Model Predictive Control and Reinforcement Learning
– Lecture 4: Dynamic Programming and Linear Quadratic Regulator –

Joschka Boedecker and Moritz Diehl

University Freiburg

July 27, 2021

Overview

Overview

Bellman's Principle of Optimality

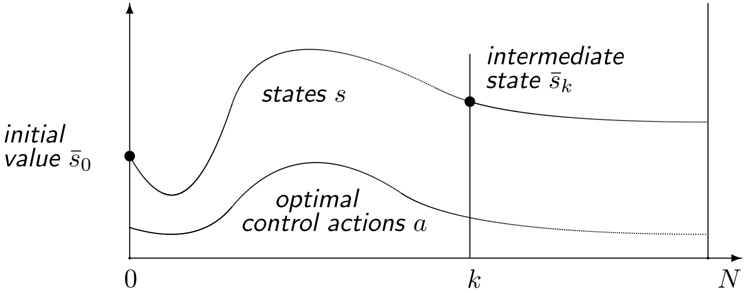
"Principle of Optimality:

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]

Principle of Optimality - Visualization

Any subarc of an optimal trajectory is also optimal.



Subarc on $[k, N]$ is optimal solution for initial value $\bar{s}_k$.

Prelude: Expressing Constraints via Infinite Cost Values

Can assign infinite cost to infeasible points, using extended reals $\bar{\mathbb{R}} := \mathbb{R} \cup \{\infty, -\infty\}$

Constrained Optimal Control Problem

$$\begin{aligned}
\min_{s,a} \quad & \sum_{k=0}^{N-1} c(s_k, a_k) + E(s_N) \\
\text{s.t.} \quad & s_0 = \bar{s}_0 \\
& s_{k+1} = f(s_k, a_k) \\
& 0 \geq h(s_k, a_k), \ k = 0, \ldots, N-1 \\
& 0 \geq r(s_N)
\end{aligned}$$

Equivalent Unconstrained Formulation

$$\begin{aligned}
\min_{s,a} \quad & \sum_{k=0}^{N-1} \bar{c}(s_k, a_k) + \bar{E}(s_N) \\
\text{s.t.} \quad & s_0 = \bar{s}_0 \\
& s_{k+1} = f(s_k, a_k), \ k = 0, \ldots, N-1
\end{aligned}$$

with $\bar{c}(s,a) = \left\{ \begin{array}{cl} c(s,a) & \text{if } h(s,a) \leq 0 \\ \infty & \text{else} \end{array} \right\}$

and $\bar{E}(s) = \left\{ \begin{array}{cl} E(s) & \text{if } r(s) \leq 0 \\ \infty & \text{else} \end{array} \right\}$

Prelude: Expressing Constraints via Infinite Cost Values

Can assign infinite cost to infeasible points, using extended reals $\bar{\mathbb{R}} := \mathbb{R} \cup \{\infty, -\infty\}$

Equivalent Unconstrained Formulation

$$\begin{aligned} \min_{s,a} \quad & \sum_{k=0}^{N-1} \bar{c}(s_k, a_k) + \bar{E}(s_N) \\ \text{s.t.} \quad & s_0 = \bar{s}_0 \\ & s_{k+1} = f(s_k, a_k), \; k = 0, \ldots, N-1 \end{aligned}$$

with $\bar{c}(s,a) = \left\{ \begin{array}{cc} c(s,a) & \text{if } h(s,a) \leq 0 \\ \infty & \text{else} \end{array} \right\}$

and $\bar{E}(s) = \left\{ \begin{array}{cc} E(s) & \text{if } r(s) \leq 0 \\ \infty & \text{else} \end{array} \right\}$

Prelude: Expressing Constraints via Infinite Cost Values

Can assign infinite cost to infeasible points, using extended reals $\bar{\mathbb{R}} := \mathbb{R} \cup \{\infty, -\infty\}$

Equivalent Unconstrained Formulation

$$\begin{aligned} \min_{s,a} \quad & \sum_{k=0}^{N-1} c(s_k, a_k) + E(s_N) \\ \text{s.t.} \quad & s_0 = \bar{s}_0 \\ & s_{k+1} = f(s_k, a_k), \ k = 0, \ldots, N-1 \end{aligned}$$

with $c : \mathbb{S} \times \mathbb{A} \to \bar{\mathbb{R}}$

and $E : \mathbb{S} \to \bar{\mathbb{R}}$

Dynamic Programming Cost-to-go

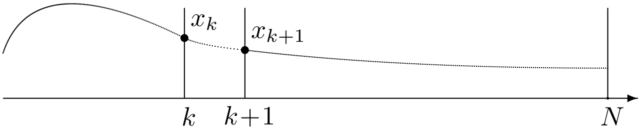
IDEA:

