Dynamic Programming

0.1 Motivation: Why Dynamics Matter

- Many economic problems involve *decisions over time*.
- In static optimization, we choose a single vector x to maximize $f(x)$ once.
- In dynamic optimization, we choose an entire sequence $\{x_t\}_{t=0}^{T}$ to maximize

$$\sum_{t=0}^{T} \beta^t u(s_t, x_t)$$

  subject to evolving constraints.

The key feature of a dynamic problem is **intertemporal dependence** today's decision affects tomorrow's feasible set.

Consider a groundwater extraction problem:

$$\max_{q_1, q_2} \ \pi_1(q_1) + \beta \pi_2(q_2)$$

subject to

$$s_2 = s_1 - q_1, \quad q_t \leq s_t, \quad s_t \geq 0.$$

It will not be optimal to leave any resource in the ground at the end, so $s_2 = 0$ and $q_2 = s_1 - q_1$. Substituting q_2 into the objective gives

$$\max_{q_1} \ \pi_1(q_1) + \beta \pi_2(s_1 - q_1).$$

The first-order conditions give

$$\pi_1'(q_1) - \beta \pi_2'(q_2) = 0,$$

which equates the **marginal benefit of current extraction** to the **discounted marginal benefit of future extraction**.

Examples

- Resource extraction (trade-off between current and future use)
  - Suppose a manager extracts a nonrenewable resource such as groundwater or oil. Each period, extraction reduces the remaining stock. The manager faces a trade-off between the *immediate profit* from extracting today and the *future value* of the remaining stock. Dynamic optimization determines the optimal extraction path balancing current and future gains — the or *shadow value* of the resource.
- Investment and capital accumulation
  - Firms decide how much to invest in physical capital (machinery, buildings) or human capital (training, education). Investment today increases the productive capacity tomorrow, affecting future profits. The firm must weigh the *cost of investment* against the *expected future returns*, leading to an optimal investment strategy over time.

- Crop rotations or livestock feeding
  - Farmers choose how to allocate land and resources over multiple periods. Planting a crop today may deplete soil nutrients, affecting yields in subsequent seasons. Similarly, feeding strategies for livestock impact their growth and productivity over time. Dynamic programming helps determine optimal crop rotations or feeding schedules that maximize long-term farm profitability.
- Forest rotation and timber harvesting
  - Forest managers decide when to harvest trees to maximize the value of timber while ensuring sustainability. Harvesting too early may yield lower quantities of wood, while waiting too long can lead to overgrowth where the marginal wood growth does not justify the cost. Dynamic optimization helps find the optimal rotation period that balances immediate revenue with future discounted value.
- Pollution control and climate policy
  - Governments and firms must decide how much to invest in pollution abatement technologies or carbon reduction strategies. Reducing emissions today can mitigate future environmental damage and associated costs. Dynamic programming helps evaluate the trade-offs between current abatement costs and long-term benefits, guiding optimal policy decisions over time.

0.2 Backward Induction

- In a *finite-horizon sequential decision problem*, one can solve by starting at the last period and then working backward.
- At the final period, the decision is trivial (or known) because nothing is left to optimize afterward.
- Then one moves to the second-to-last period: given that the future is solved optimally, choose your action now to maximize current payoff plus continuation value.
- This process continues backward until reaching the initial period. This is the logic of dynamic programming in finite-horizon settings.
- Backward induction is prefered over brute force enumeration because it simplifies potentially complex problems. Starting from the beginning, the number of paths can explode combinatorially, making it infeasible to evaluate all possible sequences of decisions.
- This was the intuition in the problem above: we used the constraint to eliminate q_2 and reduce the problem to a single decision variable q_1. Because there was no salvage value (no value to not using all of the resource at the end), we could assume that $q_2 = s_2 = s_1 - q_1$.
- In a finite-horizon problem, backward induction works because there is a last period: after that, no decisions remain. We can anchor the recursion at T and move backward.
- In an infinite-horizon problem, there is no terminal period. We need a different anchor. What we look for is a stationary path (or policy) that is internally consistent over time: i.e. at every date, given the current state, the decision you make will also be optimal for the continuation. In other words, you want a path or policy such that there is no incentive to deviate at any point — because the same 'value' structure recurs over time. That is what gives the stability and time-independence of the optimal policy in many infinite-horizon problems.

0.3 Stochastic Dynamic Programming

- In many problems, there is uncertainty about future states or payoffs.
- The state variable s_t may evolve according to a stochastic process, influenced by random shocks.
- The decision-maker must consider the expected value of future payoffs, integrating over possible future states.

0.3.1 Markov property

A problem is **Markovian** if the current state s_t contains all relevant information from the past. The future state depends only on the current state and decision.

