

Model Predictive Control and Reinforcement Learning

– Lecture 03: Markov Decision Processes, Dynamic Programming and LQR –

Jasper Hoffmann

with material adapted from Moritz Diehl and Joschka Boedecker

University of Freiburg

School on Model Predictive Control and Reinforcement Learning
Freiburg, 7-16 October 2026

universität freiburg



NTNU

Norwegian University of
Science and Technology

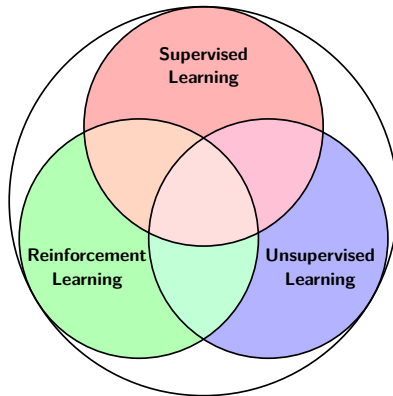
Reinforcement Learning

Markov Decision Processes

Dynamic Programming and LQR

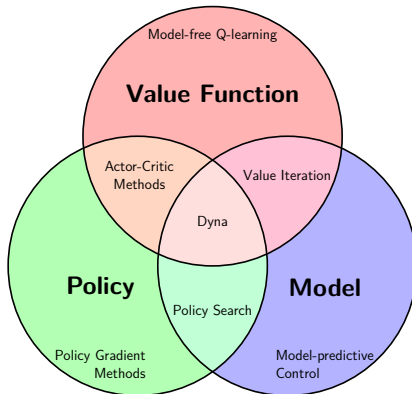
Bellman Equations, Policy and Value Iteration

Summary



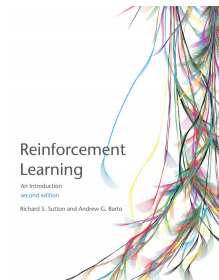
Sutton & Barto (2018): Reinforcement learning (RL) is the third pillar of machine learning.

Note: Boundaries overlap in practice.



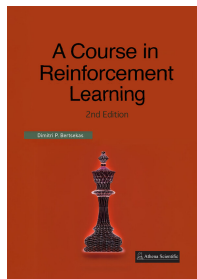
Note: In general distinctions are blurry, RL people might say that MPC is also RL.

Today: Value functions and policies with access to a *given* model



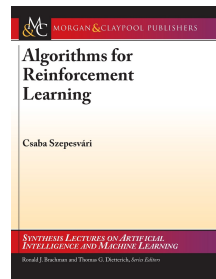
Reinforcement Learning:
An Introduction
(Sutton and Barto, 2018)

<http://incompleteideas.net/book/the-book.html>



A Course in Reinforcement
Learning
(Bertsekas, 2025)

<https://web.mit.edu/dimitrib/www/RLbook.html>



Algorithms for Reinforcement
Learning
(Szepesvári, 2010)

[https:](https://sites.ualberta.ca/~szepesva/RLBook.html)

[//sites.ualberta.ca/~szepesva/RLBook.html](https://sites.ualberta.ca/~szepesva/RLBook.html)

Same problem, different language: rewards vs. costs, max vs. min, agent vs. controller. We use **costs** and **minimize**.

Slide contents are partially based on the books above and the Reinforcement Learning lecture by David Silver.

Reinforcement Learning

Markov Decision Processes

Dynamic Programming and LQR

Bellman Equations, Policy and Value Iteration

Summary

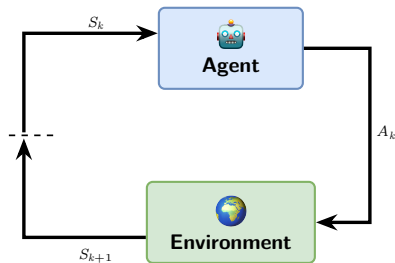
Time steps k : $0, 1, 2, \dots$

States: $S_0, S_1, S_2, \dots$

Actions: $A_0, A_1, A_2, \dots$

Given an initial state $S_0 = s_0$, the agent and environment generate the interaction sequence

$$S_0, A_0, S_1, A_1, S_2, \dots$$



Note. Capital letters denote random variables; lowercase letters denote realizations.

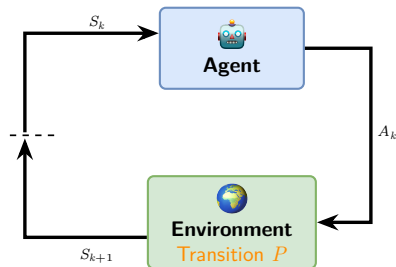
Markov property: the next state depends only on the current state and action,

$$S_{k+1} \sim P(\cdot \mid S_k, A_k).$$

Future is independent of past given present.

Finite state space: $P(s' \mid s, a)$ is the probability of transitioning to s' , thus $\sum_{s'} P(s' \mid s, a) = 1$.

Deterministic dynamics: $S_{k+1} = f(S_k, A_k)$.