- Introduce **optimal-cost-to-go** function on $[k, N]$

$$J_k(x) := \min_{s_k, a_k, \ldots, s_N} \sum_{i=k}^{N-1} c(s_i, a_i) + E(s_N) \quad \text{s.t.} \quad s_k = x, \ldots$$

- Use **principle of optimality** on intervals $[k, k+1]$:

$$J_k(x_k) = \min_{s_k, a_k, s_{k+1}} c(s_k, a_k) + J_{k+1}(s_{k+1})$$
$$\text{s.t.} \quad s_k = x_k, \; s_{k+1} = f(s_k, a_k)$$



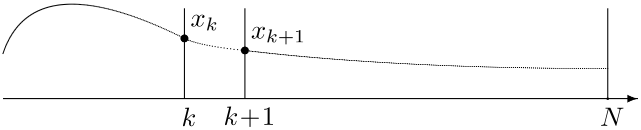
Dynamic Programming Step

Can simplify

$$J_k(x_k) = \min_{s_k, a_k, s_{k+1}} c(s_k, a_k) + J_{k+1}(s_{k+1})$$
$$\text{s.t.} \quad s_k = x_k, \; s_{k+1} = f(s_k, a_k)$$

by trivial elimination of s_k, s_{k+1} to

$$J_k(x_k) \;=\; \min_{a_k} \; c(x_k, a_k) + J_{k+1}(f(x_k, a_k))$$



Dynamic Programming Recursion

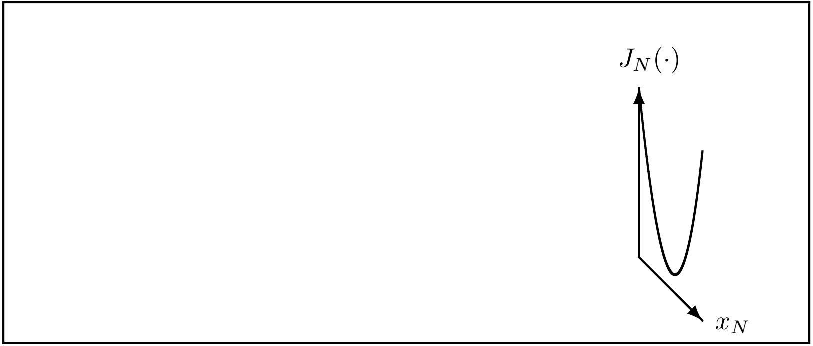
Iterate backwards, starting from $J_N(x) := E(x)$ for all $x \in \mathbb{S}$
for $k = N-1, N-2, \ldots$

$$J_k(x) = \min_a c(x,a) + J_{k+1}(f(x,a))$$

Dynamic Programming Recursion

Iterate backwards, starting from $J_N(x) := E(x)$ for all $x \in \mathbb{S}$
for $k = N-1, N-2, \dots$

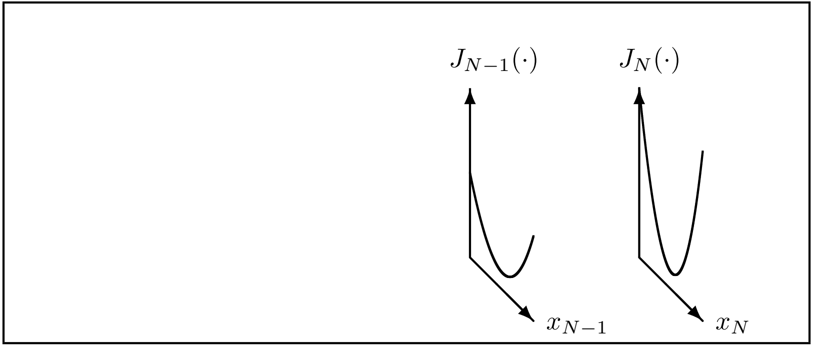
$$J_k(x) = \min_a c(x,a) + J_{k+1}(f(x,a))$$



Dynamic Programming Recursion

Iterate backwards, starting from $J_N(x) := E(x)$ for all $x \in \mathbb{S}$
for $k = N-1, N-2, \dots$

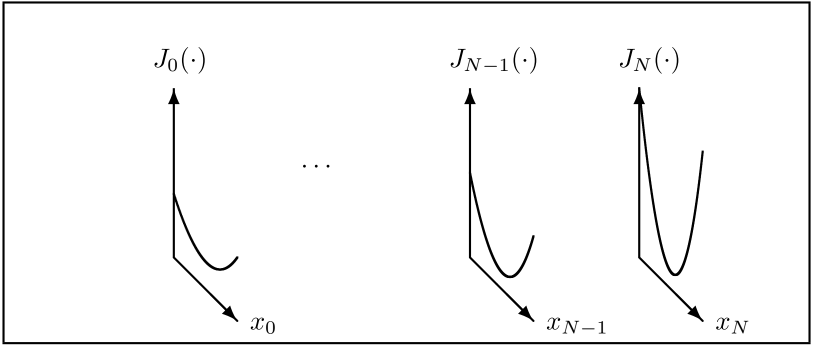
$$J_k(x) = \min_a c(x,a) + J_{k+1}(f(x,a))$$



