_i + \varepsilon_{t+1} \leq m_j\right) \\
&= \Phi\left(\frac{m_j - \rho y_i - \mu(1-\rho)}{\sigma}\right) - \Phi\left(\frac{m_{j-1} - \rho y_i - \mu(1-\rho)}{\sigma}\right),
\end{aligned}$$

where Φ is the c.d.f. of the standard normal distribution.

Step 3: Transition Probabilities (Case $j = 1$)

For $j = 1$:

$$
\begin{aligned}
\pi_{i1} &= \Pr\left(y_{t+1} = y_1 \,|\, y_t = y_i\right) \\
&= \Pr\left(\mu(1-\rho) + \rho y_t + \varepsilon_{t+1} = y_1 \,|\, y_t = y_i\right) \\
&= \Pr\left(\mu(1-\rho) + \rho y_i + \varepsilon_{t+1} = y_1\right) \\
&\approx \Pr\left(\mu(1-\rho) + \rho y_i + \varepsilon_{t+1} \leq m_1\right) \\
&= \Phi\left(\frac{m_1 - \rho y_i - \mu(1-\rho)}{\sigma}\right).
\end{aligned}
$$

Step 3: Transition Probabilities (Case $j = N$)

For $j = N$:

$$\begin{aligned}\pi_{iN} &= \Pr\left(y_{t+1} = y_N \,|\, y_t = y_i\right) \\ &= 1 - \Phi\left(\frac{m_{N-1} - \rho y_i - \mu(1-\rho)}{\sigma}\right).\end{aligned}$$

Resulting Markov Chain

The discretization yields:

- A grid $\{y_i\}_{i=1}^N$,
- A transition matrix Π with rows summing to 1.

Example: $N = m = 3,\ \mu = 1,\ \rho = 0.9,\ \sigma = 0.5$

$$\{y_i\}_{i=1}^3 = \{-2.44,\ 1,\ 4.44\}, \qquad \Pi = \begin{pmatrix} 0.997 & 0.003 & 0 \\ 0.0003 & 0.9994 & 0.0003 \\ 0 & 0.003 & 0.997 \end{pmatrix}.$$

What is a Markov Chain?

Definition. A *Markov chain* is a stochastic process $\{y_t\}_{t=0}^{\infty}$ taking values in a finite or countable set $Y = \{y_1, y_2, \ldots, y_N\}$ such that:

$$\Pr(y_t = y_j \mid y_{t-1}, y_{t-2}, \ldots, y_0) = \Pr(y_t = y_j \mid y_{t-1}) \quad \text{for all } j.$$

Markov property: The distribution of y_t depends only on the most recent past state y_{t-1}, not the full history.

Transition probabilities:

$$\pi_{ij} = \Pr(y_t = y_j \mid y_{t-1} = y_i), \quad \sum_{j=1}^{N} \pi_{ij} = 1.$$

Transition matrix:

$$\Pi = \begin{bmatrix} \pi_{11} & \pi_{12} & \cdots & \pi_{1N} \\ \pi_{21} & \pi_{22} & \cdots & \pi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \pi_{N1} & \pi_{N2} & \cdots & \pi_{NN} \end{bmatrix}$$

Why Markov Chains in Macroeconomics?

- **Discrete approximations of continuous shocks:** AR(1) TFP/income processes are often discretized into a finite-state Markov chain (Tauchen, Rouwenhorst) to solve/estimate models.
- **Regime switching:** Policy or technology regimes (e.g., high/low inflation, ZLB on rates) modeled as state-dependent transitions.
- **Heterogeneity & panel simulation:** Idiosyncratic income/unemployment states in HANK models; simulating cross-sectional distributions.
- **Dynamic programming:** With discrete exogenous states y_t, the Bellman operator and transition operator Π are straightforward to compute.
- **Estimation by simulation:** Simulated moments/likelihoods require long draws from $\{y_t\}$.

Markov Chain: Objects and Notation

State space: $Y = \{y_1, \ldots, y_N\}$.