Given S_k and A_k , the next state and the cost must not depend on the earlier history (Markov property).

Typical choices, given measurements Y_k (in RL often denoted O_k , observations):

- ▶ Measurement: $S_k = Y_k$, if it contains everything relevant.
- ▶ History: the last n measurements and actions, $S_k = (Y_k, A_{k-1}, Y_{k-1}, \dots, A_{k-n}, Y_{k-n})$.
- ▶ State estimate: $S_k = (\hat{x}_k, \Sigma_k)$, mean and covariance of a Kalman filter.
- ▶ Augmented: e.g. add the previous action A_{k-1} or task information such as a reference r_k .

In practice, S_k is often only approximately Markov. RL is applied anyway; the performance loss grows roughly with the information that S_k misses.

Note. Sutton & Barto (2018), Sec. 17.3; Dong, Van Roy & Zhou (2022), JMLR.

What makes an agent good? Minimize costs



Cost: We quantify the transition quality through a transition cost function

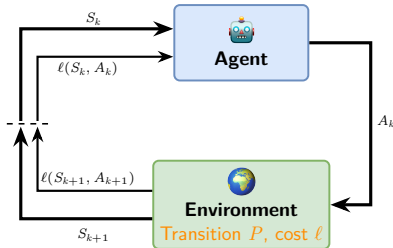
$$\ell : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}, \quad \ell(S_k, A_k).$$

Feedback Loop: At each time step k , the environment provides cost feedback, yielding the interaction sequence

$$S_0, A_0, \ell(S_0, A_0), S_1, A_1, \ell(S_1, A_1), \dots$$

Convention: Standard RL *maximizes* a reward signal R_k , which is obtained via

$$R_k := -\ell(S_k, A_k).$$



Return: The return is defined as the *discounted cumulative cost*

$$G_k := \ell(S_k, A_k) + \gamma \ell(S_{k+1}, A_{k+1}) + \gamma^2 \ell(S_{k+2}, A_{k+2}) + \cdots = \sum_{i=0}^{\infty} \gamma^i \ell(S_{k+i}, A_{k+i}).$$

with discount factor $\gamma \in (0, 1]$.

Goal of the agent: Minimize the expected discounted return $\mathbb{E}[G_k]$.

Why discount with γ ?

- ▶ The horizon can be infinite. Discounting with $\gamma < 1$ avoids an infinite cumulative cost for infinite horizon problems.
- ▶ It weights short-term more than long-term costs.

Note. In control theory, one often wants to show that a controller is stabilizing. Thus, no discount factor is used and we want to have that the controller controls the system such that the costs vanish over time.



All of what we mean by goals and purposes can be well thought of as the optimization of the expected value of the cumulative sum of a received scalar signal (called reward).

Sutton

Examples (as costs):

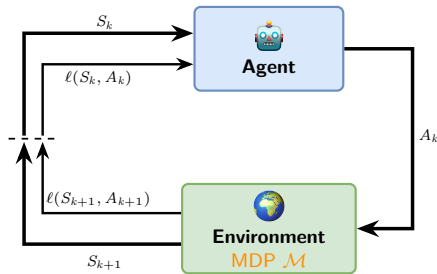
- ▶ Chess: -1 for winning, $+1$ for losing
- ▶ Walking: negative cost for forward progress and staying upright.
- ▶ Tracking controller: $\ell(s, a) = \|s - s^{\text{ref}}\|_Q^2 + \|a\|_R^2$

Definition

A Markov decision process (MDP) is a 5-tuple $\mathcal{M} := (\mathcal{S}, \mathcal{A}, P, \ell, \gamma)$, where

- ▶ $\mathcal{S}$ is the set of states,
- ▶ $\mathcal{A}$ is the set of actions,
- ▶ $P : \mathcal{S} \times \mathcal{A} \rightarrow \text{Dist}(\mathcal{S})$ is the stochastic transition model,
- ▶ $\ell : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \cup \{\infty\}$ is the cost function,
- ▶ $\gamma \in (0, 1]$ is the discount factor.

The agent interacts with the environment $\mathcal{M}$ through its actions.





A house cleaning robot:

- ▶ Three charge levels: *high*, *low* and *empty*.
- ▶ At every time step, it can *clean* or *recharge*, unless its battery is empty.
- ▶ When cleaning, the charge drops one level with probability ρ .
- ▶ Cleaning is preferable to recharging, but the robot must not run out of battery.

Formalize this problem as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \ell, \gamma)$. How to choose cost ℓ ?