Dynamic Programming Recursion

Iterate backwards, starting from $J_N(x) := E(x)$ for all $x \in \mathbb{S}$
for $k = N-1, N-2, \ldots$

$$J_k(x) = \min_a c(x,a) + J_{k+1}(f(x,a))$$



The Q-function and the optimal feedback control policy

The finite horizon Bellman recursion is based on minimizing what is often called the **Q-function**:

$$\begin{aligned} J_k(s) &= \min_a \underbrace{c(s,a) + J_{k+1}(f(s,a))}_{=:Q_k(s,a)} \\ &= \min_a Q_k(s,a) \end{aligned}$$

and the **optimal feedback control law** π_k^* at time k is defined by

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

These feedback laws together define the **optimal feedback control policy** $(\pi_0^*, \ldots, \pi_{N-1}^*)$ which tells us for any state s at any time index k what would be the optimal control action.

How to get optimal trajectories ?

The optimal policy $(\pi_0^*, \ldots, \pi_{N-1}^*)$ allows us to solve the original optimal control problem.

Starting with $s_0^* := \bar{s}_0$, we simulate the closed loop system for $k = 0, 1, \ldots, N-1$:

$$\begin{aligned} a_k^* &:= \pi_k^*(s_k^*) \\ s_{k+1}^* &:= f(s_k^*, a_k^*) \end{aligned}$$

yielding the optimal trajectories $s^* = (s_0^*, \ldots, s_N^*)$ and $a^* = (a_0^*, \ldots, a_N^*)$ that solve problem (1).

Optimal Control Problem

$$\begin{aligned} \min_{s,a} \quad & \sum_{k=0}^{N-1} c(s_k, a_k) + E(s_N) \\ \text{s.t.} \quad & s_0 = \bar{s}_0 \\ & s_{k+1} = f(s_k, a_k), \\ & \qquad k = 0, \ldots, N-1 \end{aligned} \tag{1}$$

How to get optimal trajectories ?

The optimal policy $(\pi_0^*, \ldots, \pi_{N-1}^*)$ allows us to solve the original optimal control problem.

Starting with $s_0^* := \bar{s}_0$, we simulate the closed loop system for $k = 0, 1, \ldots, N-1$:

$$\begin{aligned} a_k^* &:= \pi_k^*(s_k^*) \\ s_{k+1}^* &:= f(s_k^*, a_k^*) \end{aligned}$$

yielding the optimal trajectories $s^* = (s_0^*, \ldots, s_N^*)$ and $a^* = (a_0^*, \ldots, a_N^*)$ that solve problem (1).

Optimal Control Problem

$$\begin{aligned} \min_{s,a} \; & \sum_{k=0}^{N-1} c(s_k, a_k) + E(s_N) \\ \text{s.t.} \; & s_0 = \bar{s}_0 \\ & s_{k+1} = f(s_k, a_k), \\ & \qquad k = 0, \ldots, N-1 \end{aligned} \tag{1}$$

Note: MPC applies only $\pi_0^*(\bar{s}_0)$. The MPC law can be generated in one of three ways:
(a) via dynamic programming,
(b) via online solution of (1) in classical MPC, or
(c) via offline solution of (1) based on parametric programming in **explicit MPC**.

Bellman's curse of dimensionality

(a) Exact Dynamic Programming is an elegant and powerful way to solve any optimal control problem to global optimality, independent of convexity. It can be interpreted an efficient implementation of an exhaustive search that explores all possible control actions for all possible circumstances.

However, it requires the tabulation of cost-to-go functions $J_k(s)$ for all possible states $s \in \mathbb{S}$. Thus, it is exactly implementable only for discrete state and action spaces, and otherwise requires a discretization of the state space. Its computational complexity grows exponentially in the state dimension. This "curse of dimensionality", a phrase coined by Richard Bellman, unfortunately makes exact DP impossible to appy to systems with larger state dimensions.

(b) Classical MPC does circumvent this problem by restricting itself to finding only the optimal trajectory that starts at the current state s_0.

(c) Explicit MPC suffers from the same curse of dimensionality as dynamic programming.