Transition probabilities:

$$\pi_{ij} \equiv \Pr(y_t = y_j \mid y_{t-1} = y_i), \qquad \sum_{j=1}^{N} \pi_{ij} = 1 \ \ \forall i.$$

Transition matrix:

$$\Pi = \{\pi_{ij}\}_{i,j=1}^{N} \quad \Rightarrow \quad \Pr(y_t = \cdot \mid y_{t-1} = y_i) = \text{row } i \text{ of } \Pi.$$

Stationary distribution (if it exists):

$$\omega \in \Delta^{N-1} \text{ s.t. } \omega = \omega\Pi.$$

Intuition: How Simulation Works

- Given current state $y_{t-1} = y_i$, look at row i of Π: it lists probabilities of jumping to each next state.
- Turn that row into a **CDF** via cumulative sums.
- Draw a uniform $p_t \sim \mathcal{U}[0, 1]$ and pick the first bin where CDF exceeds p_t.

Mini example (3 states):

$$\Pi_{i,\cdot} = (0.2,\ 0.5,\ 0.3) \ \Rightarrow\ \Pi^c_{i,\cdot} = (0.2,\ 0.7,\ 1.0).$$

If $p_t \in (0, 0.2] \Rightarrow y_t = y_1$; if $p_t \in (0.2, 0.7] \Rightarrow y_2$; if $p_t \in (0.7, 1] \Rightarrow y_3$.

Simulating a Markov Chain: Setup

Goal. Simulate a path $\{y_t\}_{t=0}^{T}$ for a chain with state space $Y = \{y_1, \ldots, y_N\}$ and transition matrix Π.

Step 0 (Initial condition).

- Choose initial index i_0 (e.g., fixed state or draw from ω if known).
- Optionally set a seed for reproducibility.
- Plan a burn-in B to reduce dependence on i_0 if you care about stationary moments.

Step 1: Build Row-wise CDFs

Compute the cumulative distribution for each row:

$$\Pi^c_{ij} = \sum_{k=1}^{j} \pi_{ik} \qquad (i = 1, \ldots, N;\ j = 1, \ldots, N).$$

Why? It converts discrete probabilities into bins on $[0, 1]$ to use the inverse-CDF method with uniform draws.

Numerical tip: Ensure each row of Π^c ends at exactly 1 (enforce $\Pi^c_{iN} \leftarrow 1$ to avoid round-off issues).

Step 2: Draw Uniforms

Draw i.i.d. uniforms:

$$\{p_t\}_{t=1}^{T} \sim \text{i.i.d. } \mathcal{U}[0,1].$$

Interpretation. Each p_t selects a bin in the CDF of the current state's row to determine the next state.

Practical tips:

- Vectorize draws (allocate once).
- Use a fixed seed during debugging/replication.
- If simulating many agents, reuse Π^c and draw matrix of uniforms.

Step 3: State Update by Inverse-CDF

Suppose $y_{t-1} = y_i$. Find the smallest $j \in \{1, \ldots, N\}$ such that:

$$\Pi^c_{ij} \geq p_t,$$

then set $y_t = y_j$.

Edge cases. If $p_t \leq \Pi^c_{i1} = \pi_{i1}$, pick $y_t = y_1$. If $p_t > \Pi^c_{i,N-1}$, pick $y_t = y_N$.

Simulation Pseudocode

1. Input Π; compute Π^c by cumulative row sums.
2. Choose $i \leftarrow i_0$; set $y_0 \leftarrow y_{i_0}$.
3. For $t = 1, \ldots, T$:
   - 3.1 Draw $p_t \sim \mathcal{U}[0,1]$.
   - 3.2 Set $j \leftarrow \min\{m : \Pi^c_{im} \geq p_t\}$.
   - 3.3 Record $y_t \leftarrow y_j$; update $i \leftarrow j$.

Good practice: Burn-in B, then compute moments on $\{y_t\}_{t=B+1}^{T}$.

Example: Two-State Business Cycle

States: $Y = \{\text{Recession } (y_1),\ \text{Expansion } (y_2)\}$.

Transition matrix:

$$\Pi = \begin{bmatrix} 0.85 & 0.15 \\ 0.10 & 0.90 \end{bmatrix} \quad \Rightarrow \quad \Pi^c = \begin{bmatrix} 0.85 & 1.00 \\ 0.10 & 1.00 \end{bmatrix}.$$

Update rule: If $y_{t-1} = y_1$ and $p_t \leq 0.85 \Rightarrow y_t = y_1$, else $y_t = y_2$. If $y_{t-1} = y_2$ and $p_t \leq 0.10 \Rightarrow y_t = y_1$, else $y_t = y_2$.

