Dynamic Programming Part B

1 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.

1.1 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."

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.*

1.1.1 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:

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

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.

1.2 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.

1.2.1 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:

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

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**.

1.2.2 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)**.

1.3 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.

1.3.1 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.

1.3.2 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.

1.4 Euler Equations and the Envelope Condition

Dynamic optimization gives two complementary characterizations of optimal behavior:

- **Euler equation (FOC in the control):** how the agent trades off current vs. future returns when choosing x.
- **Envelope condition (FOC "in the state"):** how the lifetime value V changes with the state s.

The envelope condition is what lets us **eliminate messy derivatives of the policy function** and express the Euler equation in terms of primitives and V' only.

We work with the same notation as above:

$$V(s) = \max_{x \in X(s)} \{\pi(s,x) + \beta V(f(s,x))\}, \quad 0 < \beta < 1,$$

with optimal policy $x^*(s)$.

1.4.1 From Bellman to First-Order Conditions

Assume an interior, differentiable solution for intuition (we add bounds/KKT below).

- **Stationarity (optimum in x):**

$$\pi_x(s, x^*(s)) + \beta\, V'(f(s, x^*(s)))\, f_x(s, x^*(s)) = 0.$$

- **Envelope (optimum in s):**

$$V'(s) = \pi_s(s, x^*(s)) + \beta\, V'(f(s, x^*(s)))\, f_s(s, x^*(s)).$$

Why envelope works: when differentiating the max wrt s, the chain-rule term involving $x_s^*(s)$ vanishes because the stationarity condition sets the derivative wrt x to zero at the optimum. Intuitively, a marginal change in s doesn't induce a first-order change through the (already optimized) $x^*(s)$.

1.4.2 The (One-Step-Ahead) Euler Equation

Write the time-t Bellman equation at (s_t, x_t) with $s_{t+1} = f(s_t, x_t)$:

$$\pi_x(s_t, x_t) + \beta\, V'(s_{t+1})\, f_x(s_t, x_t) = 0.$$

This is the **Euler equation**: the current marginal payoff from x_t equals the discounted marginal value of how x_t moves the state into the future.

We can **eliminate** $V'(s_{t+1})$ using the envelope condition at $t+1$:

$$V'(s_{t+1}) = \pi_s(s_{t+1}, x_{t+1}) + \beta\, V'(s_{t+2})\, f_s(s_{t+1}, x_{t+1}),$$

which yields a purely **primitive** intertemporal tradeoff once substituted back. In many applications π does not depend directly on s (only through feasibility), making this especially clean.

1.4.2.1 Consumption–Savings (simple, separable)

Let's illustrate with the **savings–consumption problem**.

The household chooses next period's assets a_{t+1} each period:

$$V(a_t) = \max_{a_{t+1}\geq 0} \Big\{u(c_t) + \beta V(a_{t+1})\Big\},$$

subject to the budget constraint

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

Substitute $c_t = (1+r)a_t + y - a_{t+1}$, so

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

The First-Order Condition (Euler Equation)

Take the derivative of the Bellman RHS with respect to a_{t+1}:

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

which gives

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

This is the **Euler condition** in implicit form: the marginal utility today equals the discounted shadow value of next period's assets.

The Envelope Condition

Differentiate the Bellman equation with respect to a_t, treating a_{t+1} as constant (by the envelope theorem):

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

Intuitively: a marginal increase in assets a_t raises consumption by $(1+r)$, increasing utility by $u'(c_t)(1+r)$.

Why We Can "Treat a_{t+1} as Constant"

$$\frac{dV(a_t)}{da_t} = u'(c_t)\big[(1+r) - \frac{da^*_{t+1}(a_t)}{da_t}\big] + \beta V'(a_{t+1})\frac{da^*_{t+1}(a_t)}{da_t}.$$

Now, note that the **first-order condition** for optimal a_{t+1} is:

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

Rearrange and substitute this into the derivative:

$$\frac{dV(a_t)}{da_t} = u'(c_t)(1+r) - \big[u'(c_t) - \beta V'(a_{t+1})\big]\frac{da^*_{t+1}(a_t)}{da_t}.$$

The term in brackets equals zero by the FOC, so the entire last term drops out:

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

We can use the envelope condition one period ahead

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

Then substitute it into the original FOC wrt a_{t+1}

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

This is the **standard consumption Euler equation**, expressing intertemporal optimality purely in terms of marginal utilities and the return $1+r$.

Economic interpretation

The Euler equation equates the *marginal benefit* and *marginal cost* of saving.

- The left-hand side, $u'(c_t)$, is the **marginal utility cost** of giving up one unit of consumption today.
- The right-hand side, $\beta(1+r)u'(c_{t+1})$, is the **discounted marginal utility benefit** of the extra consumption made possible tomorrow by saving that unit and earning the return $(1+r)$.

In equilibrium, these two must be equal — meaning that the household is indifferent (at the margin) between consuming one more unit today or saving it to consume tomorrow.

Interpreting the ratio of marginal utilities

Rearranging gives

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

- The **left-hand side** is the *intertemporal marginal rate of substitution* (IMRS) — the rate at which the household is willing to trade future consumption for current consumption.
- The **right-hand side** is the *intertemporal price ratio* implied by markets — the relative price of tomorrow's consumption in terms of today's, discounted by β and adjusted for the gross interest rate $(1+r)$.

In equilibrium, these two ratios are equal: the household's willingness to substitute consumption across periods equals the market's rate of return on savings.

Intuition

- If the interest rate r rises, the right-hand side decreases.
  To restore equality, the ratio $u'(c_{t+1})/u'(c_t)$ must fall — meaning c_{t+1}/c_t rises.
  $\rightarrow$ **Higher interest rates encourage saving and future consumption.**
- If β falls (the agent becomes more impatient), the right-hand side decreases.
  $\rightarrow$ **The household consumes more today and less tomorrow.**

This condition captures the **core intertemporal tradeoff** at the heart of dynamic economic behavior: balancing impatience, returns, and the curvature of utility (risk aversion or diminishing marginal utility).

Let's assume constant relative risk aversion

$$u(c) = \frac{c^{1-\sigma}}{1-\sigma}, \quad \sigma > 0,$$

where σ is the **coefficient of relative risk aversion** (and $1/\sigma$ is the **intertemporal elasticity of substitution**).

Then

$$u'(c) = c^{-\sigma}.$$

Plug this into the Euler equation:

$$u'(c_t) = \beta(1+r)u'(c_{t+1}) \quad \Rightarrow \quad c_t^{-\sigma} = \beta(1+r)c_{t+1}^{-\sigma}.$$

Rearrange for the **growth rate of consumption**:

$$\frac{c_{t+1}}{c_t} = \big[\beta(1+r)\big]^{1/\sigma}.$$

Interpretation

- If $\beta(1+r) = 1$, then $c_{t+1} = c_t$: consumption is *constant* over time.
- If $\beta(1+r) > 1$, the household values future consumption relatively more $\rightarrow$ **consumption grows** over time.
- If $\beta(1+r) < 1$, the household is relatively impatient $\rightarrow$ **consumption declines** over time.

Economic parameters: - β captures *patience*: higher $\beta \rightarrow$ slower consumption decline (more saving). - σ governs *willingness to smooth consumption*: higher σ (more curvature) $\rightarrow$ less sensitivity of growth to interest rates.

This simple form makes the Euler condition directly testable and provides a foundation for empirical work in both **macroeconomics** and **household finance**.