Illustrative DP Example: Early Stage of a Pandemic (1)

Regard early stage of a new pandemic, where an infectious disease appears in an insular country (e.g. Australia) and nearly nobody is immune to it yet. Sampling time is the period during which a person remains infectious (one week). The state of the system in week k is the number $s_k \equiv I_k$ of infectious people, with state space $\mathbb{S} = \mathbb{N}$. Via social distancing, the government can control the reproduction number $a_k \equiv R_k$ that describes how many new infections an infected person produces on average. This number can be varied in the interval $\mathbb{A} = [R_{\min}, R_{\max}]$ with $R_{\min} = 0.5$ and $R_{\max} = 4$. The system dynamics $s^+ = f(s, a)$ is given by

$$I_{k+1} = f(I_k, R_k) := \lfloor R_k \; I_k \rfloor$$

The government assumes very high economic costs associated to small values of R, and zero costs for doing nothing (i.e. for $R = R_{\max}$). It also puts a small penalty $\epsilon = 10^{-4}$ on every infected person. The stage cost $c(s, a)$ is given by

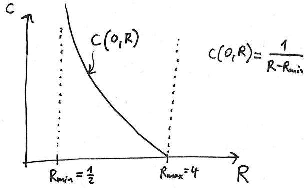
$$c(I, R) := \frac{1}{R - R_{\min}} - \frac{1}{R_{\max} - R_{\min}} + \epsilon I$$

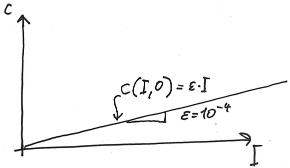
There is no terminal cost, i.e., $E(s) = 0$.

Illustrative DP Example: Early Stage of a Pandemic (2)

$$f(I,R) := \lfloor R\, I \rfloor$$

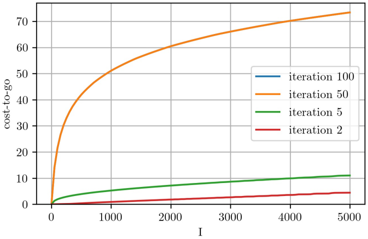
$$c(I,R) = \frac{1}{R - R_{\min}} - \frac{1}{R_{\max} - R_{\min}} + \epsilon I$$

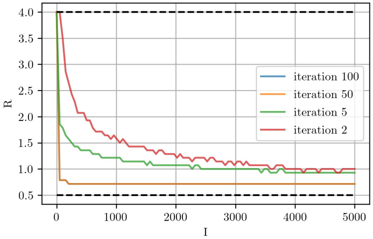


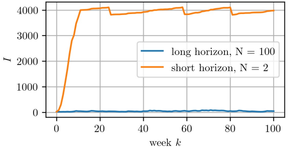


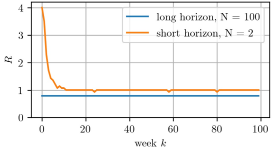
Illustrative DP Example: Early Stage of a Pandemic (3)

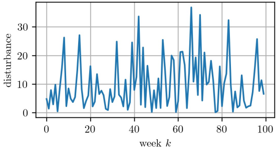
Cost-to-go function J_0, feedback law π_0^*, and
MPC closed-loop simulations $s^+ = f(s, \pi_0^*(s)) + \epsilon$
for different $N = 2, 5, 50, 100$ [by K. Baumgärtner]











Overview

Linear Quadratic Problems

Regard now linear quadratic optimal control problem of the form

$$\underset{x,u}{\text{minimize}} \quad \sum_{i=0}^{N-1} \begin{bmatrix} x_i \\ u_i \end{bmatrix}^T \begin{bmatrix} Q_i & S_i^T \\ S_i & R_i \end{bmatrix} \begin{bmatrix} x_i \\ u_i \end{bmatrix} + x_N^T P_N x_N$$

subject to

$$\begin{aligned} x_0 - \bar{x}_0 &= 0, & & \text{(initial value)} \\ x_{i+1} - A_i x_i - B_i u_i &= 0, & i = 0, \ldots, N-1, \quad & \text{(discrete system)} \end{aligned}$$

This is an equality constrained quadratic program and could thus be solved by linear algebra.

But how to apply dynamic programming here?

Linear Quadratic Recursion

If

$$c(x,u) = \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} Q & S^T \\ S & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

and

$$J_{k+1} = x^T P x$$

and

$$f(x,u) = Ax + Bu$$

then

$$J_k(x) = \min_u \quad \begin{bmatrix} x \\ u \end{bmatrix}^T \left(\begin{bmatrix} Q & S^T \\ S & R \end{bmatrix} + \begin{bmatrix} A^T P A & A^T P B \\ B^T P A & B^T P B \end{bmatrix} \right) \begin{bmatrix} x \\ u \end{bmatrix}$$

If $R + B^T P B$ is positive definite, the solution can be computed via a Schur complement.

Schur Complement Lemma

Let us simplify notation and regard

$$\phi(x) = \min_u \underbrace{\begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} Q & S^T \\ S & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}}_{=\psi(x,u)}$$

with R invertible. Then

$$\phi(x) = x^T \left(Q - S^T R^{-1} S \right) x$$

and

$$\arg\min_u \psi(x,u) = -R^{-1} S x$$

PROOF: exercise.