Example: MDP Cleaning Robot



Solution:

- ▶ $\mathcal{S} = \{\text{high}, \text{low}, \text{empty}\}$, $\mathcal{A} = \{\text{clean}, \text{recharge}\}$.
- ▶ P is given by the transition graph, e.g.,

$$P(\text{empty} \mid \text{low}, \text{clean}) = \rho,$$

$$P(\text{low} \mid \text{low}, \text{clean}) = 1 - \rho,$$

$$P(\text{high} \mid \text{low}, \text{recharge}) = 1.$$

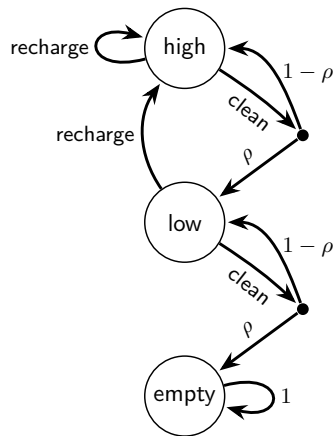
- ▶ Use simple stage costs:

$$\ell(s, \text{clean}) = -1, \quad s \in \{\text{high}, \text{low}\},$$

$$\ell(s, \text{recharge}) = +1, \quad s \in \{\text{high}, \text{low}\},$$

$$\ell(\text{empty}, a) = +10.$$

- ▶ Choose $\gamma < 1$ for the infinite-horizon problem.



Note. We minimize costs: cleaning gives negative cost, while recharging and being empty are undesirable.

A **policy** specifies how agents choose actions:

- ▶ In general, the action may depend on the entire interaction history $S_0, A_0, S_1, A_1, \dots$.
- ▶ For an MDP, we focus on **Markov policies**, which depend only on the current state:

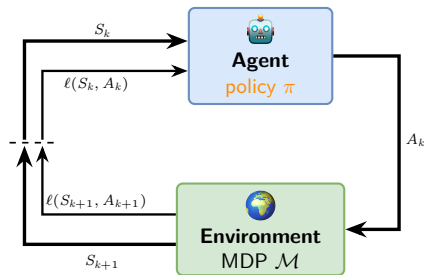
$$A_k \sim \pi(\cdot \mid S_k).$$

- ▶ A Markov policy can be
 - ▶ **stochastic:**

$$\pi(a \mid s) = \Pr(A_k = a \mid S_k = s),$$

- ▶ or **deterministic:**

$$A_k = \mu(S_k).$$



Fix a policy π , we obtain $S_0, A_0, S_1, A_1, \dots$ by

$$A_k \sim \pi(\cdot \mid S_k), \quad S_{k+1} \sim P(\cdot \mid S_k, A_k).$$

The **value function** $V^\pi(s)$ is the expected discounted cost when starting in state s and following policy π :

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \ell(S_k, A_k) \mid S_0 = s \right].$$

Thus, $V^\pi(s)$ answers:

What is the expected future cost when starting in s and following π ?

A policy is **optimal** if its expected cost is no larger than that of any other policy, at every state.

Optimality of a policy π^*

A policy π^* is called *optimal* if

$$V^{\pi^*}(s) \leq V^{\pi}(s), \quad \forall s \in \mathcal{S}, \forall \pi.$$

The corresponding *optimal value function* is

$$V^*(s) = \min_{\pi} V^{\pi}(s).$$

Finding π^* by searching over all possible policies is generally impractical.

Reinforcement Learning

Markov Decision Processes

Dynamic Programming and LQR

Bellman Equations, Policy and Value Iteration

Summary

Our goal is to find the optimal value function V^* and an optimal policy π^* .

To introduce the key idea, we first consider:

- ▶ known **deterministic** dynamics $s_{k+1} = f(s_k, a_k)$,
- ▶ a **finite horizon** N with terminal cost $\bar{V}(s_N)$,
- ▶ no discounting, thus $\gamma = 1$.

Finite-Horizon Optimal Control Problem

Starting from $s_0 = \bar{s}_0$, choose $a_0, \dots, a_{N-1}$ to minimize

$$\min_{a_0, \dots, a_{N-1}} \sum_{k=0}^{N-1} \ell(s_k, a_k) + \bar{V}(s_N)$$

where $s_{k+1} = f(s_k, a_k)$ determines the state trajectory.

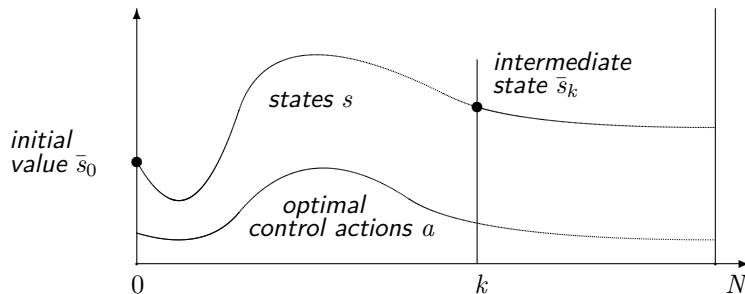
For now, this is a planning problem. We will link it later to a policy.



An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Bellman, 1957

The remaining part of an optimal plan must itself be optimal from the state reached.



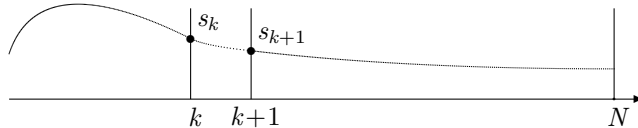
From state $\bar{s}_k$, the remaining actions are still optimal.

