Chapter 04 - LQR, iLQR, MPC

Author: Gianmarco Scarano

gianmarcoscarano@gmail.com

1. Finite-Horizon MDPs

In Finite-Horizon MDPs, we have a slightly different formulation:

$$MDP = (S, A, R, T, H, \mu_0)$$

With:

- $\boldsymbol{H \geq 0}$
  - This is the Horizon, basically the maximum number of steps that I can execute.
- $\boldsymbol{\mu_0}$
  - The initial state (s_0) is sampled from this distribution.
- $\boldsymbol{\pi = \{\pi_0, \pi_1, \pi_2, \dots, \pi_{H-1}\}}$
  - These are now time-dependent policies, meaning that actions might be different for the same, state according to t

1.1 Policy interaction

For example, we can define this quartet (s_0, a_0, s_1, a_1) as follows:

$$s_0 \sim \mu_0$$
$$a_0 = \pi_0(s_0)$$
$$s_1 \sim P(\cdot\,|s_0, a_0)$$
$$a_1 = \pi_1(s_1)$$
$$\dots$$

And so on. This goes on until we reach s_{H-1}, a_{H-1}. Please note that π_0 and π_1 are respectively the policy at time 0 and at time 1.

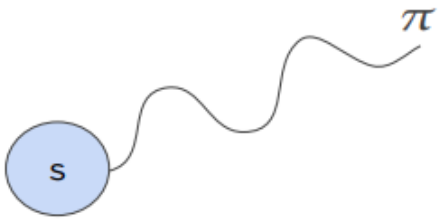
Remarks on the notation: We use capital h (H) for denoting the last time step, while we use lowercase h (h) to denote the actual time step.

1.2 Value Function, Q-Function and Bellman Equation

$$V_h^\pi(s) = \mathbb{E}_\pi[\sum_{T=h}^{H-1} r(s_T, a_T)]$$

Where $s_h = s, a_T = \pi_T(s_T)$ and $s_{T+1} \sim P(\cdot\,|s_T, a_T)$.

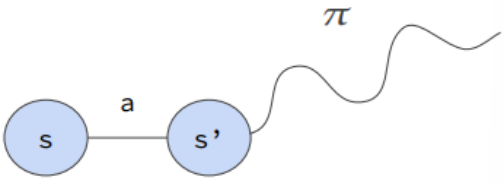
As we can see, there is no discount factor gamma (γ) and we can sum the rewards due to the fact that we are in a finite-horizon MDP.



As for Q instead, we can define it as follows:

$$Q_h^\pi(s) = \mathbb{E}_\pi[\sum_{T=h}^{H-1} r(s_T, a_T)]$$

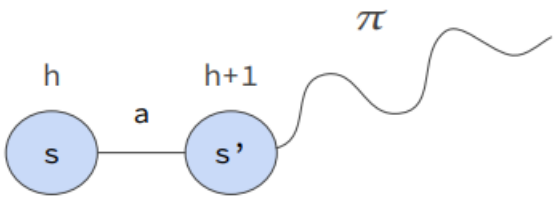
The difference here relies on the fact that the first action $a_h = a$, while before (in V) the first action was chosen by the policy π. The rest is basically the same.



The Bellman Equation in a Finite-Horizon MDP is the following:

$$Q_h^{\pi}(s,a) = r(s,a) + \mathbb{E}_{s' - p(.\,|\,s,\ a)}[V_{h+1}^{\pi}(s')]$$

Also, both Q and V depend on Time (Q_h / V_h).



1.3 Finding the Optimal Policy

In order to find the optimal policy, we want to find $\pi^* = \{\pi_0^*, \pi_1^*, \pi_2^*, \dots, \pi_{H-1}^*\}$. For doing so, we can apply dynamic programming by reasoning backwards in time.

$$Q_{H-1}{}^*(s,a) = r(s,a)$$

$$\pi_{H-1}{}^*(s) = \text{argmax}_a Q_{H-1}{}^*(s,a)$$

$$V_{H-1}{}^*(s) = \max_a Q_{H-1}{}^*(s,a) = Q_{H-1}{}^*(s, \pi_{H-1}{}^*(s))$$

As we can see, I can find everything I need by just going one step before the last one. This is done thanks to the fact that, through exploitation of time, I will arrive to a point where it will finish (it's not like the infinite horizon MDPs).

2. Control Problems

In general, in control problems we have x in $\mathbb{R}^d$ and u in $\mathbb{R}^k$.

With x we denote the state, while with u, we denote the action. So $x = s, u = a$.

We also talk about cost instead of the reward, due to the fact that cost is the inverse of reward. Simply, we want to minimize the cost instead of maximizing the reward.