Examples

- Stochastic resource growth model:
  Let fish stock evolve as
  $$s_{t+1} = G(s_t) + \varepsilon_{t+1} - h_t$$
  where ε_{t+1} is a random growth shock (possibly discrete). Then transition probabilities $P(s' \mid s, h)$ come from the distribution of ε. Note that s_{t+1} depends only on s_t and h_t, not on earlier states or actions.
- Crop yield shocks in agriculture:
  A farmer chooses input (fertilizer) x_t. The next soil state evolves as
  $$s_{t+1} = g(s_t, x_t)$$
  . The stochastic yield shock induces $P(s_{t+1} \mid s_t, x_t)$.
- Capital accumulation with productivity shocks:
  Suppose production is
  $$y_t = F(k_t, z_t)$$
  where z_t is a stochastic productivity state evolving as a Markov chain (discrete or discretized AR(1)). The decision is investment i_t, state vector is (k_t, z_t), and transitions are given by $\Pr(z_{t+1} \mid z_t)$ combined with deterministic $k_{t+1} = (1-\delta)k_t + i_t$.

0.4 Vocabulary and the components of dynamic programming problem

- **State variable** s_t: summarizes all relevant information at time t needed to make decisions and predict future states. Examples: resource stock, capital stock, productivity level.
- **Control variable** x_t: the decision made at time t. Examples: extraction quantity, investment amount, input usage.
- **Transition function** $s_{t+1} = g(s_t, x_t, \varepsilon_{t+1})$: describes how the state evolves from t to $t+1$ given current state, action, and random shocks.
- **Reward function** $u(s_t, x_t)$: the immediate payoff from taking action x_t in state s_t. Examples: profit, utility, yield.
- **Policy** $\{x_t(s_t)\}$: a rule that specifies the action to take in each state at each time.
- **Value function** $V_t(s_t)$: the maximum expected discounted sum of future rewards starting from state s_t at time t.
- **Discount factor** $\beta \in (0, 1)$: reflects the preference for current rewards over future rewards. It reflects time preference or impatience: how much less future utility / profit is worth relative to now. In growth / resource models, discounting ensures that long-run payoffs are "worth less" the farther they are in the future, which prevents over-emphasis on infinitely deferred rewards.
  - It also enables contraction mappings in infinite-horizon problems.
  - We often assume a constant discount factor for simplicity, but in reality, discounting may vary over time or depend on circumstances.
- **Salvage value**: the value of remaining resources or capital at the end of the planning horizon. In finite-horizon problems, this can be zero or some terminal value function.

0.5 Problem Types

1. Here is a finite time discrete state and time example:
   $$\max_{h_t} \left[\sum_{t=0}^{T-1} \beta^t \pi_t(h_t) + \beta^T V_T(s_T) \right]$$

subject to the simple deterministic transition dynamics:

$$s_{t+1} = s_t - h_t + G(s_t)$$

Identify the components of the problem

- State: s_t (can be either discrete or continuous)
- Control: h_t
- Transition: $s_{t+1} = s_t - h_t + G(s_t)$
- Reward: $\pi_t(h_t)$
- Policy: $h_t(s_t)$
- Value function: $V_t(s_t)$
- Discount factor: β
- Salvage value: $V_T(s_T) = 0$ or some terminal value

2. If time is continuous:

$$\max_{h(t)} \int_0^T e^{-\rho t}\, \pi(s(t), h(t))\, dt \;+\; e^{-\rho T}\, V_T(s(T))$$

subject to

$$\frac{ds}{dt} = G(s(t)) - h(t), \quad s(0) = s_0$$

The components are similar, but the transition is now a differential equation.

3. In an infinite-horizon problem:

$$\max_{\{h_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t \pi(h_t)$$

subject to

$$s_{t+1} = s_t - h_t + G(s_t), \quad s_0 \text{ given}$$

The components are the same, but there is no terminal time or salvage value. The problem is usually well-behaved when $\beta < 1$ and $\pi(h_t)$ is bounded.

---

1 Discrete time and state space

- Dynamic optimization problems describe sequential decision-making over time, where future states depend on current actions.
- In discrete time and discrete state settings, time proceeds in periods $t = 0, 1, \ldots, T$, and the system occupies one of a finite set of states $s_t \in S$ at each time t.
- The decision variable is denoted by $x_t \in X(s_t)$, representing the feasible choices when the system is in state s_t.

1.1 General Problem

- The general **finite-horizon** problem is to choose a sequence $\{x_t\}_{t=0}^{T-1}$ to maximize the expected discounted sum of single-period payoffs:

$$\max_{\{x_t\}_{t=0}^{T-1}} \sum_{t=0}^{T-1} \beta^t \pi(s_t, x_t) + \beta^T V_T(s_T)$$

where:

- $\beta \in (0, 1)$ is the discount factor,

- $\pi(s_t, x_t)$ is the single-period reward (or profit),
- $V_T(s_T)$ is the *terminal value function* (salvage value) at the end of the horizon.

The **transition dynamics** are deterministic:

$$s_{t+1} = f(s_t, x_t),$$

with an initial state s_0 given.