- Introduce the **optimal cost-to-go** (value function) on $[k, N]$:

$$V_k^*(\bar{s}) := \min_{s_k, a_k, \dots, s_N} \sum_{i=k}^{N-1} \ell(s_i, a_i) + \bar{V}(s_N) \quad \text{with} \quad s_k = \bar{s}, \quad s_{i+1} = f(s_i, a_i)$$

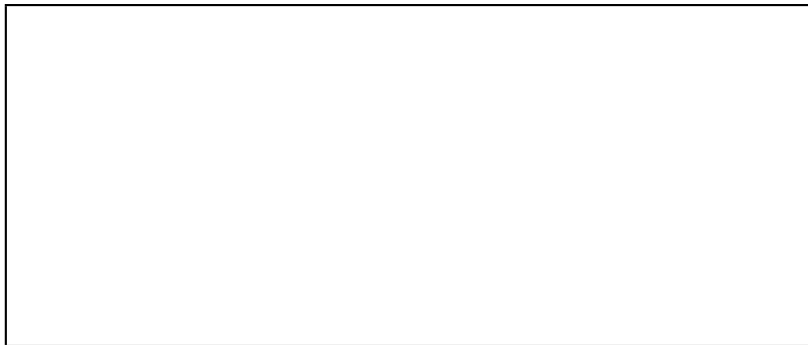
- Use the **principle of optimality** on the interval $[k, k+1]$ and eliminate s_{k+1} :

$$V_k^*(s) = \min_a \ell(s, a) + V_{k+1}^*(f(s, a))$$



Iterate backwards, starting from $V_N^*(s) := \bar{V}(s)$ for all $s \in \mathcal{S}$,
for $k = N - 1, N - 2, \dots, 0$:

$$V_k^*(s) = \min_a \ell(s, a) + V_{k+1}^*(f(s, a))$$



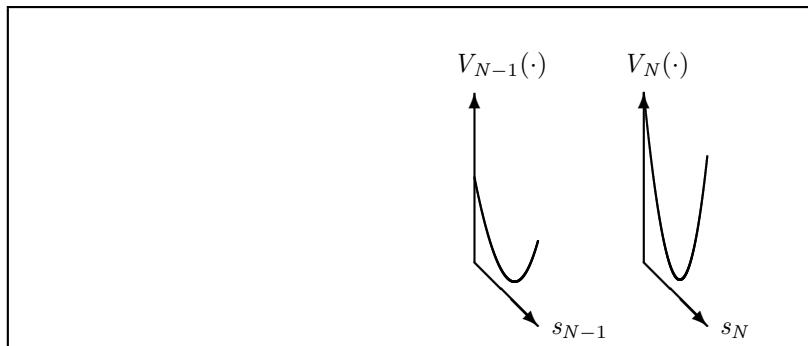
Iterate backwards, starting from $V_N^*(s) := \bar{V}(s)$ for all $s \in \mathcal{S}$,
for $k = N - 1, N - 2, \dots, 0$:

$$V_k^*(s) = \min_a \ell(s, a) + V_{k+1}^*(f(s, a))$$



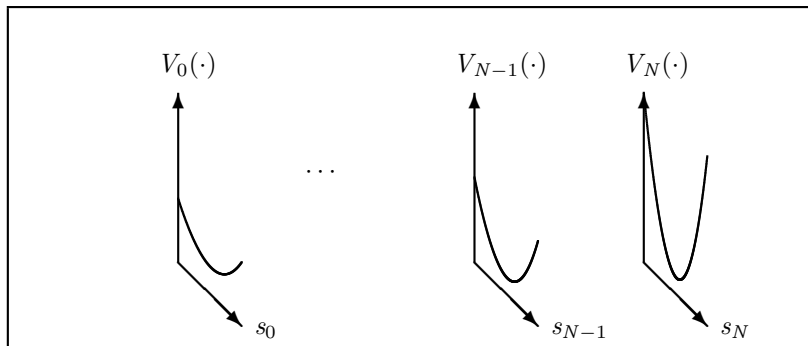
Iterate backwards, starting from $V_N^*(s) := \bar{V}(s)$ for all $s \in \mathcal{S}$,
for $k = N - 1, N - 2, \dots, 0$:

$$V_k^*(s) = \min_a \ell(s, a) + V_{k+1}^*(f(s, a))$$



Iterate backwards, starting from $V_N^*(s) := \bar{V}(s)$ for all $s \in \mathcal{S}$,
for $k = N - 1, N - 2, \dots, 0$:

$$V_k^*(s) = \min_a \ell(s, a) + V_{k+1}^*(f(s, a))$$



The recursion minimizes the **Q-function**:

$$V_k^*(s) = \min_a \underbrace{\ell(s, a) + V_{k+1}^*(f(s, a))}_{=: Q_k^*(s, a)}$$

and the **optimal feedback law or optimal policy** at time k is

$$\mu_k^*(s) := \arg \min_a Q_k^*(s, a).$$

- ▶ The laws $(\mu_0^*, \dots, \mu_{N-1}^*)$ give the optimal action for any state at any time.
- ▶ Simulating $s_{k+1}^* = f(s_k^*, \mu_k^*(s_k^*))$ from $s_0^* = \bar{s}_0$ recovers the optimal trajectory.