Riccati Recursion

The Schur Complement Lemma applied to the LQ recursion:

$$J_k(x) = \min_u \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} Q + A^T P A & S^T + A^T P B \\ S + B^T P A & R + B^T P B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

delivers directly, if $R + B^T P B$ is invertible:

$$J_k(x) = x^T P_{\text{new}} x$$

with

$$P_{\text{new}} = Q + A^T P A - (S^T + A^T P B)(R + B^T P B)^{-1}(S + B^T P A)$$

Thus, if $J_{k+1}(\cdot)$ was quadratic, also $J_k(\cdot)$ is quadratic.

Difference Riccati Equation

Backwards recursion: starting with P_N, we iterate for $k = N-1, \dots, 0$

$$\begin{array}{rcl} P_k & := & Q_k + A_k^T P_{k+1} A_k \\ & - & (S_k^T + A_k^T P_{k+1} B_k)(R_k + B_k^T P_{k+1} B_k)^{-1}(S_k + B_k^T P_{k+1} A_k) \end{array}$$

Then, we obtain the optimal feedback laws π_k^* by

$$\pi_k^*(x_k) = -\underbrace{(R_k + B_k^T P_{k+1} B_k)^{-1}(S_k + B_k^T P_{k+1} A_k)}_{=:K_k} x_k$$

and the optimal trajectory via a forward sweep started at $x_0^* := \bar{x}_0$, for $k = 0, 1, \dots, N-1$

$$\begin{array}{rcl} u_k^* & := & -K_k x_k^* \\ x_{k+1}^* & := & A_k x_k^* + B_k u_k^* \end{array}$$

Difference Riccati Equation as KKT System Solution

Interestingly, one can also obtain multipliers $\lambda_k^* := P_k x_k$.

One can show that the three trajectories $x^* = (x_0^*, \ldots, x_N^*)$, $u^* = (u_0^*, \ldots, u_{N-1}^*)$ and $\lambda^* = (\lambda_0^*, \ldots, \lambda_N^*)$ satisfy the first order (KKT) optimality conditions of the original QP, which we call a *KKT system*.

Thus, the Riccati recursion can be interpreted as a structure exploiting linear algebra routine that solves the KKT system of the original sparse QP.

Inhomogenous Linear Systems

Can we extend the Riccati recursion also to inhomogenous costs and systems? I.e. problems of the form:

$$\underset{x,u}{\text{minimize}}$$

$$\sum_{i=0}^{N-1} \begin{bmatrix} 1 \\ x_i \\ u_i \end{bmatrix}^T \begin{bmatrix} 0 & q_i^T & s_i^T \\ q_i & Q_i & S_i^T \\ s_i & S_i & R_i \end{bmatrix} \begin{bmatrix} 1 \\ x_i \\ u_i \end{bmatrix} + \begin{bmatrix} 1 \\ x_N \end{bmatrix}^T \begin{bmatrix} 0 & p_N^T \\ p_N & P_N \end{bmatrix} \begin{bmatrix} 1 \\ x_N \end{bmatrix}$$

subject to

$$\begin{aligned} x_0 - x_0^{\text{fix}} &= 0, & & \text{(initial value)} \\ x_{i+1} - A_i x_i - B_i u_i - c_i &= 0, & i = 0, \dots, N-1, \quad & \text{(discrete system)} \end{aligned}$$

Why Inhomogenous Systems and Costs?

They appear in

- Linearization of Nonlinear Systems
- Reference Tracking Problems e.g. with $c_i(x_i, u_i) = \|x_i - x_i^{\text{ref}}\|_Q^2 + \|u_i\|_R^2$
- Filtering Problems (Moving Horizon Estimation, Kalman Filter) with cost $c_i(x_i, u_i) = \|Cx_i - y_i^{\text{meas}}\|_Q^2 + \|u_i\|_R^2$
- Subproblems in active set methods or interior point methods for inequality constrained QP

A Simple Programming Trick

By augmenting the system states x_k to

$$\tilde{x}_k = \begin{bmatrix} 1 \\ x_k \end{bmatrix}$$

and replacing the dynamics by

$$\tilde{x}_{k+1} = \begin{bmatrix} 1 & 0 \\ c_k & A_k \end{bmatrix} \tilde{x}_k + \begin{bmatrix} 0 \\ B_k \end{bmatrix} u_k$$

with initial value

$$\tilde{x}_0^{\text{fix}} = \begin{bmatrix} 1 \\ x_0^{\text{fix}} \end{bmatrix}$$

This is a homogenous problem and can be solved exactly as before!