1.2 Bellman Equation

- The recursive formulation expresses the problem in terms of a **value function** $V_t(s_t)$ that gives the maximal attainable value starting from state s_t at time t:

$$V_t(s_t) = \max_{x_t \in X(s_t)} \Big\{ \pi(s_t, x_t) + \beta\, V_{t+1}(f(s_t, x_t)) \Big\}.$$

- The **terminal condition** closes the recursion:

$$V_T(s_T) = \pi_T(s_T) \quad \text{or a given terminal value function.}$$

The Bellman equation expresses the principle of optimality:

> The value of being in state s_t is the maximum over current actions of the immediate payoff plus the discounted value of the next state.

1.2.1 Backward Recursion (Finite Horizon)

To solve:

1. Start from the terminal period T, where $V_T(s_T)$ is known.
2. For each earlier period $t = T-1, \dots, 0$, compute $V_t(s_t)$ from the Bellman equation.
3. The corresponding optimal policy is

$$x_t^*(s_t) = \max_{x_t \in X(s_t)} \Big\{ \pi(s_t, x_t) + \beta\, V_{t+1}(f(s_t, x_t)) \Big\}.$$

This recursion yields both the value function and the optimal policy for each possible state.

1.2.2 Example Template

Once this structure is clear, we can apply it directly to the **mine management problem**, where:

- state: remaining ore s_t, and initial stock $s_0 = 4$,
- control: extraction $x_t \in \{0, 1, \dots, s_t\}$,
- transition: $s_{t+1} = s_t - x_t$,
- reward: $\pi(s_t, x_t) = 10x_t - (2x_t + x_t^2)$,
- terminal condition: $V_T(s_T) = 0$
- horizon: $T = 3$,
- discount factor: $\beta = 0.9$.

Students: set up the problem/Bellman equation and solve via backward induction.

We solve by **backward induction**:

$$V_t(s) = \max_{x \in \{0,\dots,s\}} \{\pi(x) + \beta V_{t+1}(s - x)\}, \quad V_3(s) \equiv 0.$$

Lookup: - $\pi(0) = 0$, $\pi(1) = 7$, $\pi(2) = 12$, $\pi(3) = 15$, $\pi(4) = 16$.

We solve by **backward induction**:

$$V_t(s) = \max_{x \in \{0,\dots,s\}} \{\pi(x) + \beta V_{t+1}(s-x)\}, \quad V_3(s) \equiv 0.$$

1.2.3 Step 1: Final decision period $t = 2$

Here $V_3(\cdot) = 0$, so extract to maximize $\pi(x)$ subject to $x \leq s$:

- $V_2(0) = 0$
- $V_2(1) = \pi(1) = 7$
- $V_2(2) = \pi(2) = 12$
- $V_2(3) = \pi(3) = 15$
- $V_2(4) = \pi(4) = 16$

Optimal action at $t = 2$ is $x_2^*(s) = s$ (extract all that remains).

1.2.4 Step 2: Period $t = 1$

Compute $V_1(s) = \max_{x \leq s}\{\pi(x) + \beta V_2(s-x)\}$:

- $s = 0$: only $x = 0 \Rightarrow V_1(0) = 0$
- $s = 1$:
  - $x = 0$: $0 + 0.9 \cdot V_2(1) = 6.3$
  - $x = 1$: $7 + 0.9 \cdot 0 = 7$
    $\Rightarrow V_1(1) = 7$ at $x = 1$
- $s = 2$:
  - $x = 0$: $0 + 0.9 \cdot 12 = 10.8$
  - $x = 1$: $7 + 0.9 \cdot 7 = 13.3$
  - $x = 2$: $12 + 0 = 12$
    $\Rightarrow V_1(2) = 13.3$ at $x = 1$
- $s = 3$:
  - $x = 0$: $0 + 0.9 \cdot 15 = 13.5$
  - $x = 1$: $7 + 0.9 \cdot 12 = 17.8$
  - $x = 2$: $12 + 0.9 \cdot 7 = 18.3$
  - $x = 3$: $15 + 0 = 15$
    $\Rightarrow V_1(3) = 18.3$ at $x = 2$
- $s = 4$:
  - $x = 0$: $0 + 0.9 \cdot 16 = 14.4$
  - $x = 1$: $7 + 0.9 \cdot 15 = 20.5$

– $x = 2$: $12 + 0.9 \cdot 12 = 22.8$

– $x = 3$: $15 + 0.9 \cdot 7 = 21.3$

– $x = 4$: $16 + 0 = 16$
$\Rightarrow V_1(4) = 22.8$ at $x = 2$

Summary at $t = 1$:
$V_1(0) = 0,\ V_1(1) = 7,\ V_1(2) = 13.3,\ V_1(3) = 18.3,\ V_1(4) = 22.8.$

1.2.5 Step 3: Initial period $t = 0$ (with $s_0 = 4$)

Compute $V_0(4) = \max_{x \leq 4}\{\pi(x) + \beta V_1(4 - x)\}$:

- $x = 0$: $0 + 0.9 \cdot 22.8 = 20.52$
- $x = 1$: $7 + 0.9 \cdot 18.3 = 23.47$
- $x = 2$: $12 + 0.9 \cdot 13.3 = 23.97$
- $x = 3$: $15 + 0.9 \cdot 7 = 21.3$
- $x = 4$: $16 + 0 = 16$

$\Rightarrow$ **Optimal** $x_0^*(4) = 2$ and $V_0(4) = 23.97$.