Recall: dynamic programming gives us the optimal action for *every* state. Therefore, it requires $V_k^*(s)$ for *every* state s .

- ▶ For a **finite state space**, we can compute and store $V_k^*(s)$ for every state.
- ▶ For a **continuous state space**, we need a suitable numerical representation of the value function.
- ▶ For example, a grid with m points in each of n_s state dimensions already contains

$$m^{n_s} \text{ grid points.}$$

The resulting exponential growth of computational and storage requirements with dimension is known as the **curse of dimensionality**.

What casts the pall over our victory celebration? It is the curse of dimensionality.

Bellman, 1961

Consider linear dynamics and quadratic costs:

$$s_{k+1} = f(s_k, a_k), \quad f(s, a) = As + Ba,$$
$$\ell(s, a) = \begin{bmatrix} s \\ a \end{bmatrix}^\top \begin{bmatrix} Q & S^\top \\ S & R \end{bmatrix} \begin{bmatrix} s \\ a \end{bmatrix}, \quad \bar{V}(s) = s^\top Q_N s.$$

Q, R, S are cost weights; Q is not the Q-function.

Key observation: if the next value function is quadratic,

$$V_{k+1}^*(s) = s^\top X_{k+1} s,$$

then the Bellman recursion becomes

$$V_k^*(s) = \min_a \begin{bmatrix} s \\ a \end{bmatrix}^\top \begin{bmatrix} Q + A^\top X_{k+1} A & S^\top + A^\top X_{k+1} B \\ S + B^\top X_{k+1} A & R + B^\top X_{k+1} B \end{bmatrix} \begin{bmatrix} s \\ a \end{bmatrix}.$$

Quadratic value function in $\implies$ quadratic value function out.

For LQR, the value function is completely described by a matrix:

$$V_k^*(s) = s^\top X_k s.$$

Starting from $X_N = Q_N$, dynamic programming only needs to propagate X_k backwards.

Difference Riccati equation:

$$X_k = Q + A^\top X_{k+1} A - (S^\top + A^\top X_{k+1} B)(R + B^\top X_{k+1} B)^{-1}(S + B^\top X_{k+1} A).$$

The corresponding optimal feedback is

$$\mu_k^*(s) = -K_k s, \quad K_k = (R + B^\top X_{k+1} B)^{-1}(S + B^\top X_{k+1} A).$$

Instead of storing V_k^* over the state space, DP only needs to propagate the matrix X_k .



Now return to the **infinite-horizon** problem. For LQR, the optimal value function is stationary:

$$V^*(s) = s^\top X s.$$

The Riccati recursion therefore becomes the fixed-point equation $X = \mathcal{R}(X)$.

Discrete-Time Algebraic Riccati Equation:

$$X = Q + A^\top X A - (S^\top + A^\top X B)(R + B^\top X B)^{-1}(S + B^\top X A).$$

Linear Quadratic Regulator (LQR)

The optimal feedback and value function are

$$\mu^*(s) = -K s, \quad K = (R + B^\top X B)^{-1}(S + B^\top X A), \quad V^*(s) = s^\top X s.$$

Excursion: Why No Discounting in LQR?



For the deterministic infinite-horizon LQR, we can instead consider the **undiscounted case** $\gamma = 1$. Under suitable conditions, the optimal feedback stabilizes the closed-loop system.

Sufficient conditions:

$$Q = C^\top C, \quad R \succ 0,$$

with (A, B) controllable and (A, C) observable.

[Bertsekas, 2012]

Then $A - BK$ is Schur stable: states and inputs decay exponentially.

$$s_{k+1} = (A - BK)s_k, \quad \|s_k\| \leq M\rho^k\|s_0\|, \quad 0 < \rho < 1.$$

Hence, the quadratic stage costs decay geometrically, so their infinite sum is finite:

$$\sum_{k=0}^{\infty} \ell(s_k, a_k) < \infty.$$

The infinite-horizon cost can therefore be finite without discounting.

Example: The Winter Hill

Consider a vehicle trying to reach and stabilize at the **top of a hill**.