1.4.2.2 Renewable Resource (fishery; matches your notes)

Bellman: $V(s) = \max_{0\le h\le s}\{p\,h - c\,h + \beta V(G(s-h))\}$ with $s' = G(s-h)$.

- **Euler (interior):**
$$(p-c) - \beta\,V'(s')\,G'(s-h) = 0.$$
- **Envelope:**
$$V'(s) = \beta\,V'(s')\,G'(s-h).$$

Combine them to eliminate $V'(s')$:

$$p - c = V'(s).$$

Interpretation: harvest until marginal net revenue equals the **shadow value** of the stock (the user cost). With bounds, you get the usual "escapement" logic: at low s, $h^* = 0$ (rebuilding); at high s, h^* hits the upper bound $h = s$ (if profitable).

1.5 Analytical Example: Hotelling Resource Extraction

We now work through a fully analytical example that ties together the **Euler equation** and the **Envelope (shadow value)** condition.

1.5.1 Setup

A social planner (or resource owner) chooses extraction $\{q_t\}$ to maximize the discounted value of profits from a nonrenewable resource.

$$\max_{\{q_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t [p\,q_t - c(q_t)]$$

subject to the **resource stock constraint**

$$S_{t+1} = S_t - q_t, \qquad S_t \geq 0,$$

with given S_0 and discount factor $0 < \beta < 1$.

1.5.2 Bellman Equation

Let $V(S)$ be the value of having S units of the resource left:

$$V(S) = \max_{0 \leq q \leq S} \left\{ p\, q - c(q) + \beta\, V(S - q) \right\}.$$

1.5.3 First-Order and Envelope Conditions

1.5.3.1 (a) Euler (stationarity in q)

$$p - c'(q_t) = \beta\, V'(S_{t+1}).$$

The planner extracts up to the point where the **current net price** equals the **discounted shadow value of remaining stock**.

1.5.3.2 (b) Envelope (marginal value of the stock)

Differentiating the Bellman equation with respect to S and using the envelope theorem:

$$V'(S_t) = \beta\, V'(S_{t+1}).$$

1.5.4 Combine Euler and Envelope

Shift the envelope one period forward:

$$V'(S_{t+1}) = \beta\, V'(S_{t+2}).$$

Substitute into the Euler:

$$p - c'(q_t) = \beta\, V'(S_{t+1}) = \beta^2 V'(S_{t+2}) = \cdots$$

so $V'(S_t)$ grows at the rate $1/\beta$.

Define $\lambda_t \equiv V'(S_t)$ as the **shadow price** or **resource rent**. Then:

$$\lambda_{t+1} = \frac{\lambda_t}{\beta}.$$

If $\beta = (1 + r)^{-1}$, this gives the classic **Hotelling rule**:

$$\lambda_{t+1} = (1 + r)\lambda_t.$$

That is, the *resource rent* (or in equilibrium, the **net price**) must grow at the rate of interest.

1.5.5 Linear-Cost Example

Let $c(q) = \frac{1}{2}\gamma q^2$ with $\gamma > 0$.
Then the FOC is

$$p - \gamma q_t = \beta V'(S_{t+1}).$$

Using the envelope $V'(S_t) = \beta V'(S_{t+1})$, we can eliminate V':

$$p - \gamma q_t = V'(S_t).$$

Differentiate over time to get the *Euler equation in observable variables*:

$$(p - \gamma q_{t+1}) = \frac{1}{\beta}(p - \gamma q_t).$$

If $\beta = (1+r)^{-1}$,

$$p - \gamma q_{t+1} = (1+r)(p - \gamma q_t),$$

so the **net price (marginal rent)** rises at the interest rate.

1.5.6 6. Economic Interpretation

Condition	Meaning	Intuition
$p - c'(q_t) = \beta V'(S_{t+1})$	*Euler:* marginal profit = discounted shadow value	Extract until current profit = opportunity cost of future scarcity
$V'(S_t) = \beta V'(S_{t+1})$	*Envelope:* shadow price grows at $1/\beta$	Keeping one more unit today increases lifetime value by the discounted rent
$\lambda_{t+1} = (1+r)\lambda_t$	*Hotelling rule*	In equilibrium, the resource rent (or net price) grows at the interest rate

---

1.5.7 Policy Intuition

- When the interest rate r is **high**, future rents are heavily discounted $\rightarrow$ faster extraction.
- When r is **low** or β high, future rents matter more $\rightarrow$ slower extraction.
- The rule does *not* depend on the level of stock S_t; it is purely an **arbitrage condition** between leaving the resource in the ground vs. extracting and investing the proceeds.

Note

Why Must Rents Grow at the Interest Rate?
The Hotelling rule says that in equilibrium,

$$\lambda_{t+1} = (1+r)\lambda_t,$$

or equivalently that the **net price (resource rent)** grows at the rate of interest.
Let's unpack *why* this must be true.

The Resource as an Asset
Think of the remaining stock S_t as an **asset**: each unit in the ground is a claim on a future profit when extracted.

- If you **extract one more unit now**, you earn today's net revenue
  current rent $= p - c'(q_t) = \lambda_t$.
- If you **leave it in the ground**, you keep the option to sell it in the future, when scarcity will be higher and rent will be larger.

Thus, each unit of resource is an asset that yields **capital gains** (through rising rent) but **no dividends** (unless extracted).
No-Arbitrage Condition
An owner can hold wealth in two forms:

1. **Financial asset** yielding interest rate r.

2. **Resource asset** whose value (the rent λ_t) appreciates over time.

If both are riskless, **arbitrage** implies they must yield the same return.
Otherwise:

- If $\lambda_{t+1}/\lambda_t > 1 + r$:
  holding the resource yields higher return $\rightarrow$ everyone hoards $\rightarrow$ no extraction.
- If $\lambda_{t+1}/\lambda_t < 1 + r$:
  financial assets dominate $\rightarrow$ everyone extracts and invests proceeds.

Only when $\lambda_{t+1}/\lambda_t = 1 + r$ can both be held in equilibrium.
That is exactly the **Hotelling condition**:

$$\frac{\lambda_{t+1} - \lambda_t}{\lambda_t} = r.$$

2 Continuous Time and State

Like in many other contexts, continuous processes can simplify or allow the use of calculus. Instead of difference equations, we use differential equations to describe evolution of the state variable. The flows, π, are instantaneous. The continuous discounting is exponential, $e^{-\rho t}$.

$$V(s) = \max_x \int_0^\infty e^{-\rho t}\pi(s(t), x(t))dt$$
$$\text{s.t. } \dot{s}(t) = f(s(t), x(t))$$

and terminal condition $\lim_{t\to\infty} e^{-\rho t}V(s(t)) = 0$.

2.1 Current Value Hamiltonian

The current value Hamiltonian is the dynamic analog to the Lagrangian in static optimization. It combines current payoffs and the value of changing the state. We will drop the time arguments for clarity.

$$H(s, x, \lambda) = \pi(s, x) + \lambda f(s, x)$$

where $\lambda(t)$ is the co-state variable or shadow price associated with the state $s(t)$ in current value terms.

2.1.1 FOC (Pontryagin maximum principle)

The necessary first order conditions are

- Control equation: $\frac{\partial H}{\partial x} = 0$
- Co-state equation: $\dot{\lambda} = \rho\lambda - \frac{\partial H}{\partial s}$
- State equation: $\dot{s} = f(s, x)$
- Transversality condition: $\lim_{t\to\infty} e^{-\rho t}\lambda(t)s(t) = 0$

Control Condition

$$\frac{\partial H}{\partial x} = \frac{\partial \pi(s, x)}{\partial x} + \lambda\frac{\partial f(s, x)}{\partial x} = 0$$

This condition equates the marginal benefit and marginal cost of adjusting the control variable x.