Overview

Infinite Horizon Problem

Can regard more general infinite horizon problem:

$$\underset{s,a}{\text{minimize}} \quad \sum_{i=0}^{\infty} c(s_i, a_i)$$

subject to

$$\begin{aligned} s_0 - x_0 &= 0, & & \text{(initial value)} \\ s_{i+1} - f(s_i, a_i) &= 0, \quad i = 0, \ldots, \infty, & & \text{(discrete system)} \end{aligned}$$

The Stationary Bellman Equation

Requiring stationarity of solutions of Dynamic Programming Recursion:

$$J_k = J_{k+1}$$

leads directly to the stationary Bellman Equation:

$$J(s) = \min_a \underbrace{c(s,a) + J(f(s,a))}_{=Q(s,a)}$$

The optimal controls are then obtained by the function

$$\pi^*(s) = \arg\min_a \; Q(s,a).$$

This feedback is called the stationary *Optimal Feedback Control*, the holy grail of this course.

Infinite Horizon Problems with Discounted Cost

To express that future costs matter less than immediate costs, one introduces exponentially decaying weights with discounting factor $\gamma \in (0,1)$ as follows

$$\underset{s,a}{\text{minimize}} \quad \sum_{i=0}^{\infty} (\gamma)^i \;\; c(s_i, a_i)$$

subject to

$$\begin{aligned} s_0 - x_0 &= 0, && && \text{(initial value)} \\ s_{i+1} - f(s_i, a_i) &= 0, && i = 0, \ldots, \infty, && \text{(discrete system)} \end{aligned}$$

The stationary Bellman equation then simply becomes

$$J(s) = \min_a \underbrace{c(s,a) + \gamma J(f(s,a))}_{=Q(s,a)}$$

Linear Quadratic Regulator (LQR)

Regard now LQ problem with infinite horizon and time independent system and cost:

$$\underset{x,u}{\text{minimize}} \quad \sum_{i=0}^{\infty} \begin{bmatrix} x_i \\ u_i \end{bmatrix}^T \begin{bmatrix} Q & S^T \\ S & R \end{bmatrix} \begin{bmatrix} x_i \\ u_i \end{bmatrix}$$

subject to

$$\begin{aligned} x_0 - x_0^{\text{fix}} &= 0, & & \text{(initial value)} \\ x_{i+1} - Ax_i + Bu_i &= 0, & i = 0, \ldots, \infty. \quad & \text{(discrete system)} \end{aligned}$$

How to apply dynamic programming here?

Algebraic Riccati Equation

Require stationary solution of Riccati Recursion:

$$P_k = P_{k+1}$$

i.e.

$$P = Q + A^T P A - (S^T + A^T P B)(R + B^T P B)^{-1}(S + B^T P A)$$

This is called the Algebraic Riccati Equation (in discrete time) Then, we obtain the optimal feedback $\pi^*(s)$ by

$$\pi^*(s) = -\underbrace{(R + B^T P B)^{-1}(S + B^T P A)}_{=K} s$$

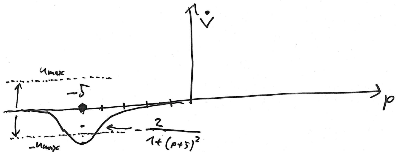
The resulting controller is called the Linear Quadratic Regulator (LQR), and K is the LQR gain. Implementing it online just requires one matrix vector multiplication: $a = -Ks$

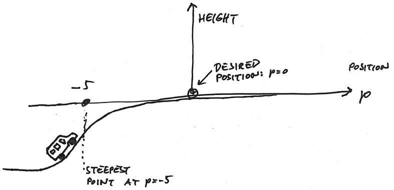
Note that the cost for an optimal trajectory starting at s_0 is $J(s_0) = s_0^\top P s_0$.

Winter Hill Example (1)

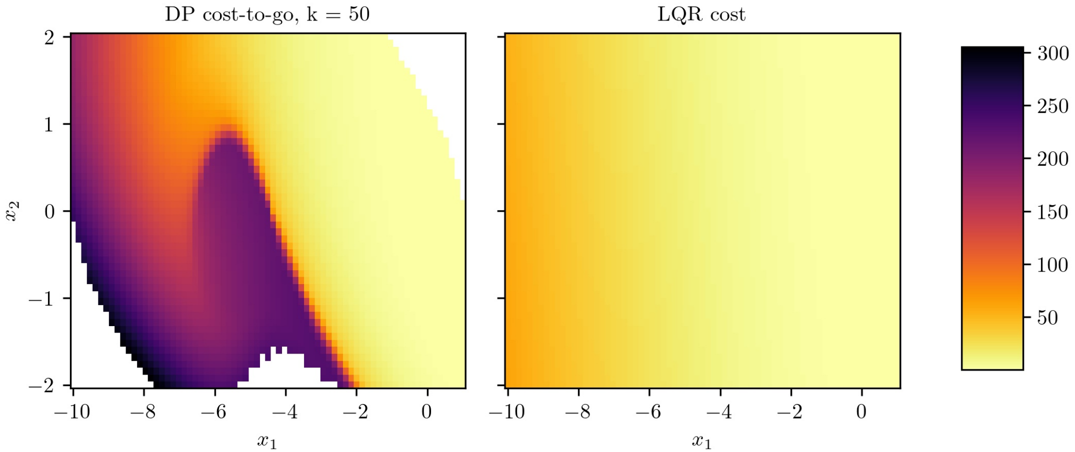
We want to drive on top of a hill in winter, on an icy road. State $x = (x_1, x_2) = (p, v)$. Tire friction too small to counteract gravity for steepest slope at $p = -5$.