1.2.6 Optimal path and value

Starting at $s_0 = 4$:

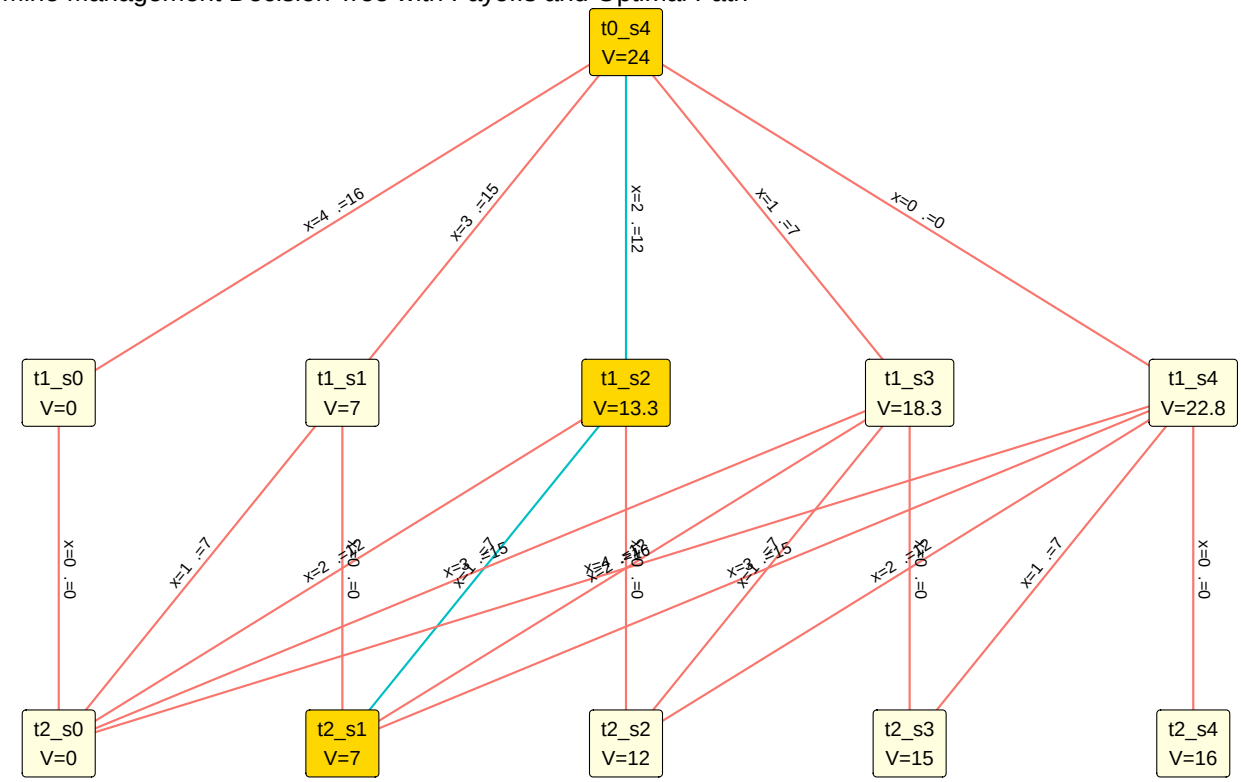
- $t = 0$: extract $x_0 = 2 \Rightarrow s_1 = 2$
- $t = 1$: with $s_1 = 2$, optimal $x_1 = 1 \Rightarrow s_2 = 1$
- $t = 2$: with $s_2 = 1$, extract all $x_2 = 1 \Rightarrow s_3 = 0$

Discounted payoff:

- $t = 0$: $\pi(2) = 12$
- $t = 1$: $\beta\pi(1) = 0.9 \cdot 7 = 6.3$
- $t = 2$: $\beta^2\pi(1) = 0.81 \cdot 7 = 5.67$

Total $= 12 + 6.3 + 5.67 = 23.97 = V_0(4)$. Thus, the optimal extraction policy is to extract 2 units at $t = 0$, 1 unit at $t = 1$, and 1 unit at $t = 2$, yielding a total discounted profit of 23.97.

Mine Management Decision Tree with Payoffs and Optimal Path



See excel example for full table.

1.3 The Curse of Dimensionality

Dynamic programming offers an elegant recursive solution principle, but it suffers from a major practical limitation known as the **curse of dimensionality**.

The term was coined by Richard Bellman to describe the exponential growth in computational complexity as the number of state or control variables increases.

1.3.1 Illustration

In our mine management problem, the state s_t took on a small number of discrete values (e.g., $s_t \in \{0, 1, 2, 3, 4\}$).
If we introduce additional state variables — such as capital, technology, or another resource — the number of possible state combinations grows rapidly.