- $\frac{\partial \pi}{\partial x}$ is the direct marginal profit from increasing x.
- $\lambda\frac{\partial f}{\partial x}$ is the indirect effect through how x changes the state variable — multiplied by the shadow value of the state.

So at the optimum the immediate gain from changing x must equal the value of the induced change in the future state.

Co-state equation (dynamic law of motion of the shadow price)

The condition $\dot{\lambda} = \rho\lambda - H_s$ says that shadow values evolve like an asset price - their rate of appreciation equals the interest rate minus the marginal effect of the state

$$\dot{\lambda} = \rho\lambda - \frac{\partial H}{\partial s} = \rho\lambda - \left(\frac{\partial \pi}{\partial s} + \lambda\frac{\partial f}{\partial s}\right)$$

This is a law of motion for the shadow price — it tells us how the value of the state evolves over time.

- The term $\rho\lambda$ is the “required return” on the shadow asset (the discounting effect).

- $\frac{\partial \pi}{\partial s}$ is the direct marginal effect of more s on current profit. For example, in a growth model, more capital s raises output today. If more capital already raises current profits, its shadow value doesn't need to grow as quickly.
- $\lambda \frac{\partial f}{\partial s}$ is the indirect effect through how s changes its own future growth. If a larger state today leads to faster growth tomorrow, larger future states are easier to read, so the current shadow value falls (lowering the rate of growth)

State equation

This describes the evolution of the physical or economic state.

$$\dot{s} = f(s, x)$$

The chosen control policy $x(t)$ determines how the system transitions through states — e.g. how capital accumulates, resources deplete, or pollution stock evolves. This is the link between short-run choices and long-run consequences.

Transversality condition

This condition ensures that the present value of the shadow price times the state variable vanishes at infinity:

$$\lim_{t \to \infty} e^{-\rho t} \lambda(t) s(t) = 0$$

This rules out "Ponzi schemes" where the shadow price grows so fast that its discounted value remains positive indefinitely. It ensures that the optimization problem is well-posed and that the agent cannot exploit infinite future value.

Summary

The FOCs of the current-value Hamiltonian are the continuous-time counterpart of the Euler equation and envelope condition in discrete-time dynamic programming:

- The control equation parallels the Euler condition: it characterizes optimal intertemporal allocation.
- The co-state equation mirrors the envelope theorem: it tells how the value function changes with the state.
- The transversality condition ensures well-behaved solutions.
- Together, they describe how current actions balance immediate payoffs and discounted future consequences.

i Where does the Co-state equation come from

1. We introduce a Lagrange multiplier $\mu(t)$ for the constraint $\dot{s} - f(s, x) = 0$.
The present-value Lagrangian is:

$$\mathcal{L} = \int_0^\infty e^{-\rho t} \left[\pi(s, x) + \mu(t)(\dot{s} - f(s, x))\right] dt$$

2. Integrate the term with $\dot{s}$ by parts: To get the first-order conditions, we want all terms multiplied by s (not $\dot{s}$).

$$\int_0^\infty e^{-\rho t} - \mu(t)\dot{s}(t) dt = \left[e^{-\rho t}\mu(t)s(t)\right]_0^\infty + \int_0^\infty e^{-\rho t} s(t) \left(\dot{\mu}(t) - \rho\mu(t)\right) dt$$

Assuming the boundary term vanishes at infinity, we have:

$$\mathcal{L} = \int_0^\infty e^{-\rho t} \left[\pi(s, x) - \mu(t) f(s, x) + s(t) \left(\dot{\mu}(t) - \rho\mu(t)\right)\right] dt$$

3. Define the current-value co-state variable:

$$\lambda(t) = \frac{\mu(t)}{e^{-\rho t}} = e^{\rho t}\mu(t)$$

   This rescales the multiplier to account for discounting.

Differentiating $\lambda(t)$ and rearranging gives:

$$\dot{\mu}(t) = e^{-\rho t}(\dot{\lambda}(t) - \rho\lambda(t))$$

When we plug this back into the FOCs, the necessary condition for $s(t)$ (the "state variable") gives:

$$\dot{\lambda}(t) = \rho\lambda(t) - \left(\frac{\partial \pi}{\partial s} + \lambda(t)\frac{\partial f}{\partial s}\right)$$

This is the **co-state equation** we wanted to derive

2.2 Connecting the Hamiltonian to the HJB Equation

The Hamilton-Jacobi-Bellman (HJB) equation is the continuous-time analog of the Bellman equation in discrete time. It characterizes the value function $V(s)$ as the solution to a differential equation.

The HJB equation states that at each instant in time, the value of being in state s equals the maximum of current payoffs plus the expected change in value from moving to a new state.

$$\rho V(s) = \max_{x}\{\pi(s,x) + V'(s)f(s,x)\}$$

2.2.1 Define the value function

$$V(s) = \max_{x}\int_0^{\infty} e^{-\rho t}\pi(s(t),x(t))dt$$

At any time t, define the remaining value of the program:

$$V(s(t)) = \max_{x(\cdot)}\int_t^{\infty} e^{-\rho(u-t)}\pi(s(u),x(u))du$$

Differentiating with respect to time t gives:

$$\frac{dV(s(t))}{dt} = \rho V(s(t)) - \pi(s(t),x(t)) - V'(s(t))\dot{s}(t)$$

i Leibniz Rule for Differentiation Under the Integral Sign

If you have an integral with variable limits:

$$I(t) = \int_{a(t)}^{b(t)} g(u,t)du$$

The derivative with respect to t is:

$$\frac{dI(t)}{dt} = g(b(t),t)b'(t) - g(a(t),t)\cdot a'(t) + \int_{a(t)}^{b(t)}\frac{\partial g(u,t)}{\partial t}du$$

In our case, the lower limit is t and the upper limit is ∞. The derivative becomes:

$$\frac{d}{dt}\int_t^\infty e^{-\rho(u-t)}\pi(s(u),x(u))du = -e^{-\rho(t-t)}\pi(s(t),x(t)) + \int_t^\infty \frac{\partial}{\partial t}\left(e^{-\rho(u-t)}\pi(s(u),x(u))\right)du$$

Note that $t-t=0$ so $e^{-\rho(t-t)}=1$ and

$$\frac{\partial}{\partial t}\left(e^{-\rho(u-t)}\pi(s(u),x(u))\right) = \rho e^{-\rho(u-t)}\pi(s(u),x(u)) = \rho V(s(t))$$

Note that $V(s(t))$ is the maximum value, so the derivative wrt to time must be zero at the optimum:

$$0 = \rho V(s(t)) - \pi(s(t),x(t)) - V'(s(t))\dot{s}(t)$$

Rearranging gives the HJB equation:

$$\rho V(s) = \pi(s,x) + V'(s)f(s,x)$$

Maximizing over x gives the full HJB:

$$\rho V(s) = \max_x\{\pi(s,x) + V'(s)f(s,x)\}$$

2.2.2 Link to the Hamiltonian

The value function is linked to the co-state variable by:

$$\lambda(s) = V'(s)$$

Substituting into the HJB gives:

$$\rho V(s) = \max_x\{\pi(s,x) + \lambda(s)f(s,x)\} = \max_x H(s,x,\lambda)$$

Thus, the Hamiltonian is the maximand in the HJB equation. The intuition is the same: λ captures the shadow value of the state, which evolves based on the effects of s on current and future payoffs. This is the same intuition as $V'(s)$.