State and input:

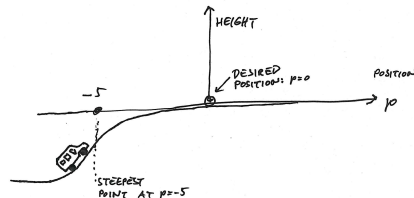
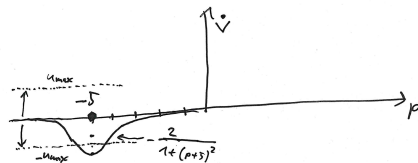
$$s = \begin{bmatrix} p \\ v \end{bmatrix}, \quad |a| \leq 1.5.$$

Nonlinear dynamics:

$$\dot{p} = v, \quad \dot{v} = a - \frac{2}{1 + (p + 5)^2}.$$

Stage cost:

$$\ell(s, a) = p^2 + 0.01v^2 + 0.01a^2.$$



[Baumgaertner]

Target: $p = 0, v = 0$

Can LQR solve this nonlinear, constrained problem?

Example: Winter Hill

To apply LQR, we **linearize the nonlinear dynamics locally** around the target equilibrium

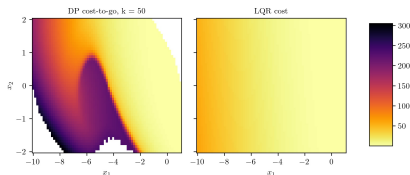
$$s^* = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad a^* = \frac{1}{13}.$$

With $\delta s = s - s^*$ and $\delta a = a - a^*$, the first-order approximation is

$$\delta \dot{s} \approx A \delta s + B \delta a, \quad A = \left. \frac{\partial f}{\partial s} \right|_{(s^*, a^*)}, \quad B = \left. \frac{\partial f}{\partial a} \right|_{(s^*, a^*)},$$

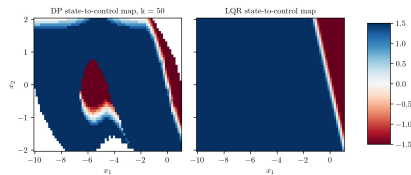
This local LQR design also ignores the input constraint $|a| \leq 1.5$.

Cost-to-go $V^*(s)$



[by K. Baumgärtner]

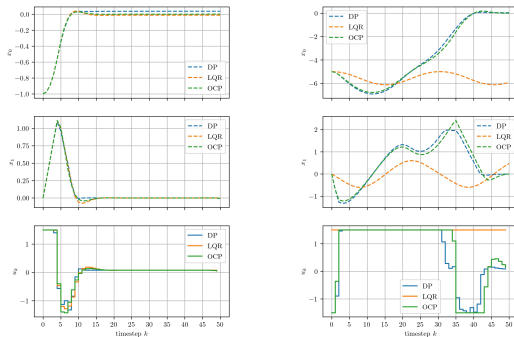
Feedback policy $\mu^*(s)$



[by K. Baumgärtner]

LQR is accurate locally; DP accounts for the full nonlinear, constrained problem.

Closed-loop: $S_0, A_0, S_1, A_1, \dots$



[by K. Baumgärtner]

- ▶ Near the target, **LQR and DP behave similarly**: the linear approximation is accurate.
- ▶ Far from the target, **DP captures the nonlinear dynamics and the input constraint**.

Reinforcement Learning

Markov Decision Processes

Dynamic Programming and LQR

Bellman Equations, Policy and Value Iteration

Summary

We have already seen dynamic programming in a special case.

	LQR	General MDP
Dynamics	$s_{k+1} = As_k + Ba_k$	$S_{k+1} \sim P(\cdot \mid S_k, A_k)$
Cost	$\ell(s, a)$ quadratic	$\ell(s, a)$ general
Value	$V^*(s) = s^\top X s$	$V^*(s)$ general
Policy	$\mu^*(s) = -Ks$	π^* general
DP	Riccati equation	Bellman equation

Same principle of optimality, different structure.

Recall the infinite-horizon MDP

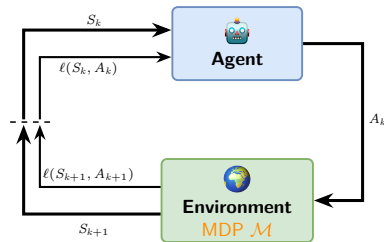
$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \ell, \gamma).$$

At each time step,

$$A_k \sim \pi(\cdot \mid S_k), \quad S_{k+1} \sim P(\cdot \mid S_k, A_k).$$

For a fixed policy π ,

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i \ell(S_{k+i}, A_{k+i}) \mid S_k = s \right].$$



The return can be split into the current cost and the remaining return:

$$G_k = \ell(S_k, A_k) + \gamma G_{k+1}.$$

For a fixed policy π , the expected remaining cost after reaching S_{k+1} is $V^\pi(S_{k+1})$. Hence,

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[\ell(S_k, A_k) + \gamma V^\pi(S_{k+1}) \mid S_k = s] \\ &= \sum_a \pi(a \mid s) \left[\ell(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s') \right]. \end{aligned}$$

Bellman equation for V^π

$$V^\pi(s) = \sum_a \pi(a \mid s) \left[\ell(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s') \right].$$

Current value = expected current cost + discounted future value.