Stationary distribution: $\omega = \omega\Pi \Rightarrow \omega \approx (0.40,\ 0.60)$.

Stationary Distribution: Definition & Intuition

Definition. A probability vector $\omega = (\omega_1, \ldots, \omega_N)$ is a *stationary distribution* for a Markov chain with transition matrix Π if:

$$\omega = \omega\Pi, \qquad \text{with } \omega_j \geq 0 \text{ and } \sum_{j=1}^{N} \omega_j = 1.$$

Intuition. If $y_0 \sim \omega$, then $y_t \sim \omega$ for all t. Under mild conditions (irreducible & aperiodic), the *empirical frequencies* of states in a long simulation converge to ω.

Flow-balance condition. In steady state, the *inflow* to any state equals the *outflow*:

$$\omega_i \pi_{ij} \text{ (flow } i \to j) \quad \Rightarrow \quad \sum_i \omega_i \pi_{ij} \;=\; \omega_j \;\; \forall j.$$

Two-State Example: Calculation & Interpretation

States. $Y = \{\text{Recession } (y_1),\ \text{Expansion } (y_2)\}$:

$$\Pi = \begin{bmatrix} 0.85 & 0.15 \\ 0.10 & 0.90 \end{bmatrix} \quad \Big(\pi_{12} = 0.15,\ \pi_{21} = 0.10\Big).$$

Solve. $\omega = \omega\Pi$ with $\omega_1 + \omega_2 = 1$:

$$\omega_1 = \omega_1\pi_{11} + \omega_2\pi_{21} = \omega_1(1 - \pi_{12}) + \omega_2\pi_{21}$$

$$\Rightarrow \quad \omega_1\pi_{12} = \omega_2\pi_{21} \quad \Rightarrow \quad \frac{\omega_1}{\omega_2} = \frac{\pi_{21}}{\pi_{12}}.$$

With $\omega_1 + \omega_2 = 1$:

$$\boxed{\omega_1 = \frac{\pi_{21}}{\pi_{12} + \pi_{21}} = \frac{0.10}{0.25} = 0.40, \qquad \omega_2 = \frac{\pi_{12}}{\pi_{12} + \pi_{21}} = \frac{0.15}{0.25} = 0.60.}$$

Expected Durations in Each State

General idea. If the process is currently in state y_i, the probability of remaining there next period is π_{ii}. The *expected duration* (average number of consecutive periods in y_i) is:

$$\mathbb{E}[\text{duration of } y_i] = \frac{1}{1-\pi_{ii}}.$$

Derivation.

- Duration follows a *geometric distribution*: $\Pr(\text{stay } k \text{ periods then exit}) = (\pi_{ii})^{k-1}(1-\pi_{ii})$.
- Its expectation is $\sum_{k=1}^{\infty} k(\pi_{ii})^{k-1}(1-\pi_{ii}) = \frac{1}{1-\pi_{ii}}$.

Expected Durations in Each State

General idea. If the process is currently in state y_i, the probability of remaining there next period is π_{ii}. The *expected duration* (average number of consecutive periods in y_i) is:

$$\mathbb{E}[\text{duration of } y_i] = \frac{1}{1 - \pi_{ii}}.$$

Derivation.

- Duration follows a *geometric distribution*: $\Pr(\text{stay } k \text{ periods then exit}) = (\pi_{ii})^{k-1}(1 - \pi_{ii})$.
- Its expectation is $\sum_{k=1}^{\infty} k(\pi_{ii})^{k-1}(1 - \pi_{ii}) = \frac{1}{1-\pi_{ii}}$.

Two-state example:

$$\pi_{11} = 0.85 \;\Rightarrow\; \mathbb{E}[\text{Recession}] = \tfrac{1}{0.15} \approx 6.7 \text{ periods.}$$

$$\pi_{22} = 0.90 \;\Rightarrow\; \mathbb{E}[\text{Expansion}] = \tfrac{1}{0.10} = 10 \text{ periods.}$$

Interpretation: On average, recessions last about 7 periods, while expansions last about 10 periods. This helps translate transition probabilities into economically meaningful durations.