2.3 Hotelling Resource Extraction in Continuous Time

Consider a resource owner extracting a nonrenewable resource over time to maximize discounted profits:

$$\max_{\{q(t)\}_{t=0}^\infty}\int_0^\infty e^{-\rho t}\left[pq(t) - c(q(t))\right]dt$$

subject to the resource stock constraint:

$$\dot{S}(t) = -q(t), \quad S(t)\geq 0, \quad S(0) = S_0$$

2.3.1 Current-Value Hamiltonian

The current-value Hamiltonian is:

$$H(S,q,\lambda) = pq - c(q) + \lambda(-q) = (p-\lambda)q - c(q)$$

2.3.2 First-Order Conditions

- **Control (optimal extraction):**

$$\frac{\partial H}{\partial q} = p - c'(q) - \lambda = 0 \quad \Rightarrow \quad p - c'(q) = \lambda$$

  This condition states that the marginal profit from extraction equals the shadow price of the resource.

- **Co-state (shadow price evolution):**

$$\dot{\lambda} = \rho\lambda - \frac{\partial H}{\partial S} = \rho\lambda - 0 = \rho\lambda$$

  Since H does not depend on S, the shadow price grows at the rate ρ.
  λ is the shadow value of the resource in situ, i.e. the resource rent.
  It must grow at the rate of interest ρ.
  Thus, resource rents rise at the interest rate — the Hotelling rule.
  If $\dot{\lambda} = \rho\lambda$, rents increase exponentially — extraction slows as the resource becomes scarcer.

2.4 Numeric Example

2.4.1 Hotelling extraction with quadratic costs (continuous time)

We consider the nonrenewable resource problem in continuous time with constant price and quadratic extraction costs:

- Objective: maximize discounted profits

$$\max_{\{x(t)\}_{t\geq 0}} \int_0^\infty e^{-\rho t}[p\, x(t) - C(x(t))]dt, \qquad C(x) = \tfrac{1}{2}cx^2$$

- State dynamics and initial stock

$$\dot{S}(t) = -x(t), \quad S(t) \geq 0, \quad S(0) = S_0$$

With price p constant and $C'(x) = cx$, the current-value Hamiltonian is

$$H(S, x, \lambda) = p\, x - \tfrac{1}{2}cx^2 + \lambda(-x) = (p - \lambda)x - \tfrac{1}{2}cx^2.$$

FOCs (Pontryagin) give

$$\begin{array}{ll} \text{Control:} & \dfrac{\partial H}{\partial x} = p - cx - \lambda = 0 \;\Rightarrow\; x(t) = \dfrac{p - \lambda(t)}{c} \;\;(\geq 0) \\ \text{Co-state:} & \dot{\lambda}(t) = \rho\, \lambda(t) \Rightarrow \lambda(t) = \lambda_0 e^{\rho t} \\ \text{State:} & \dot{S}(t) = -x(t) \end{array}$$

Because H does not depend on S, the rent (shadow price) grows at the interest rate: $\dot{\lambda}/\lambda = \rho$. The optimal extraction policy is therefore

$$\boxed{x(t) = \max\left\{\frac{p - \lambda_0 e^{\rho t}}{c}, 0\right\}}$$

Extraction continues until λ reaches p (see the numerator of the optimal policy); denote the stopping time T^*, defined by $\lambda(T^*) = p$. This yields

$$\boxed{T^* = \frac{1}{\rho}\log\left(\frac{p}{\lambda_0}\right)}$$

2.4.2 Numerical calibration (the script and figure)

Using the parameters $p = 1$, $\rho = 0.05$, $S_0 = 10$ and setting $c = 0.5$ for the quadratic cost slope (so $C(x) = \frac{1}{2}cx^2$ and $MC = cx$), the implied values are

$$\lambda_0 \approx 0.4488, \qquad T^* \approx 16.02\,\text{time units.}$$

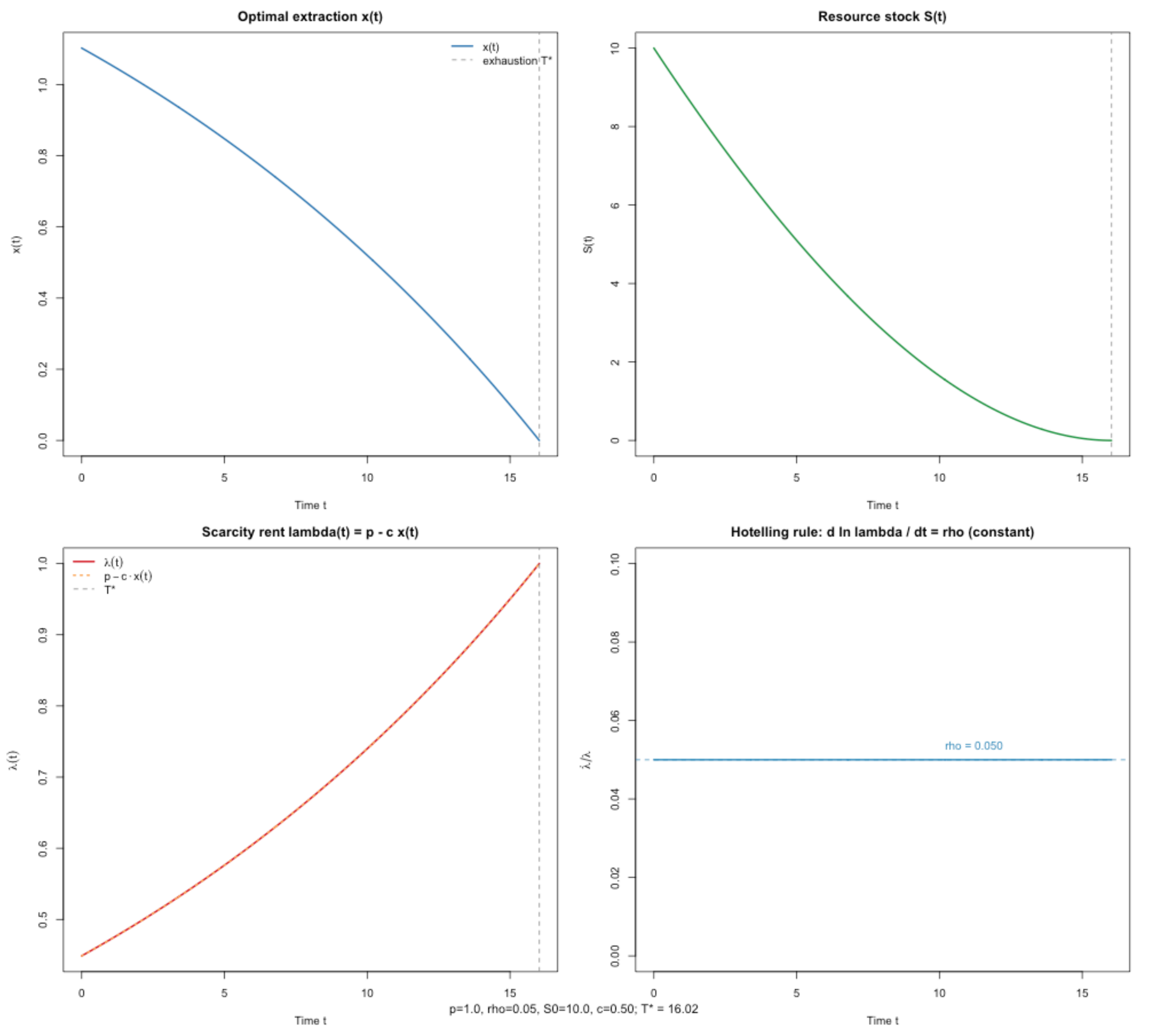
The trajectories are

$$\lambda(t) = \lambda_0 e^{\rho t}, \qquad x(t) = \frac{p - \lambda(t)}{c} \text{ for } t \le T^*, \qquad S(t) = S_0 - \int_0^t x(u)\,du.$$