Namely, we want to:

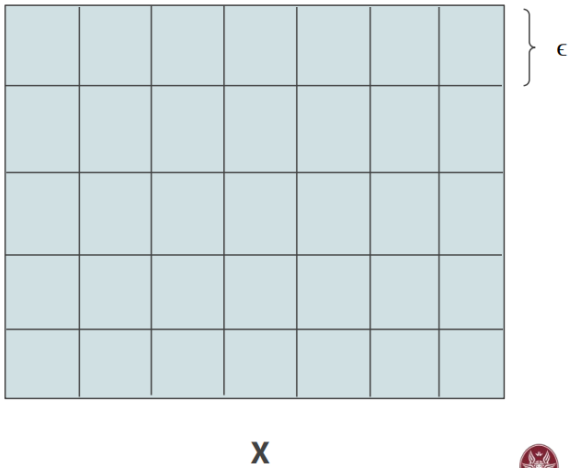
$$minimize\ \mathbb{E}_{\pi}[c_H(x_H) + \sum_{h=0}^{H-1} c_h(x_h, u_h)]$$

Where $u_h = \pi(x_h)$ and $x_0 \sim \mu_0$ and $c_H(x_H)$ is the FINAL cost (because the summation arrives until the 2nd last state).

Now this looks like a familiar problem. We can treat this as a Finite-Horizon MDP and use Value Iteration etc, only if we discretize the state space and the action space.

2.1 Discretization

If I have an n-dimensional discrete space, we could take an empty square and a value ε, discretizing the (state, action) space with this ε and produce a state space that looks like a Grid World problem.



But, if we have a huge space, the number of total points on the discretized grid increases exponentially in n.

This is known as the Bellman's Curse of Dimensionality.

2.2 Linear Systems

Let's consider now a system of this kind:

$$x_{t+1} = Ax_t + Bu_t$$

Where:

- $A \in \mathbb{R}^{dxd}, B \in \mathbb{R}^{dxk}$
- x_t = State at time t
- u_t = Control (action) at time t | Dimension: K
- $x_{t+1} = TRANSITION\ FUNCTION$

2.2.1 Quadratic Cost Function

Now, let's consider a cost function of this kind:

$$c(x_t, u_t) = x_t{}^T Q x_t + u_t{}^T R u_t$$

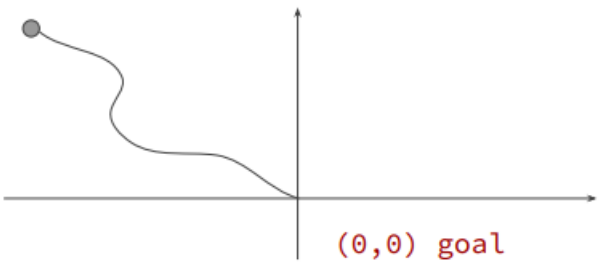
$$OR$$

$$c(x_t, u_t) = g(x_t + u_t)$$

Where:

- $Q \in \mathbb{R}^{dxd}, R \in \mathbb{R}^{kxk}$ as two square matrices
- Both Q and R are positive definite

As a result, there is a non-zero cost for any non-zero state with all-zero control (shown in the figure below).



2.2.2 Q Functions for Linear Systems and Quadratic Cost

Optimal Value Function at next time step

$$Q_h^*(x,u) = c(x,u) + \mathbb{E}_{x' \sim p(.|x,u)}[V_{h+1}^*(x')] =$$

$$x_t^T Q x_t + u_t^T R u_t + \mathbb{E}_{x' \sim p(.|x,u)}[V_{h+1}^*(x')]$$

$$x_{t+1} = Ax_t + Bu_t \quad \text{(for now: Deterministic)}$$

$$\Downarrow$$

$$x_t^T Q x_t + u_t^T R u_t + V_{h+1}^*(Ax_t + Bu_t)$$

As in induction, if we assume that $V_{h+1}^*(x_t) = x_t^T P_{h+1} x_t$ with $P \in \mathbb{R}^{dxd}$, can we also prove that $V_h^*(x_t) = x_t^T P_h x_t$? (Keep in mind that P is the Matrix encoding of Value Function).

Let's change the terms in the orange equation above, namely implementing the assumption.

$$x_t^T Q x_t + u_t^T R u_t + V_{h+1}^*(Ax_t + Bu_t)$$

$$\Downarrow \quad V_{h+1}^*(x_t) = x_t^T P_{h+1} x_t$$

$$x_t^T Q x_t + u_t^T R u_t + (Ax_t + Bu_t)^T P_{h+1} (Ax_t + Bu_t)$$

Now, we are interested in finding the $\pi_h^*(x) = argmin_u Q_h^*(x,u)$ which is the policy (in this case the action) which minimizes my Q^* (because we are in the state space of cost).

In order to find the minima era, we set the gradient (∇) of Q_h^* w.r.t u = 0 and solve for u.

1. Set $Q_h^*(x_t,u_t) = x_t^T Q x_t + u_t^T R u_t + (Ax_t + Bu_t)^T P_{h+1} (Ax_t + Bu_t)$
2. $\nabla_u Q_h^*(x_t,u_t) = 2Ru_t + 2B^T P_{h+1}(Ax_t + Bu_t) = 0$
3. Solve for $u = \pi_h^*(x_t) = -(R + B^T P_H B)^{-1} B^T P_{h+1} A x_t = K_h^* x_t$

As a result of this, I know the term u, which I simply replace in the general formula of Q_h^* (knowing also that $V_h^*(x) = max_u Q_h^*(x,u)$), obtaining $Q_h^* = x_t^T P_h x_t$. I know also the controller K.