Two-State Example: Calculation & Interpretation

Economic interpretation:

- Long-run time shares: economy spends 40% in recession, 60% in expansion.
- Steady flows balance: $\omega_1 \pi_{12} = \omega_2 \pi_{21}$ (recession$\to$expansion equals expansion$\to$recession).
- Expected durations: $\mathbb{E}[\text{length of } y_1] = \frac{1}{1-\pi_{11}} = \frac{1}{0.15} \approx 6.7$; $\mathbb{E}[\text{length of } y_2] = \frac{1}{1-\pi_{22}} = \frac{1}{0.10} = 10$ periods.

From AR(1) to a Markov Chain (Practice)

Continuous process: $y_{t+1} = \mu(1-\rho) + \rho y_t + \varepsilon_{t+1}, \quad \varepsilon_{t+1} \sim \mathcal{N}(0, \sigma^2)$.

Discretization (e.g., Tauchen/Rouwenhorst):

- Build grid $Y = \{y_1, \ldots, y_N\}$ and transition matrix Π that matches moments/persistence.
- Then simulate using the steps above.
- Use $\{y_t\}$ as exogenous state in the model (TFP, income, markups, etc.).

Practical Considerations

- **Ergodicity & mixing:** For long-run moments, ensure chain is irreducible & aperiodic; use sufficiently long T and burn-in B.
- **Initial condition:** If ω unknown, either burn in or approximate ω by power iteration on Π.
- **Numerics:** Guard against rounding ($\Pi^c_{iN} \leftarrow 1$), prefer binary search on CDF rows for speed.
- **Replication:** Fix RNG seed; store code and seed with results.
- **Large-scale panels:** Precompute Π^c once; vectorize uniforms and searches across agents.

MATLAB Utilities Available

`markovappr.m` — Tauchen's discretization of an AR(1)

- **Purpose:** Build grid $Y = \{y_1, \ldots, y_N\}$ and transition matrix Π for $y_{t+1} = \mu(1-\rho) + \rho y_t + \varepsilon_{t+1},\ \varepsilon_{t+1} \sim \mathcal{N}(0, \sigma^2)$.
- **Inputs (typical):** persistence ρ, std. dev. σ, scaling parameter (e.g. m for $m\sigma$ bounds), grid size N.
- **Outputs:** grid vector `Y`, transition matrix `PI`.
- **Example:** `[Y,PI] = markovappr(rho, sigma, m, N);`

`markov.m` — Simulate a Markov chain

- **Purpose:** Generate T realizations $\{y_t\}_{t=1}^{T}$ given transition matrix `PROB`.
- **Inputs (typical):** `PROB` ($N \times N$), horizon `T`, (optional) initial state index `i0`.
- **Outputs:** index path `idx(1:T)` or state path `y(1:T)`.
- **Example:** `idx = markov(PROB, T, i0);` *(convert `idx` to levels via `Y(idx)`)*

4. Practical Dynamic Programming — VFI (Stochastic)

Back to Road Map

Stochastic Dynamic Programming

Suppose consumer maximizes:

$$V(k_0, z_0) = \max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} = E_0 \left[\sum_{t=0}^{\infty} \beta^t u(c_t) \right], 0 < \beta < 1,$$

s.t.

$$k_{t+1} + c_t \leq z_t f(k_t),$$

$$z_{t+1} = \rho z_t + \epsilon_{t+1} \quad , \rho \in (0,1), \epsilon \sim F\left(0, \sigma_\epsilon^2\right),$$

$$k_0 > 0 \quad \text{given, and}$$

$$c_t \geq 0.$$

- Note: u(.) and f(.) satisfy standard properties.
- At period t agents know the realization of the shock z_t.