The Hotelling rule is visible directly as a constant growth rate

$$\frac{\dot{\lambda}(t)}{\lambda(t)} = \rho.$$

Below is the numerically generated figure from `extraction_sim.R` (saved alongside this file) illustrating $x(t)$, $S(t)$, $\lambda(t)$, and the constant rent growth $\dot{\lambda}/\lambda = \rho$:



Key takeaways:

- The optimal policy extracts while $\lambda(t) < p$, with $x(t)$ declining to zero at T^* as rents rise exponentially.
- Rents λ grow at the interest rate ρ (Hotelling), so the net price equals the scarcity rent and appreciates like a riskless asset.
- Choosing λ_0 to satisfy the stock constraint ensures complete exhaustion at T^*.

3 Projection and Collocation Methods

In many economic models, especially dynamic ones, the optimality conditions or value functions are expressed as functional equations. Analytical solutions are rare, so we must approximate functions numerically.

When the state space is continuous and the model has no closed-form solution, the Bellman equation

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

defines infinitely many conditions — one for every possible value of s. Solving it exactly would mean finding a function $V(\cdot)$ that satisfies this equality everywhere, which is an infinite-dimensional problem.

Projection and collocation methods make this problem computationally tractable:

- We restrict attention to a family of approximate functions $\hat{V}(s;\theta)$ described by finitely many parameters θ.
- We then anchor the functional equation to hold exactly (collocation) or approximately (projection) at a finite set of representative states s_i.

A special and widely used case of projection methods is **collocation**, where the approximation is made to satisfy the equation exactly at selected nodes (the collocation points).

3.1 Collocation Method

1. **Choose Basis Functions:** Select a set of basis functions $\{\phi_j(s)\}_{j=1}^N$ to represent the value function. Common choices include polynomials, splines, or Chebyshev polynomials.

2. **Approximate the Value Function:** Represent the value function as a linear combination of the basis functions:

$$\hat{V}(s;\theta) = \sum_{j=1}^{N} \theta_j \phi_j(s)$$

   where $\theta = (\theta_1, \dots, \theta_N)$ are the coefficients to be determined.

3. Define the residual function that measures the deviation from the Bellman equation:

$$R(s;\theta) = \hat{V}(s;\theta) - \max_{x \in X(s)} \{\pi(s,x) + \beta \hat{V}(f(s,x);\theta)\}$$

   The goal of the collocation method is to make this residual as close to zero as possible.

4. **Select Collocation Points:** Choose a finite set of collocation points $\{s_i\}_{i=1}^M$ within the state space where the Bellman equation will be enforced exactly.

- For smooth problems, Chebyshev nodes are common.
- For economic models with kinks or bounds, uniform or adaptive grids are often used.

5. **Enforce the Bellman Equation:** At each collocation point, require the residual to vanish:

$$R(s_i;\theta) = 0, \quad i = 1, \dots, M,$$

   or equivalently,

$$\hat{V}(s_i;\theta) = \max_{x \in X(s_i)} \{\pi(s_i,x) + \beta \hat{V}(f(s_i,x);\theta)\}$$

   This yields a system of M equations in the N unknowns θ.

6. **Solve for Coefficients:** Solve the resulting system of equations for the coefficients θ using numerical methods (e.g., nonlinear solvers).

3.2 Example: Hotelling Resource Extraction via Collocation

A resource owner extracts x_t units from a finite stock s_t.

State dynamics:

$$s_{t+1} = s_t - x_t, \quad s_t \geq 0$$

Profit function:

$$\pi(s_t, x_t) = px_t - \frac{1}{2}cx_t^2$$

Bellman equation:

$$V(s_t) = \max_{0 \leq x_t \leq s_t} \left\{ px_t - \frac{1}{2}cx_t^2 + \beta V(s_t - x_t) \right\}$$

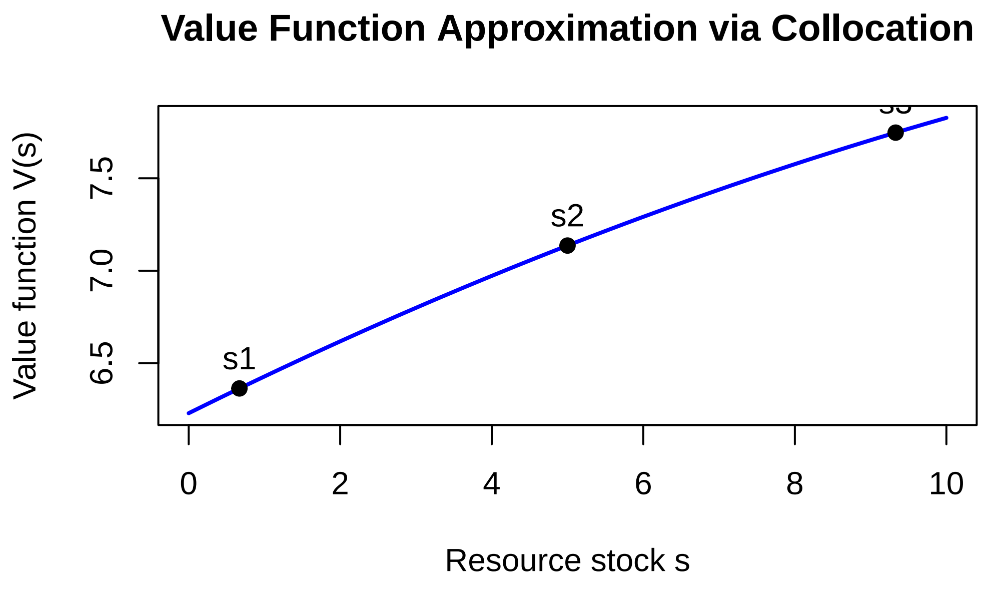
We'll approximate $V(s)$ using Chebyshev polynomials and enforce this equality at collocation nodes.

```
# Collocation for Hotelling extraction problem
library(pracma)   # for chebfun, chebpts
library(rootSolve) # for multiroot
```

```
Attaching package: 'rootSolve'

The following objects are masked from 'package:pracma':

    gradient, hessian
```

```
# Parameters
beta <- 0.95
Smax <- 10           # maximum stock
N <- 3               # number of basis functions
M <- N               # collocation points

# ---- Step 1: Chebyshev nodes on [0, Smax] ----
# Chebyshev nodes cluster near the endpoints of the interval.
# That's crucial for avoiding oscillations (Runge's phenomenon) when approximating functions with glo
# Generate N Chebyshev points on [-1, 1]
chebpts <- function(N) {
  k <- 0:(N-1)
  x <- cos(pi * (2*k + 1) / (2*N))
  return(rev(x))  # optional: reverse so they go from -1 to 1
}


s_nodes <- 0.5 * (Smax * (1 + chebpts(M)))  # map from [-1,1] to [0,Smax]

# ---- Step 2: Chebyshev basis function ----
cheb_basis <- function(s) {
  # Map s   [0, Smax] to z   [-1,1]
  z <- 2 * s / Smax - 1
  T0 <- rep(1, length(z))
  T1 <- z
  T2 <- 2*z^2 - 1
  T3 <- 4*z^3 - 3*z
  cbind(T0, T1, T2)[, 1:N]   # truncate to N basis functions
}
```

```
# ---- Step 3: Approximate value function ----
V_hat <- function(s, theta) {
  phi <- cheb_basis(s)
  as.numeric(phi %*% theta)
}

# ---- Step 4: Define residual function R(s;  ) ----
residuals_fun <- function(theta) {
  R <- numeric(M)
  for (i in 1:M) {
    s <- s_nodes[i]
    # maximize over x in [0, s]
    objective <- function(x) {
      profit <- x - 0.5 * x^2 + beta * V_hat(s - x, theta)
      return(-profit)  # negative for minimization
    }
    opt <- optimize(objective, c(0, s))
    R[i] <- V_hat(s, theta) - (-opt$objective)
  }
  R
}

# ---- Step 5: Solve for coefficients  ----
theta_init <- rep(0, N)
sol <- multiroot(f = residuals_fun, start = theta_init)
theta_star <- sol$root

# ---- Step 6: Evaluate results ----
s_grid <- seq(0, Smax, length.out = 100)
V_approx <- V_hat(s_grid, theta_star)

plot(s_grid, V_approx, type = "l", lwd = 2, col = "blue",
     xlab = "Resource stock s", ylab = "Value function V(s)",
     main = "Value Function Approximation via Collocation")
points(s_nodes, V_hat(s_nodes, theta_star), pch = 19)
text(s_nodes, V_hat(s_nodes, theta_star), labels = paste0("s", 1:M), pos = 3)
```