At the end of everything, let's recall that our goal is to minimize $\mathbb{E}_\pi[c_H(x_H) + \sum_{h=0}^{H-1} c_h(x_h,u_h)]$.

So, at H, which is my final state, we only have $c_H(x_H)$ (also denoted as cost-to-go or final cost), which is typically 0 or $x_H^T Q x_H$. This is our base-case and we set P_H to 0 or Q or Q_{final}.

NOTE:

Q or Q_{final} is a square matrix that we use to define the cost function.

Q_h^* is a totally different thing (Q-function).

2.3 LQR

With this basis, we can now explicit the LQR algorithm, which is the following:

- Initialize P_H (at 0 or Q)
- Starting from h = H-1, backwards
  - Set $K_h^* = -(R+B^T P_H B)^{-1} B^T P_{h+1} A$
  - Compute $u = \pi_h^*(x_t) = K_h^* x_t$
  - Set $\mathbf{P_h = (Q + K_h^{*T} R K_h^* + (A + BK_h^*)^T P_{h+1}(A + BK_h^*))}$
  - Set $V_h^* = x_t^T P_h x_t$

Riccati Equation

P_{h+1} in the Riccati Equation it's P at next step because we are going backwards. So, everything is in function of what I have next (2nd bullet point).

In this case, we are not outputting policies, but we are outputting matrices (P).

Since V_h^* is the value function, we will rename it to J (namely, cost-to-go).

We can apply LQR to affine systems, systems with stochasticity, trajectory following for non-linear systems etc.

For affine systems, in detail:

Affine system: $Ax_t + Bu_t + \mathbf{c}$

Cost: $c(x_t,u_t)=x_t^T Q x_t + u_t^T R u_t$

Solve it **in the same way** by simply re-defining the state and transition function as

$$z_t = [x_t; 1]$$

$$z_{t+1} = \begin{bmatrix} x_{t+1} \\ 1 \end{bmatrix} = \begin{bmatrix} A & c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ 1 \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_t = A' z_t + B' u_t$$

Also, for stochastic systems where instead of adding c, we add w_t which is sampled from a zero-mean Gaussian distribution ($\mathcal{N}(0, \sigma^2 I)$.

Stochastic system: $Ax_t + Bu_t + \mathbf{w_t}$ **with** $\mathbf{w_t}$ **zero-mean Gaussian noise**

$$\mathbf{w_t \sim N(0,\sigma^2 I)}$$

Cost: $c(x_t,u_t)=x_t^T Q x_t + u_t^T R u_t$

The optimal control policy is the same; cost-to-go has an extra term depending on the variance of the syste and that cannot be controlled: $J_h^* = x_t^T P_h x_t + p_h$

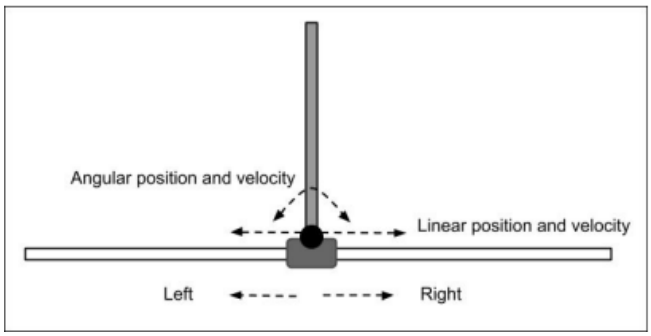
$$p_h = \mathrm{tr}(\sigma^2 P_{h+1}) + p_{h+1}$$

base-case $p_H = 0$

The last one is LQR for Non-Linear Systems (such as CartPole):

Non-linear system: $\mathbf{f(x_t, u_t)}$

Cost: $c(x_t,u_t)=x_t^T Q x_t + u_t^T R u_t$



We have a goal: x^*, u^*

We want to stabilize the system around it, such that:

$$x^* = f(x^*, u^*)$$

In this case, we want to stabilize the CartPole along a point x^*.

For doing this, we can linearize the dynamics around $f(x^*, u^*)$ using Taylor expansion:

$$x_{t+1} \approx f(x^*, u^*) + \nabla_x f(x^*, u^*)(x_t - x^*) + \nabla_u f(x^*, u^*)(x_t - x^*)$$

Simply, the two gradients can be represented as matrices A and B:

$$x_{t+1} - f(x^*, u^*) \approx \mathbf{A}(x_t - x^*) + \mathbf{B}(x_t - x^*)$$

If we impose that:

(1) $z_t = x_t - x^*$ (Difference we have above)

(2) $v_t = u_t - u^*$ (controls)

Then I can say that my problem is again a linear problem expressed in this way:

$$z_{t+1} = Az_t + Bv_t$$

Being solvable in the usual way (as in LQR):

$$(3)\ v_t = Kz_t$$

Through simple swapping operations with (1) and (2), we get that (3) now becomes:

$$u_t - u^* = K\,(x_t - x^*)$$

$$u_t = u^* + K\,(x_t - x^*)$$