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= u - \frac{2}{1+(x_1+5)^2} \\
c(x,u) &= x_1^2 + 0.01x_2^2 + 0.01u^2 \\
|u| &\leq 1.5
\end{aligned}$$



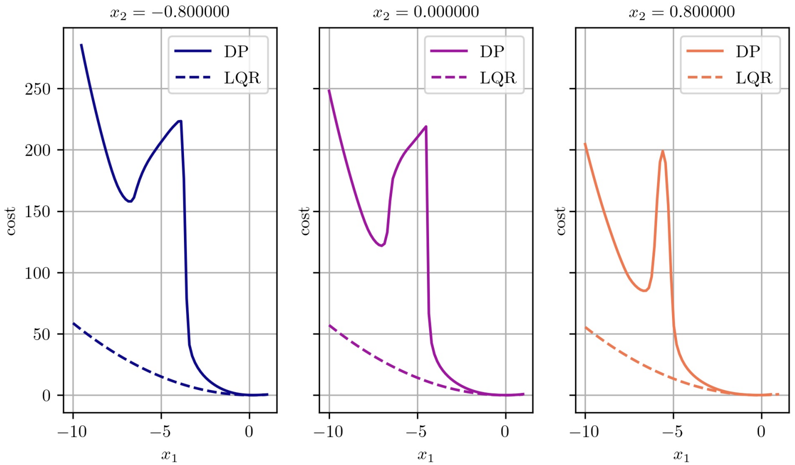


Winter Hill Example (2)



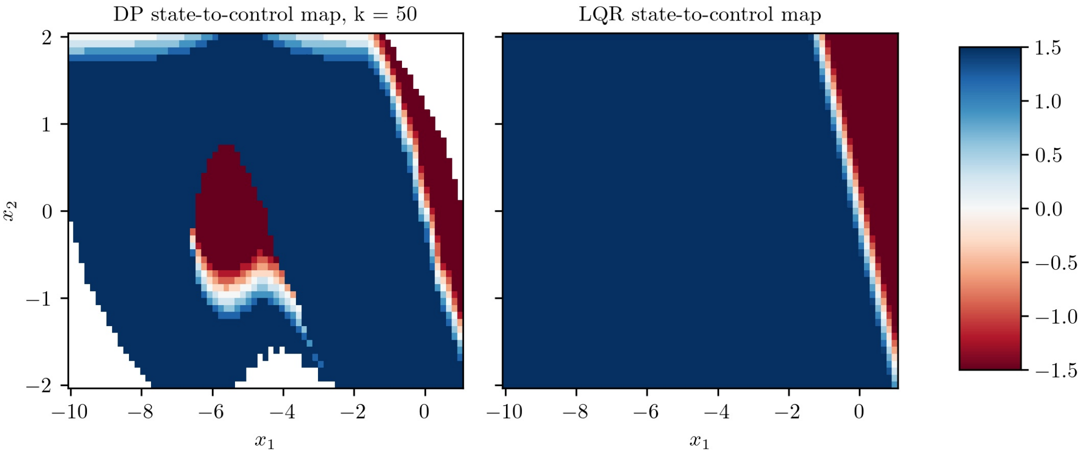
[by K. Baumgärtner]

Winter Hill Example (3)



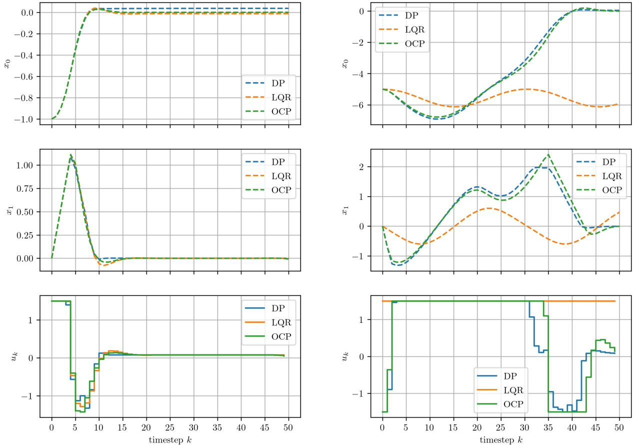
[by K. Baumgärtner]