3.3 Chebyshev Polynomials and Their Role in Collocation

1. Motivation

When we approximate a value function $\hat{V}(s)$ with global polynomials, we want an approximation that is:

- Accurate across the entire domain, not just locally;
- Stable, avoiding oscillations near the boundaries; and
- Numerically well-behaved when we increase the number of basis functions.

Standard power polynomials $(1, s, s^2, s^3, ...)$ often perform poorly because they can oscillate wildly at the edges — a phenomenon known as Runge's phenomenon.

Chebyshev polynomials solve this problem. They are special orthogonal polynomials that minimize the maximum approximation error on $[-1, 1]$, providing near-optimal stability and accuracy for smooth functions.

2. Definition

The Chebyshev polynomials of the first kind, $T_n(z)$, are defined recursively:

$$T_0(z) = 1, \quad T_1(z) = z, \quad T_{n+1}(z) = 2zT_n(z) - T_{n-1}(z).$$

or equivalently,

$$T_n(z) = \cos(n \arccos(z)), \quad z \in [-1, 1].$$

Because they're expressed in terms of cosine, they oscillate evenly over $[-1, 1]$, with minimal distortion near the boundaries.

3. Orthogonality and Stability

When two functions are orthogonal under an inner product, it means they do not "overlap" in the space of functions.

- In Euclidean geometry, vectors are orthogonal if their dot product is zero.
- In function space, functions are orthogonal if their weighted integral of the product is zero.

So here:

- $T_0(z)$, $T_1(z)$, $T_2(z)$, ... form a set of "directions" in function space.
- Each T_n captures a distinct pattern of oscillation — like basis vectors pointing in different directions.

That means you can represent any smooth function $V(z)$ as a unique weighted combination of these basis shapes:

$$V(z) \approx \sum_{n=0}^{N-1} \theta_n T_n(z).$$

Each coefficient θ_n controls one independent component of the shape, just like coordinates along orthogonal axes in $\mathbb{R}^N$.

Orthogonality is what allows the approximation $\hat{V}(s;\theta)$ to "separate" the different sources of curvature or slope in the value function.

- The first basis term (T_0) captures the average level of the value function.
- T_1 captures the linear slope.
- T_2, T_3, ... capture higher-order curvature and oscillation.

Because of orthogonality, changes in one coefficient θ_i don't distort the others — just as changing one coordinate of a vector doesn't affect the others. This is why polynomial approximations using Chebyshev bases converge much faster and are numerically stable.

4. Chebyshev Nodes

When using Chebyshev polynomials for collocation, we select collocation points (nodes) at the roots of the highest-degree Chebyshev polynomial used.

your state variable — say the stock of a resource, capital, or wealth — usually lives on some finite interval $s \in [a, b]$.

But Chebyshev polynomials are defined only on the standard interval: $z \in [-1, 1]$.

To use Chebyshev bases to approximate functions of s, we need to rescale or map between the two intervals.

That's where the $s \leftrightarrow z$ transformation comes in.

$$z_k = \cos\left(\frac{(2k-1)\pi}{2N}\right), \quad k = 1, 2, \dots, N.$$

To go back from z to s, we just invert that linear transformation:

$$s = \frac{b-a}{2} z + \frac{a+b}{2}.$$

That means you can move back and forth between:

- The physical state s (used in your model, e.g. the stock of a resource)
- The Chebyshev coordinate z (used in numerical approximation).

5. Approximating a Function

We represent the value function using a truncated Chebyshev series:

$$\hat{V}(z;\theta) = \sum_{n=0}^{N-1} \theta_n T_n(z).$$

This means we are expressing $V(s)$ as a weighted combination of orthogonal basis shapes that oscillate at increasing frequency as j grows.

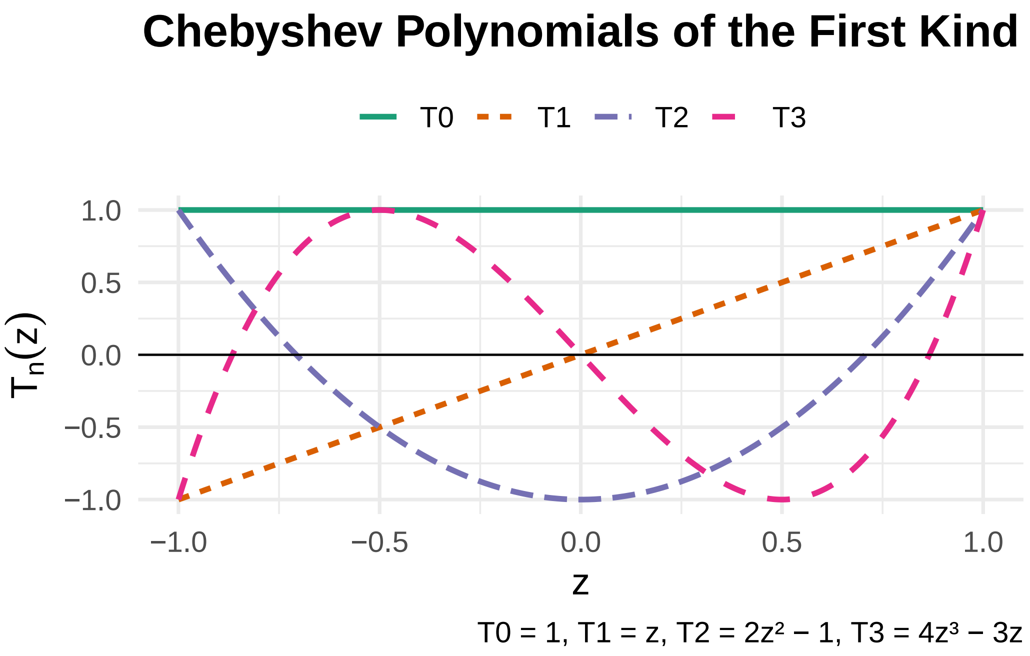
- $T_0(z)$ gives the average level (constant term).
- $T_1(z)$ gives the slope (linear component).
- Higher-order terms add curvature and oscillations.