Optimal Growth Problem with Uncertainty

- We can show that:

$$V(k_0, z_0) = \max_{0 \leq k_1 \leq z_0 f(k_0)} \{u(z_0 f(k_0) - k_1) + \beta E_0 [V(k_1, z_1)]\}.$$

Optimal Growth Problem with Uncertainty

- We can show that:

$$V(k_0, z_0) = \max_{0 \leq k_1 \leq z_0 f(k_0)} \{u(z_0 f(k_0) - k_1) + \beta E_0 [V(k_1, z_1)]\}.$$

- Since the problem repeats itself in every period we have:

$$V(k, z) = \max_{0 \leq k' \leq z f(k)} \left\{u\left(z f(k) - k'\right) + \beta E\left[V\left(k', z'\right)\right]\right\}.$$

Optimal Growth Problem with Uncertainty

- We can show that:

$$V(k_0, z_0) = \max_{0 \leq k_1 \leq z_0 f(k_0)} \{u(z_0 f(k_0) - k_1) + \beta E_0 [V(k_1, z_1)]\}.$$

- Since the problem repeats itself in every period we have:

$$V(k, z) = \max_{0 \leq k' \leq z f(k)} \left\{u\left(z f(k) - k'\right) + \beta E\left[V\left(k', z'\right)\right]\right\}.$$

- Notice that z is not our choice variable, since it is exogenous. So, we again have the **stochastic version of the Euler Equation**:

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

Markov Process

Recall we have the following definition for a Markov Process:

Markov Process

A stochastic variable z_t follows a **first-order Markov chain** if for all $k \geq 1$ for each $i = 1, ..., n$:

$$\text{Prob}\left[z_t = \hat{z}/z_{t-1}, z_{t-2}, \ldots, z_{t-k}\right] = \text{Prob}\left[z_t = \hat{z}/z_{t-1}\right].$$

We can then characterize a $n-$ dimensional Markov Process by the state space $z \in \mathcal{Z} = \{z_1, ..., z_n\}$ and by the nXn transition matrix P, where:

$$P_{ij} = \text{Prob}\left[z_{t+1} = z_j/z_t = z_i\right], \text{ notice that } \sum_{j=1}^{n} P_{ij} = 1.$$

Model with uncertainty

Now consider the model is given by:

$$
\begin{aligned}
V(k_0) &= \max E_0 \sum_{t=0}^{\infty} \beta^t \ln(c_t), \\
k_{t+1} &= A z_t k_t^{\alpha} - c_t, \\
k_0, &: \text{ given }, \\
z_t &: \text{ stochastic shock }.
\end{aligned}
$$

Model with uncertainty

Now consider the model is given by:

$$
\begin{aligned}
V(k_0) &= \max E_0 \sum_{t=0}^{\infty} \beta^t \ln(c_t), \\
k_{t+1} &= A z_t k_t^{\alpha} - c_t, \\
k_0, &: \text{ given }, \\
z_t &: \text{ stochastic shock }.
\end{aligned}
$$

There are **two state variables** in this model: k_t and z_t.

Model with uncertainty

Now consider the model is given by:

$$
\begin{aligned}
V(k_0) &= \max E_0 \sum_{t=0}^{\infty} \beta^t \ln(c_t), \\
k_{t+1} &= A z_t k_t^{\alpha} - c_t, \\
k_0, &: \text{ given }, \\
z_t &: \text{ stochastic shock }.
\end{aligned}
$$

There are **two state variables** in this model: k_t and z_t.
What is z_t?

- It can be an **i.i.d. shock**.
- It can be a **Markov chain**.
- It can be an **autoregressive process** (in this case there is need to approximate).

Computing the model using a Markov Chain

Assume:

$$z_t = \begin{cases} z_1 \\ z_2 \end{cases},$$

with transition matrix:

$$\Pi = \begin{bmatrix} \pi_{11} & \pi_{12} \\ \pi_{21} & \pi_{22} \end{bmatrix},$$

where $\pi_{i1} + \pi_{i2} = 1$ (the row sum up to one).
In what follows we will discuss the steps to solve for the Value Function Iteration.

Algorithm to Value Function Iteration

1. Define the parameters, get the steady state and make the grid (around the steady state).

Algorithm to Value Function Iteration

1. Define the parameters, get the steady state and make the grid (around the steady state).
2. Use a method (i.e., Tauchen method) to obtain the values of z and the transition matrix Π.

Algorithm to Value Function Iteration

1. Define the parameters, get the steady state and make the grid (around the steady state).
2. Use a method (i.e., Tauchen method) to obtain the values of z and the transition matrix Π.
3. Compute 2 matrices for consumption, conditional on the shock, i.e., $C_i = Az_i k^{\alpha} - k^{'}$.