For example:

Dimension	Grid points per variable	Total grid points
1 state variable	100	100
2 state variables	100	10^4
3 state variables	100	10^6
4 state variables	100	10^8

Even modest discretizations quickly lead to millions of points. At each grid point, we must compute and compare values for all possible controls, making the full state–action space enormous.

Each additional state dimension multiplies the size of:

- The **value function table** $V_t(s)$
- The **policy function** $x_t^*(s)$
- The **transition mapping** $s_{t+1} = f(s_t, x_t)$
- And the computational effort of evaluating $\pi(s_t, x_t) + \beta V_{t+1}(f(s_t, x_t))$

This explosion makes naive enumeration infeasible.

For instance, if we allowed both **ore stock** and **equipment age** as states:

$$s_t = (s_t^{ore}, s_t^{equip}),$$

and if each dimension had only 10 discrete values, then V_t must be evaluated at $10 \times 10 = 100$ points per period. Add one more variable (say, capital), and it jumps to 1,000 points. For higher dimensions, this quickly exceeds computational limits.

The curse of dimensionality is not merely a computational nuisance—it shapes how economists model behavior.

In applied work, we rarely solve high-dimensional dynamic programs exactly. Instead, we rely on:

- **Approximations**: e.g., functional forms or interpolation of $V(s)$
- **Monte Carlo methods**: random sampling of states rather than full enumeration
- **Problem decomposition**: splitting the model into subproblems or exploiting structure (e.g., separability)
- **Aggregation or discretization**: grouping similar states or actions

These methods trade accuracy for tractability.

1.4 Forest Harvest Problem

We consider a simple **finite-horizon dynamic programming** problem for forest management. At each decision date, the forest is characterized by its *age*, and the decision is whether to **cut** or **wait**.

1.4.1 Model Elements

- **State variable**: x_t = forest age at the beginning of period t
- **Decision variable**: $u_t \in \{0, 1\}$
  - $u_t = 0$ means *wait*
  - $u_t = 1$ means *harvest*
- **Payoff**:
  - Depends on age: $x_t = 0$: payoff = 0; $x_t = 50$: payoff = 8; $x_t = 100$: payoff = 10
- **Transition**:
  - If not harvested: $x_{t+1} = \min(x_t + 50, 100)$
  - If harvested: $x_{t+1} = 0$
- **Terminal value**: $V_T(x_T) = \max\{\text{salv}(x_T), \text{pay}(x_T)\}$ where
  - $\text{salv}(x_T)$ = salvage value if not harvested at end (0, 20, 60)
- **Discount factor**: $\beta = (1/1.04)^{50}$ (discounting 50 years at 4%)

1.4.2 Bellman Equation

The Bellman equation for this problem is:

$$V_t(x_t) = \max_{u_t \in \{0,1\}} \left[r(x_t, u_t) + \beta \, V_{t+1}(x_{t+1}) \right],$$

where

- $r(x_t, u_t)$ is the **immediate payoff**, and
- $x_{t+1} = \min(x_t + 50, 100)$ follows the **transition rule**.

We can solve this problem by **backward induction** starting from the terminal period T and moving backward to $t = 0$ without defining initial conditions.

1.4.3 Intuition

- What is the intuition behind the optimal policy? At each age x_t, we compare the value of harvesting now versus waiting for future growth.
- Even though the payoff function is discrete, we can still solve via backward induction by simulating outcomes at each state.

1.4.4 Implementation in R

Go to the R script `forest_harvest.R`

2 Infinite Horizon Problems

- In many economic applications, we consider **infinite-horizon** dynamic programming problems where the decision-maker optimizes over an infinite sequence of periods. Intuitively, the solution to infinite horizon problems often converges to a **steady-state policy** that does not depend on time.
- The general infinite-horizon problem is to choose a sequence $\{x_t\}_{t=0}^{\infty}$ to maximize the expected discounted sum of single-period payoffs:

$$\max_{x_t} \sum_{t=0}^{\infty} \beta^t \pi(s_t, x_t)$$

  subject to the deterministic transition dynamics:

$$s_{t+1} = f(s_t, x_t),$$

  with an initial state s_0 given.
- The Bellman equation for an infinite-horizon problem is:

$$V(s) = \max_{x \in X(s)} \{\pi(s, x) + \beta V(f(s, x))\}$$

  where we note that there is no time index on $V(s)$ since the value function is stationary (time autonomous).
- The key difference from finite-horizon problems is the absence of a terminal condition. Instead, we look for a fixed point of the Bellman operator. We can find a solution using **value function iteration** or **policy iteration** methods. Today, we focus on value function iteration.

2.1 Fishery Problem

We study a single-stock fishery with:

- discrete state $s \in \{0, 1, 2, \dots, S_{\max}\}$ (biomass),
- control (harvest) $h \in= \{0, 1, \dots, s\}$
- next period's stock is $s' = G(s - h)$.
- Per-period payoff is $\pi(s, h) = p\,h - c\,h$ (constant net price for simplicity).