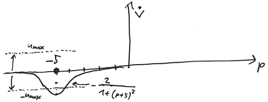
Winter Hill Example (4)

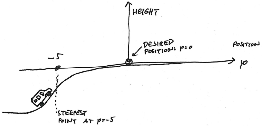


[by K. Baumgärtner]

Winter Hill Example (5)







[by K. Baumgärtner]

Overview

Stochastic Dynamic Programming

For stochastic systems, we want to find the feedback law that gives us the best expected value. We take an expectation over the disturbances ϵ_k and obtain the stochastic DP recursion:

$$J_k(s) = \min_a \underbrace{\mathbb{E}_\epsilon\{c(s,a,\epsilon) + J_{k+1}(f(s,a,\epsilon))\}}_{Q_k(s,a)}$$

where $\mathbb{E}_\epsilon\{\cdot\}$ is the expectation operator, i.e. the integral over ϵ weighted with the probability density function $p(\epsilon|s,a)$ of ϵ given s and a:

$$\mathbb{E}_\epsilon\{\ c(s,a,\epsilon)\ \} = \int c(s,a,\epsilon)\ \ p(\epsilon|s,a)\,\mathrm{d}\epsilon$$

In case of finitely many scenarios, this is just a weighted sum.
DP avoids the combinatorial explosion of scenario trees that appear in stochastic MPC.

Robust Dynamic Programming

Dynamic Programming can easily be applied to games (like chess). Here, an adverse player choses disturbances w_k against us. They influence both the stage costs c as well as the system dynamics f.
The robust DP recursion is simply:

$$J_k(s) = \min_s \underbrace{\max_w c(s,a,w) + J_{k+1}(f(s,a,w))}_{Q_k(s,a)}$$

starting with

$$J_N(s) = E(s)$$

Overview

Monotonicity of Dynamic Programming

The "cost-to-go" J_k is often also called the "value function".
The "dynamic programming operator" T acting on one value function and giving another one is defined by

$$T[J](s) := \min_a c(s,a) + J(f(s,a)).$$

Dynamic programming recursion now compactly written as $J_k = T[J_{k+1}]$.
We write $J \geq J'$ if $J(s) \geq J'(s)$ for all $s \in \mathbb{S}$.
One can prove that

$$J \geq J' \quad \Rightarrow \quad T[J] \geq T[J']$$

This is called "monotonicity" of dynamic programming. It holds also for robust or stochastic dynamic programming. It can e.g. be used in existence proofs for solutions of the stationary Bellman equation, or in stability proofs for MPC ($J_N \geq J_{N-1} \Rightarrow J_1 \geq J_0$)

Convex Dynamic Programming

Another interesting observation is that certain DP operators T preserve convexity of the value function J.

THEOREM: If system is affine and stage cost convex, i.e., if

- $f(s,a,w) = A(w)s + B(w)a + c(w)$,
- $c(s,a,w)$ is convex in (s,a)

then DP, stochastic DP, and robust DP operators T preserve convexity of J, i.e.

$$\boxed{J \text{ convex} \quad \Rightarrow \quad T[J] \text{ convex}}$$

Proof of Convexity Preservation

Regard $c(s, a, w) + J(f(s, a, w))$.
For fixed w, this is a convex function in (s, a). Because also maximum over w or expectation preserve convexity, the function

$$Q(s, a)$$

is in all three cases convex in both s and a.
Finally, the minimization of a convex function over one of its arguments preserves convexity, i.e. the resulting value function $T[J]$ defined by

$$T[J](s) = \min_a Q(s, a)$$

is convex. $\square$

Why is convexity important?

- computation of feedback law $\arg\min_a Q(s,a)$ is convex and can be solved reliably.
- can represent value function $J(s)$ more efficiently than by tabulation, e.g. as maximum of linear functions
$$J(s) = \max_i \; a_i^\top \begin{bmatrix} 1 \\ s \end{bmatrix}$$
- In robust DP, convexity of value function allows us to conclude, in case of polytopic uncertainty, that worst case is assumed on boundary of the polytope.

Summary

- Dynamic Programming recursion: $J_k(s) = \min_a c(s,a) + J_{k+1}(f(s,a))$
- feedback $\pi_k^*(s) = \arg\min_a Q_k(s,a)$ with $Q_k(s,a) := c(s,a) + J_{k+1}(f(s,a))$
- linear quadratic problems can be analytically solved (LQR) with feedback $a = -Ks$.
- in contrast to online MPC, DP suffers from curse of dimensionality
- in contrast to online MPC, DP can easily address stochastic and robust problems

Literature

- Dimitri P. Bertsekas: Dynamic Programming and Optimal Control. Athena Scientific, Belmont, 2000 (Vol I, ISBN: 1-886529-09-4) & 2001 (Vol II, ISBN: 1-886529-27-2)