6. Why Economists Like Chebyshev Polynomials

They offer:

- Global accuracy: A small number of terms often gives a close approximation to smooth value functions.
- Good behavior near boundaries: Avoids oscillations in regions with low curvature.
- Simple derivatives: Derivatives can be computed recursively, which is useful for Euler or HJB equations.
- Efficient computation: Fast polynomial evaluation

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```



3.4 From value approximation to policy and simulation

Once we have an approximate value function $\hat{V}(s;\theta)$ from collocation, we can recover the optimal extraction policy $x^*(s)$ either by:

- direct one-dimensional maximization of the Bellman RHS for each s, or
- solving the Euler/FOC using the derivative $\hat{V}'(\cdot)$ from the Chebyshev approximation:

$$p - c\,x^*(s) - \beta\,\hat{V}'(s - x^*(s);\theta) = 0, \quad x^*(s) \in [0, s].$$

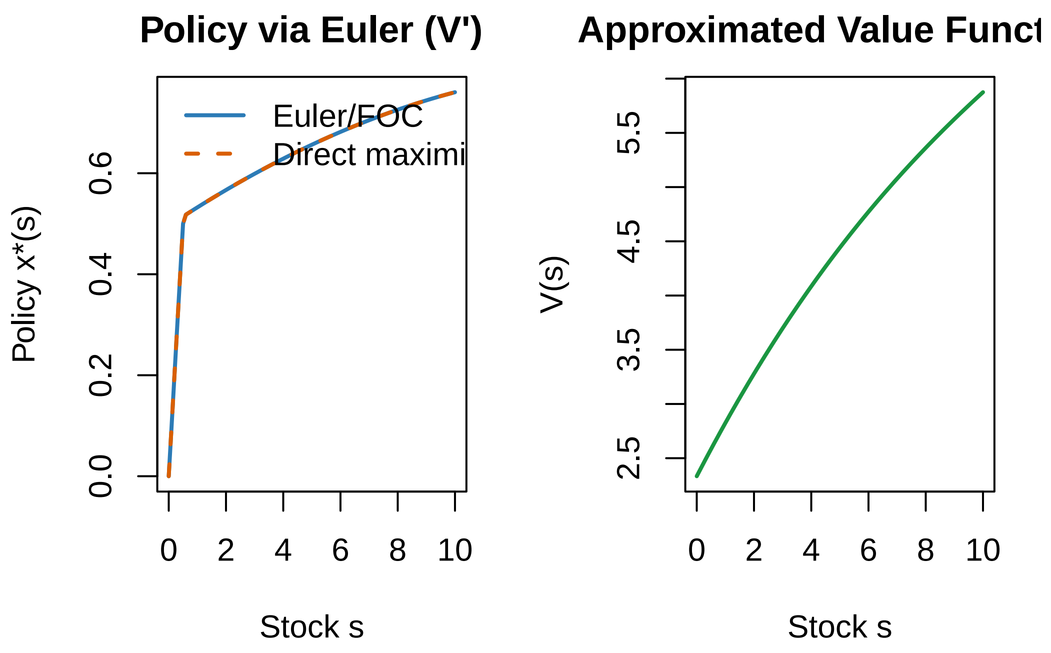
Below we implement both, leveraging that Chebyshev bases provide a closed-form derivative via the chain rule from the affine map $z = 2s/S_{\max} - 1$.

```
# Recover policy from Chebyshev-approximated value function
library(rootSolve)

# ---- Parameters (discrete time Hotelling) ----
beta  <- 0.95
```