The infinite-horizon Bellman equation (time-stationary) is

$$V(s) = \max_h \{\pi(s, h) + \beta V(s')\} = \max_h \{p\,h - c\,h + \beta V(G(s - h))\}$$

where $s' = G(s - h)$ where we assume a logistic growth function

$$G(s) = s + rs\left(1 - \frac{s}{S_{\max}}\right)$$

with intrinsic growth rate r and carrying capacity $S_{\max}$.

2.1.1 Building Intuition

If we relax to continuous s and h and interior choices (we'll still solve numerically on a grid), the maximizer $h^*(s)$ satisfies the KKT system:

- Feasibility: $0 \le h \le s$.
- Stationarity (interior): $\pi_h(s, h) - \beta V'(s'), G'(s - h) = 0$ with $s' = G(s - h)$.
- Complementary slackness for the bounds $h \ge 0$ and $h \le s$.

With $\pi_h(s,h) = p - c$ and $G'(s-h) = 1 + r(1 - 2c/s_{max})$, the interior FOC is

$$p - c - \beta V'(s')\, G'(s-h) = 0$$

We derive the envelope condition by differentiating the Bellman wrt to s (no direct dependence of π on s here) gives

$$V'(s) = 0 + \beta V'(s')\, G'(s - h^*(s)).$$

Note that the second term of the envelope condition is the same as in the stationary condition above. Combining the two (interior) yields a classic shadow-value condition:

$$p - c = V'(s) \quad \text{(if interior)}.$$

Interpretation: harvest up to the point where the marginal net revenue equals the marginal value of stock - the additional future discounted revenue from the growth of the stock (the shadow price or "user cost"). With bounds, policies tilt to no-harvest at very low s (stock rebuilding) and binding-harvest (take most of the stock) when s is very high.

2.1.2 Numerical Solution via Value Function Iteration

We'll solve by value function iteration on a coarse grid—this is the standard workhorse for discrete-time, discrete-state models.

Grid, dynamics, and payoffs

- State grid: $s \in 0, 1, 2, \ldots, 20$.
- Actions: $h \in 0, 1, \ldots, s$.
- Discount: $\beta = 0.95$.
- Logistic growth: $G(s) = s + rs(1 - s/K)$ with $r = 0.5$, $K = 20$. After computing G, round to the nearest state in $0, \ldots, 20$.
- Payoff: $\pi(s,h) = p\,h - c\,h$, with $p = 1$, $c = 0$ (net price = 1).

For each s,

$$(\mathcal{T}V)(s) = \max_{0 \le h \le s} \left\{ h + \beta V\left(\text{round}(G(s-h))\right) \right\}.$$

Start with $V^{(0)} \equiv 0$ and iterate $V^{(k+1)} = \mathcal{T}V^{(k)}$ until $|V^{(k+1)} - V^{(k)}|_\infty < 10^{-8}$. The greedy action at convergence is the optimal policy $h^*(s)$.

See R script `fishery.R` for implementation details.

3 Functional Equations and Value Function Properties

Dynamic programming problems often lead to **functional equations**—equations in which the *unknown* is a function rather than a scalar. The **Bellman equation** is the most important example in economics. Instead of solving for a scalar like x, we are solving for an entire function $V(\cdot)$ that satisfies a relationship involving itself.

3.1 What Is a Functional Equation?

- A **functional equation** relates the value of a function at one point to its value at another point.
- In its simplest form, it can look like:

$$V(s) = F(s, V(g(s)))$$

- where the function V appears on both sides.

- We are not trying to find a single number, but rather a function $V(\cdot)$ that makes this equation true for every s.
- Functional equations appear whenever we describe **recursive behavior** — situations where "the value of something today depends on the value of something tomorrow."

Think of **compound interest**:

$$A_{t+1} = (1 + r)A_t$$

The future value depends on today's value through a simple rule.

A functional equation specifies a relationship between a function and its transformed version.

In dynamic optimization:

$$V(s) = \max_{x \in X(s)} \{\pi(s, x) + \beta V(f(s, x))\}$$

where:

- $V(s)$ is the **value function**, giving the maximum attainable value from state s.
- $\pi(s, x)$ is the **current payoff** (or profit, utility).
- $f(s, x)$ gives the **next-period state**.
- $\beta \in (0, 1)$ is the **discount factor**.

The equation says: *The value of being in state s today equals the current payoff plus the discounted value of the next state, assuming optimal choice x.*

3.1.1 Functional Equations in Economics

Example: A Simple Savings Problem

Suppose an agent chooses consumption c_t and next-period assets a_{t+1}.

$$\max_{\{c_t, a_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u(c_t)$$
$$\text{s.t. } c_t + a_{t+1} = (1 + r)a_t + y_t, \quad a_t \geq 0$$

where:

- $u(c)$ is the utility function (e.g., $u(c) = \log(c)$),
- r is the interest rate,
- y_t is income,
- $\beta \in (0, 1)$ is the discount factor.