Algorithm to Value Function Iteration

1. Define the parameters, get the steady state and make the grid (around the steady state).
2. Use a method (i.e., Tauchen method) to obtain the values of z and the transition matrix Π.
3. Compute 2 matrices for consumption, conditional on the shock, i.e., $C_i = Az_i k^{\alpha} - k^{'}$.
4. Compute the first value functions like before, by using:

$$TV_1 = \left(\max\left\{U_1 + \beta * \text{zeros}(m,1) * \text{ ones }(1,m)\right\}\right)'.$$
$$TV_2 = \left(\max\left\{U_2 + \beta * \text{zeros}(m,1) * \text{ ones }(1,m)\right\}\right)'.$$

Algorithm to Value Function Iteration

1. Define the parameters, get the steady state and make the grid (around the steady state).
2. Use a method (i.e., Tauchen method) to obtain the values of z and the transition matrix Π.
3. Compute 2 matrices for consumption, conditional on the shock, i.e., $C_i = Az_i k^{\alpha} - k'$.
4. Compute the first value functions like before, by using:

$$TV_1 = (\max\{U_1 + \beta * \text{zeros}(m,1) * \text{ ones }(1,m)\})'.$$
$$TV_2 = (\max\{U_2 + \beta * \text{zeros}(m,1) * \text{ ones }(1,m)\})'.$$

5. Two stopping criteria: while check1 > 0.0001 OR check2 > 0.0001:

$$TV_1 = (\max\{U_1 + \beta * (\pi_{11} V_1 + \pi_{12} V_2) * \text{ ones }(1,m)\})'.$$
$$TV_2 = (\max\{U_2 + \beta * (\pi_{21} V_1 + \pi_{22} V_2) * \text{ ones }(1,m)\})'.$$

Algorithm to Value Function Iteration

6. Compute two policy functions (one for each state z).

Algorithm to Value Function Iteration

6. Compute two policy functions (one for each state z).
7. Simulate the economy many times to get statistics.

Algorithm to Value Function Iteration

6. Compute two policy functions (one for each state z).
7. Simulate the economy many times to get statistics.

- Notice that to simulate, we will need to create a function that generates randomly the two states.
- This can be generated easily by two states, but it can be challenge in case we have a lot of them.

Additional steps

- **Check that the policy functions is not constrained by the discrete state space.** If k' is equal to the highest or the lowest value of capital in the grid for some i, relax the bounds of k and redo the value function iteration.
- **Check the error tolerance is small enough.** If a small reduction in the tolerance level leads to large changes in the value function or in the policy function, then the tolerance is too high.
- **Check whether or not the grid is large enough.** If a change in the grid leads to a substantial difference in the result maybe the grid is too sparse.
- **A good initial guess for the value function can reduce the computational time.**

Value Function Iteration: some comments

1. **Advantages.**
   - Always work. It is stable, so that it converges to the true solution.
2. **Disadvantages.**
   - It is slow. It suffers form the "curse of dimensionality".
   - The main problem in the value function iteration is the maximization part.

Howard's Improvement

Idea

Decision rules converge faster than value functions and maximization is time-consuming. We keep the policy function fixed.

Howard's Improvement

Idea

Decision rules converge faster than value functions and maximization is time-consuming. We keep the policy function fixed.

Howard's improvement reduces the number of times we update the policy function relative to the number of times we update the value function.

1. Guess V_0; then use the operator:

$$V_1 = T[V_0] = \max_{0 \leq k' \leq f(k)} \left\{ u\left(f(k) - k'\right) + \beta V_0\left(k'\right) \right\},$$

to find V_1 and $k' = g(k)$.

- Then for some finite $n_h \in \{1, 2, ..., N_h\}$ iterate:

$$V_1^{n_h} = u(f(k) - h(k)) + \beta V_1^{n_h - 1}(g(k)).$$

.

2. Check if $V_0 \approx V_1^{N_h}$.
3. If not, repeat (1) until $V_n \approx V_{n-1}^{N_h}$.

Value Function Iteration: code

NOW WE WILL SEE TOGETHER A CODE SOLVING THE VALUE FUNCTION ITERATION IN A STOCHASTIC ENVIRONMENT