For a fixed policy π , we follow whatever actions π prescribes:

$$V^\pi(s) = \sum_a \pi(a | s) \left[\ell(s, a) + \gamma \sum_{s'} P(s' | s, a) V^\pi(s') \right].$$

For the optimal value function, we may choose the first action. After that, the remaining decisions must again be optimal.

Bellman optimality equation

$$V^*(s) = \min_a \left\{ \ell(s, a) + \gamma \sum_{s'} P(s' | s, a) V^*(s') \right\}.$$

Fixed policy: average over π $\implies$ **Optimal: choose the best action**

This is the finite-horizon recursion

$$V_k^*(s) = \min_a \{ \ell(s, a) + V_{k+1}^*(f(s, a)) \},$$

now stochastic, discounted, and stationary.

In finite-horizon DP, we start from a terminal value and work backwards. In infinite horizon, there is no terminal time.

Starting from an arbitrary guess V_0 , repeatedly update

Iterative policy evaluation

$$V_{j+1}(s) = \sum_a \pi(a | s) \left[\ell(s, a) + \gamma \sum_{s'} P(s' | s, a) V_j(s') \right].$$

One **Bellman sweep** applies this update across the state space.

For $\gamma < 1$,

$$V_j \longrightarrow V^\pi.$$

Repeated sweeps make the estimate Bellman-consistent.

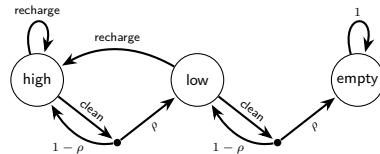
Example: Policy Evaluation for the Cleaning Robot

Let $\rho = 0.1$, $\gamma = 0.9$, and initialize $V_0 = 0$, with

$$\ell(s, \text{clean}) = -1, \quad \ell(\text{empty}, a) = 10.$$

Policy to evaluate: always *clean*,

$$\pi(\text{clean} \mid \text{high}) = \pi(\text{clean} \mid \text{low}) = 1.$$



Bellman sweep:

$$V_{j+1}(\text{high}) = -1 + 0.9[0.9 V_j(\text{high}) + 0.1 V_j(\text{low})],$$

$$V_{j+1}(\text{low}) = -1 + 0.9[0.9 V_j(\text{low}) + 0.1 V_j(\text{empty})],$$

$$V_{j+1}(\text{empty}) = 10 + 0.9 V_j(\text{empty}).$$

Resulting updates

	V_0	V_1	V_2	V_3	V_4	$\dots$	V^π
high	0	-1	-1.90	-2.62	-3.13	$\dots$	14.68
low	0	-1	-0.91	-0.03	1.42	$\dots$	42.11
empty	0	10	19.00	27.10	34.39	$\dots$	100

Repeated Bellman sweeps propagate the cost of running empty backwards.

Given V^π , take action a once and then continue with π :

Action-value function

$$Q^\pi(s, a) = \ell(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s').$$

Choose a greedy deterministic policy

$$\mu'(s) \in \arg \min_a Q^\pi(s, a).$$

The greedy policy is no worse than the original policy:

$$V^{\mu'}(s) \leq V^\pi(s) \quad \text{for all } s \in \mathcal{S}.$$

Evaluate the current policy, then improve it greedily.



Algorithm 1 Policy Iteration

- 1: Initialize a policy π
 - 2: **repeat**
 - 3: **Evaluate:** compute V^π
 - 4: **Improve:** $\mu(s) \in \arg \min_a Q^\pi(s, a)$ for every s
 - 5: $\pi \leftarrow \mu$
 - 6: **until** the policy no longer changes
 - 7: **return** π
-

$$\pi_0 \longrightarrow V^{\pi_0} \longrightarrow \pi_1 \longrightarrow V^{\pi_1} \longrightarrow \dots \longrightarrow \pi^\star.$$

Policy iteration fully evaluates a policy before improving it. **Value iteration** interleaves both steps by applying the optimal Bellman update directly.

Algorithm 2 Value Iteration

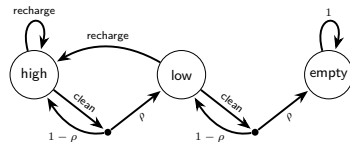
```
1: Input: tolerance  $\varepsilon > 0$ ; initialize  $V$  arbitrarily
2: repeat
3:   for each  $s \in \mathcal{S}$  do
4:      $V_{\text{new}}(s) \leftarrow \min_a \{ \ell(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V(s') \}$ 
5:   end for
6:    $\Delta \leftarrow \max_s |V_{\text{new}}(s) - V(s)|, \quad V \leftarrow V_{\text{new}}$ 
7: until  $\Delta < \varepsilon$ 
8: return a policy greedy with respect to  $V$ 
```

Each loop is one optimal Bellman sweep. For $\gamma < 1$, $V \rightarrow V^*$.