Then the Bellman equation is

$$V(a_t) = \max_{a_{t+1} \geq 0} \{u((1+r)a_t + y - a_{t+1}) + \beta V(a_{t+1})\}.$$

FOC (interior) with respect to a_{t+1}

Let $c_t = (1+r)a_t + y - a_{t+1}$. Then

$$\frac{\partial}{\partial a_{t+1}} \Big[u((1+r)a_t + y - a_{t+1}) + \beta V(a_{t+1})\Big] = -u'(c_t) + \beta V'(a_{t+1}) = 0,$$

The agent trades off **current utility** $u(c)$ versus the **future value** $\beta V(a')$.

Firm Investment Problem

A firm with capital stock k_t chooses investment i_t:

$$\max_{i_t} \sum_{t=0}^{\infty} \beta^t \pi(k_t, i_t) \quad \text{s.t.} \quad k_{t+1} = (1-\delta)k_t + i_t, \quad k_t \geq 0.$$

The recursive form:

$$V(k_t) = \max_{i_t \geq 0} \{\pi(k_t, i_t) + \beta V((1-\delta)k_t + i_t)\}.$$

The function $V(k_t)$ gives the value of capital k_t as current profit plus discounted future (optimal) value of next-period capital. Again, the value function appears on both sides.

FOC with respect to i_t:

$$\frac{\partial}{\partial i_t} \Big[\pi(k_t, i_t) + \beta V((1-\delta)k_t + i_t)\Big] = \pi_i(k_t, i_t) + \beta V'((1-\delta)k_t + i_t) = 0.$$

Resource Extraction Problem

A resource owner decides how much to extract x_t from stock s_t:

$$V(s_t) = \max_{0 \leq x_t \leq s_t} \{px_t - c(x_t) + \beta V(s_t - x_t)\}.$$

Here:

- The **state** is remaining stock s_t.
- The **control** is extraction x_t.
- The **transition** is $s_{t+1} = s_t - x_t$.

The functional equation states that the value of the resource today equals profit from extraction plus the discounted value of what remains for tomorrow.

3.2 The Bellman Operator

The Bellman equation tells us that the value of being in a given state today, $V(s)$, equals the best possible current payoff plus the discounted value of what happens next.

We can think of this process as an **operator** — a kind of "machine" that takes a guess about the value function and produces a new, updated guess.

3.2.1 How it works

Start with any function $V(s)$ that tells you what the value might be for each state.
Then define a new function:

$$(\mathcal{T}V)(s) = \max_{x \in X(s)} \{\pi(s,x) + \beta V(f(s,x))\}.$$

What this says in words:

> "Given my current guess about how valuable future states are (that's V), what would be the total value of making the best decision today?"

So $\mathcal{T}$ takes the *old* value function and gives you a *new* one that's a little closer to the truth.
It's a way to **think one step ahead**.

3.2.2 The Fixed Point Idea

The true value function, $V^*(s)$, is the one that **doesn't change** when we apply this operation again:

$$V^*(s) = (\mathcal{T}V^*)(s).$$

In other words, if you already know the correct V^*, thinking one step ahead doesn't change your beliefs because you are already correct about the future.

Why this matters

- The operator gives us a **recipe for computing** V^*:
  - start with a guess and keep applying $\mathcal{T}$ repeatedly.
  - Each time, you're improving your estimate of the value of being in each state.
- Economically, this process mirrors **learning or planning**:
  - we evaluate today's decisions using our expectations of tomorrow, adjust, and repeat until everything is internally consistent.

Visual Intuition

- If you plotted $V_0(s)$ (your first guess) and then $V_1(s) = \mathcal{T}V_0(s)$, the curves would move closer and closer together until they line up at $V^*(s)$.
- That's what it means for the Bellman equation to be a **fixed point** — a steady state in your expectations about value.

In the context of economics

- For a **consumer**, $\mathcal{T}$ means re-evaluating how much future consumption is worth.
- For a **firm**, it means updating the expected profitability of holding or investing capital.
- For a **resource owner**, it means revising how valuable it is to leave part of the stock for tomorrow.

In all cases, the Bellman operator captures the logic of **forward-looking behavior**:
today's value depends on how optimally we plan for tomorrow.

Each application of $\mathcal{T}$ corresponds to "thinking one step further ahead."

- Starting with any initial guess $V_0(s)$,
- Repeatedly applying $\mathcal{T}$, $V_{k+1} = \mathcal{T}V_k$,
- Converges to the true value function V^* under mild conditions.

This is **Value Function Iteration (VFI)**.

3.3 Existence and Uniqueness: Why the Bellman Equation Has a Single Solution

Once we define the Bellman operator $\mathcal{T}$ - the rule that takes a guess about future value and updates it - we can ask two key questions:

1. Does this process always lead to a stable value function?
2. Will it always settle on the **same** function, no matter where we start?

The answer is yes — as long as future payoffs are **discounted** (so $\beta < 1$).

The reason is the **contraction mapping property**.

A **contraction mapping** is a transformation that *pulls things closer together* every time you apply it.

$$||\mathcal{T}V_1 - \mathcal{T}V_2|| \leq \gamma \, ||V_1 - V_2||$$

Imagine taking two different guesses about the value function, say $V_1(s)$ and $V_2(s)$. When we apply the Bellman operator to both, the resulting functions $\mathcal{T}V_1$ and $\mathcal{T}V_2$ are **closer to each other** than the originals.