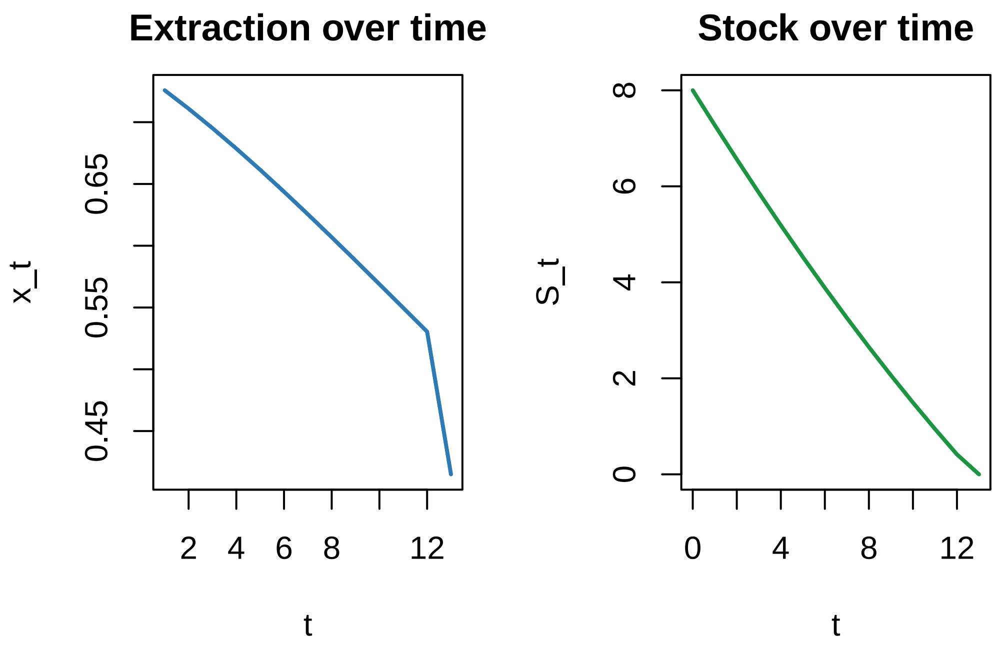
```r
p     <- 1.0
c_mc  <- 1.0          # marginal cost slope so C(x) = 1/2 c_mc x^2
Smax  <- 10           # domain for approximation of V(s)
N     <- 4            # number of Chebyshev basis terms
M     <- N            # collocation points

# ---- Chebyshev nodes on [0, Smax] ----
chebpts <- function(N) {
  k <- 0:(N-1)
  x <- cos(pi * (2*k + 1) / (2*N))
  rev(x) # from -1 to 1
}
s_nodes <- 0.5 * (Smax * (1 + chebpts(M)))

# ---- Chebyshev basis and its derivative wrt s ----
cheb_basis <- function(s, N = 4, Smax = 10) {
  z <- 2 * s / Smax - 1
  # Recurrence for T_n(z)
  T <- matrix(NA, nrow = length(z), ncol = N)
  T[,1] <- 1
  if (N > 1) T[,2] <- z
  if (N > 2) {
    for (j in 3:N) T[,j] <- 2 * z * T[,j-1] - T[,j-2]
  }
  colnames(T) <- paste0("T", 0:(N-1))
  as.matrix(T)
}

cheb_basis_deriv <- function(s, N = 4, Smax = 10) {
  z <- 2 * s / Smax - 1
  dzds <- 2 / Smax
  # Compute T_n(z) and dT_n/dz via stable recurrence
  # Initialize
  T <- matrix(0, nrow = length(z), ncol = N)
  dTdz <- matrix(0, nrow = length(z), ncol = N)
  T[,1] <- 1
  if (N > 1) {
    T[,2] <- z
    dTdz[,2] <- 1
  }
  if (N > 2) {
    for (n in 3:N) {
      # T_n = 2 z T_{n-1} - T_{n-2}
      T[,n] <- 2 * z * T[,n-1] - T[,n-2]
      # dT_n/dz = 2 T_{n-1} + 2 z dT_{n-1}/dz - dT_{n-2}/dz
      dTdz[,n] <- 2 * T[,n-1] + 2 * z * dTdz[,n-1] - dTdz[,n-2]
    }
  }
  # dT_n/ds = dT_n/dz * dz/ds -- chain rule
  dTds <- dTdz * dzds
  colnames(dTds) <- paste0("dT", 0:(N-1), "ds")
  as.matrix(dTds)
}
```

```r
V_hat <- function(s, theta) {
  # Uses global N and Smax from the chunk scope
  Phi <- cheb_basis(s, N, Smax)
  as.numeric(Phi %*% theta)
}

V_hat_prime <- function(s, theta) {
  # Uses global N and Smax from the chunk scope
  Phi_d <- cheb_basis_deriv(s, N, Smax)
  as.numeric(Phi_d %*% theta)
}

# ---- Collocation residuals with parameters (p, c_mc, beta) ----
residuals_fun <- function(theta) {
  R <- numeric(M)
  for (i in 1:M) {
    s <- s_nodes[i]
    # Maximize Bellman RHS over x in [0, s]
    objective <- function(x) {
      prof <- p * x - 0.5 * c_mc * x^2 + beta * V_hat(s - x, theta)
      return(-prof) # minimize negative profit
    }
    opt <- optimize(objective, c(0, s))
    R[i] <- V_hat(s, theta) - (-opt$objective)
  }
  R
}

# ---- Solve for Chebyshev coefficients theta* ----
theta_init <- rep(0, N)
theta_star <- multiroot(f = residuals_fun, start = theta_init)$root

# ---- Policy from FOC (Euler) using V' ----
policy_x <- function(s) {
  if (s <= 0) return(0)
  f <- function(x) p - c_mc * x - beta * V_hat_prime(s - x, theta_star)
  # Try bracketing in [0, s]
  f0 <- f(0); fs <- f(s)
  if (is.finite(f0) && is.finite(fs) && f0 * fs < 0) {
    return(uniroot(f, c(0, s))$root)
  }
  # Fall back to direct 1D optimization on the Bellman RHS
  obj <- function(x) p * x - 0.5 * c_mc * x^2 + beta * V_hat(s - x, theta_star)
  vals <- c(obj(0), obj(s))
  if (is.finite(vals[1]) && is.finite(vals[2])) {
    return(c(0, s)[which.max(vals)])
  }
  # Robust final fallback
  optimize(function(x) -obj(x), c(0, s))$minimum
}

# ---- Evaluate policy on a grid and compare with direct maximization ----
```

```r
s_grid <- seq(0, Smax, length.out = 101)
x_policy <- vapply(s_grid, policy_x, numeric(1))

x_direct <- vapply(s_grid, function(s) {
  if (s <= 0) return(0)
  objective <- function(x) -(p * x - 0.5 * c_mc * x^2 + beta * V_hat(s - x, theta_star))
  optimize(objective, c(0, s))$minimum
}, numeric(1))

par(mfrow = c(1, 2), mar = c(4.2, 4.5, 2.5, 1.2))
plot(s_grid, x_policy, type = "l", lwd = 2, col = "#2C7BB6",
     xlab = "Stock s", ylab = "Policy x*(s)", main = "Policy via Euler (V')")
lines(s_grid, x_direct, lty = 2, lwd = 2, col = "#D95F02")
legend("topleft", c("Euler/FOC", "Direct maximize"), lty = c(1, 2), lwd = 2,
       col = c("#2C7BB6", "#D95F02"), bty = "n")

plot(s_grid, V_hat(s_grid, theta_star), type = "l", lwd = 2, col = "#1A9641",
     xlab = "Stock s", ylab = "V(s)", main = "Approximated Value Function")
```



```r
# ---- Simulate a path using the policy until depletion ----
simulate_path <- function(S0, T = 50) {
  s <- S0
  S <- numeric(T + 1); X <- numeric(T)
  S[1] <- s
  t_last <- T
  for (t in 1:T) {
    x <- policy_x(s)
    X[t] <- x
    s <- max(s - x, 0)
    S[t + 1] <- s
    if (s <= 1e-8) { t_last <- t; break }
  }
  list(S = S[1:(t_last + 1)], X = X[1:t_last])
}

path <- simulate_path(S0 = 8, T = 60)
t_idx <- seq_along(path$X)

par(mfrow = c(1, 2), mar = c(4.2, 4.5, 2.5, 1.2))
plot(t_idx, path$X, type = "l", lwd = 2, col = "#2C7BB6",
     xlab = "t", ylab = "x_t", main = "Extraction over time")
plot(0:(length(path$S) - 1), path$S, type = "l", lwd = 2, col = "#1A9641",
     xlab = "t", ylab = "S_t", main = "Stock over time")
```



3.4.1 What we did

- Approximated $V(s)$ on $[0, S_{\max}]$ with N Chebyshev polynomials by collocation (re-solving for θ^*).
- Computed $\hat{V}'(s)$ analytically from the basis to use the Euler condition $p - cx - \beta \hat{V}'(s - x) = 0$.
- Built the optimal policy $x^*(s)$ by root-finding with safe fallbacks to bound checks or direct 1D maximization.
- Simulated optimal paths (x_t, S_t) using the recovered policy.

The Euler-based policy and direct maximization coincide numerically, validating the use of the Chebyshev approximation and its derivative to recover decisions efficiently.

3.5 Summary

3.5.1 Purpose and Intuition

- Collocation turns an infinite-dimensional functional equation (like the Bellman equation) into a finite system of algebraic equations.
- Instead of finding $V(s)$ for all s, we find an approximation $\hat{V}(s;\theta)$ that satisfies the Bellman equation exactly at a finite number of nodes.

- This makes solving dynamic programming problems with continuous state variables computationally tractable.

3.5.2 Core Ideas

Approximate the value function with Chebyshev basis functions

$$\hat{V}(s;\theta) = \sum_{j=0}^{N-1} \theta_j T_j(z), \quad z = \frac{2s}{S_{\max}} - 1.$$

Enforce the Bellman equation at collocation points s_i:

$$\hat{V}(s_i;\theta) = \max_{x \in X(s_i)} \{\pi(s_i, x) + \beta \hat{V}(f(s_i, x);\theta)\}, \quad i = 1, \dots, M.$$

This yields M equations in the unknown coefficients θ.

3.5.3 Computational Steps

1. Choose a basis — e.g., Chebyshev polynomials or splines.
2. Select collocation nodes — where the Bellman equation will be enforced.
3. Construct residuals — the deviation between LHS and RHS of the Bellman equation.
4. Solve for θ — using nonlinear solvers until residuals vanish.

3.5.4 Why Chebyshev Polynomials Work Well

- They provide global accuracy and numerical stability.
- Nodes are chosen to minimize oscillations (Runge's phenomenon).
- Orthogonality makes coefficients independent and accelerates convergence.
- Derivatives can be computed easily, allowing direct solution of Euler or HJB equations.

3.5.5 Key conceptual takeaway

Collocation transforms dynamic optimization from solving for functions to solving for coefficients. Once θ^* is found, the entire policy and value functions are easily computed, differentiated, and simulated.