Example: Value Iteration for the Cleaning Robot

Let $\rho = 0.1$, $\gamma = 0.9$, and initialize $V_0 = 0$, with

$$\ell(s, \text{clean}) = -1, \quad \ell(s, \text{recharge}) = 1, \quad \ell(\text{empty}, a) = 10.$$



Bellman update at *low*:

$$Q_j(\text{low}, \text{clean}) = -1 + 0.9 \left[0.9 V_j(\text{low}) + 0.1 V_j(\text{empty}) \right],$$

Resulting updates

	V_0	V_1	V_2	V_3	V_4	$\dots$	V^*
high	0	-1	-1.90	-2.62	-3.19	$\dots$	-8.35
low	0	-1	-0.91	-0.71	-1.36	$\dots$	-6.51
empty	0	10	19.00	27.10	34.39	$\dots$	100

$$Q_j(\text{low}, \text{recharge}) = 1 + 0.9 V_j(\text{high}).$$

The policy implicitly defined by V_j

	π_0	π_1	π_2	π_3	π_4	$\dots$	π^*
high	clean	clean	clean	clean	clean	$\dots$	clean
low	clean	clean	recharge	recharge	recharge	$\dots$	recharge
empty	-	-	-	-	-	$\dots$	-

The final update is

$$V_{j+1}(\text{low}) = \min_a \{ Q_j(\text{low}, a) \}.$$



Write the right-hand sides of the Bellman equations as operators:

$$(\mathcal{T}^\pi V)(s) := \sum_a \pi(a | s) \left[\ell(s, a) + \gamma \sum_{s'} P(s' | s, a) V(s') \right],$$
$$(\mathcal{T} V)(s) := \min_a \left\{ \ell(s, a) + \gamma \sum_{s'} P(s' | s, a) V(s') \right\}.$$

The Bellman equations are fixed-point equations:

$$V^\pi = \mathcal{T}^\pi V^\pi, \quad V^* = \mathcal{T} V^*.$$

Policy evaluation iterates $\mathcal{T}^\pi$; value iteration iterates $\mathcal{T}$.

For $\gamma < 1$, both operators are contractions in the sup norm, for example

$$\|\mathcal{T} V - \mathcal{T} W\|_\infty \leq \gamma \|V - W\|_\infty.$$

For a fixed policy π ,

$$(\mathcal{T}_Q^\pi Q)(s, a) := \ell(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \sum_{a'} \pi(a' \mid s') Q(s', a').$$

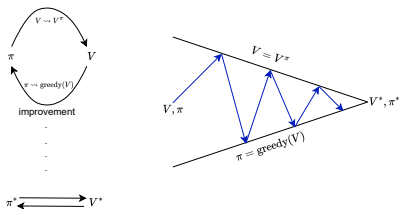
For optimal control,

$$(\mathcal{T}_Q Q)(s, a) := \ell(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \min_{a'} Q(s', a').$$

The corresponding fixed-point equations are

$$Q^\pi = \mathcal{T}_Q^\pi Q^\pi, \quad Q^* = \mathcal{T}_Q Q^*.$$

These equations will lead directly to sampled TD updates.



- **Policy evaluation** makes the value estimate more accurate for the current policy.
- **Policy improvement** makes the policy greedier with respect to the current value estimate.

Policy iteration and value iteration interleave these processes to different degrees.

Many RL algorithms follow the same pattern, using approximate values and sampled updates.



Winter Hill illustrated the trade-off: **general DP is flexible, but computing a value function over the whole state space does not scale.**

- LQR** exploit special structure to simplify DP,
- MPC** solve a finite-horizon problem online from the current state,
- RL** learn or approximate values and policies from data (next lectures).

Different ways of dealing with the complexity of optimal control.



- ▶ **Dynamic programming** uses the principle of optimality to relate current and future costs.
- ▶ **LQR** exploits linear-quadratic structure: value functions are quadratic and DP becomes the Riccati recursion.
- ▶ The **Bellman equation** evaluates a fixed policy; the **Bellman optimality equation** chooses the best action.
- ▶ **Policy iteration** alternates evaluation and improvement; **value iteration** combines both in a single update.
- ▶ Exact tabular DP requires a known model and values over the whole state space.

Next: replace exact Bellman updates using a known model by updates from sampled transitions.



- ▶ Richard S. Sutton and Andrew G. Barto: *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018. Chapters 3 and 4. <http://incompleteideas.net/book/the-book.html>
- ▶ Dimitri P. Bertsekas: *A Course in Reinforcement Learning*. Athena Scientific, 2nd edition, 2025. Chapter 1. <https://web.mit.edu/dimitrib/www/RLbook.html>
- ▶ Dimitri P. Bertsekas: *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.
- ▶ Csaba Szepesvári: *Algorithms for Reinforcement Learning*. Morgan & Claypool, 2010.
- ▶ Dimitri P. Bertsekas: *Dynamic Programming and Optimal Control*. Athena Scientific, Vol. I (2017) and Vol. II (2012).
- ▶ James B. Rawlings, David Q. Mayne and Moritz Diehl: *Model Predictive Control: Theory, Computation, and Design*. Nob Hill, 2nd edition, 2017.