Each round of updating reduces the distance between our guesses — eventually, all sequences converge to the same point, the **true value function** V^*.

Discounting is what makes this work. Because future rewards are multiplied by $\beta < 1$, any disagreement about the future is automatically **shrunk** when we think one step ahead.

Small differences in how we value the future can't explode backward into large differences today — they fade over time.

This gives the Bellman operator its *gravitational pull* toward a single stable value function.

If applying $\mathcal{T}$ repeatedly always pulls guesses closer together, there must be one and only one function that can't be improved upon — that's the **fixed point**:

$$V^*(s) = \mathcal{T}V^*(s).$$

Mathematically, this follows from the **Contraction Mapping Theorem**, but economically, you can think of it as saying:

There is one internally consistent way to value the future that agrees with itself when we plan forward.

3.3.1 How this helps us in practice

Because $\mathcal{T}$ is a contraction, we can find V^* by **value function iteration**:

1. Start with any initial guess $V_0(s)$ — even something crude.
2. Apply $\mathcal{T}$ to get an updated guess $V_1 = \mathcal{T}V_0$.
3. Keep repeating: $V_{n+1} = \mathcal{T}V_n$.

Each iteration gets us closer to the truth.
No calculus tricks, no global search — just forward iteration guided by economic logic.

3.3.2 The big takeaway

- Discounting and diminishing returns make the future "well-behaved."

- Together they guarantee that the Bellman equation has **one** solution, and that repeated forward-looking reasoning will find it.
- This is why we can compute dynamic equilibria with confidence: as long as the problem is discounted and well-behaved, there is a single, stable value function waiting to be found.

3.4 The Envelope Theorem in Dynamic Programming

Consider a general Bellman equation:

$$V(s) = \max_{x \in X(s)} \{\pi(s,x) + \beta V(f(s,x))\}.$$

Let $x^*(s)$ be the optimal policy function.
We often need the derivative $V'(s)$ to derive first-order conditions or Euler equations.

3.4.1 The Problem

- The right-hand side depends on s **both directly** (through π and f) and **indirectly** through the optimal choice $x^*(s)$.
- If we differentiate $V(s)$ directly, we would in principle need to differentiate through $x^*(s)$, which is messy.
- The envelope theorem tells us we can ignore that part.

3.4.2 The Envelope Condition

At the optimum $x^*(s)$,

$$V'(s) = \left.\frac{\partial \pi(s,x)}{\partial s}\right|_{x=x^*(s)} + \beta\, V'(f(s,x^*(s)))\, \left.\frac{\partial f(s,x)}{\partial s}\right|_{x=x^*(s)}.$$

Intuition:

- Because the first-order condition for x^* holds, the indirect effect of changing s through a change in $x^*(s)$ is *zero*.
- We only need to consider the **direct** effects of s on the current payoff and next-period state.

3.4.3 Dynamic Envelope Condition (Benveniste–Scheinkman Theorem)

In discrete time, the theorem states:

- If the Bellman equation is differentiable and the policy function $x^*(s)$ is interior and continuous, then

$$V'(s) = \frac{\partial}{\partial s}\Big[\pi(s,x^*(s)) + \beta V(f(s,x^*(s)))\Big].$$

- You can interpret $V'(s)$ as the **shadow value of the state variable** - the marginal increase in lifetime value from a marginal increase in the current state.

3.4.4 Example: Consumption–Savings Problem

Bellman equation:

$$V(a_t) = \max_{a_{t+1}} \{u((1+r)a_t + y - a_{t+1}) + \beta V(a_{t+1})\}.$$

FOC w.r.t. a_{t+1}:

$$-u'(c_t) + \beta V'(a_{t+1}) = 0 \quad \rightarrow \quad u'(c_t) = \beta V'(a_{t+1}).$$

Envelope condition:

$$V'(a_t) = u'(c_t) \cdot (1+r).$$

Differentiate the Bellman RHS w.r.t. a_t holding the optimizer fixed:

$$V'(a_t) = u'(c_t) \cdot (1+r).$$

Euler equation

Apply the envelope condition at $t+1$:

$$V'(a_{t+1}) = (1+r)u'(c_{t+1}).$$

Substitute into the FOC:

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

Interpretation:

- An extra dollar of wealth today raises lifetime value by $(1+r)u'(c_t)$ - it can either be consumed now (worth $u'(c_t)$) or saved to yield $(1+r)u'(c_{t+1})$ next period. The household is indifferent between consuming today and saving for tomorrow when the Euler equation holds.

3.4.5 Why This Matters

- The envelope theorem allows us to link **marginal value** ($V'(s)$) to **marginal utility** or **marginal profit**, depending on context.
- It simplifies derivations of **Euler equations**, **costate equations**, and **first-order necessary conditions**.
- It provides a recursive way to compute derivatives of value functions in both analytical and numerical work.