Previous MDP doesn't consider action in reward function. (Infinite horizon, discount factor)
Also they can only approximate V*

In this setting we can compute exact V*

CS229 Lecture notes

Dan Boneh & Andrew Ng

Part XIV
LQR, DDP and LQG

Linear Quadratic Regulation, Differential Dynamic Programming and Linear Quadratic Gaussian

1 Finite-horizon MDPs

In the previous set of notes about Reinforcement Learning, we defined Markov Decision Processes (MDPs) and covered Value Iteration / Policy Iteration in a simplified setting. More specifically we introduced the **optimal Bellman equation** that defines the optimal value function V^{π^*} of the optimal policy π^*.

$$V^{\pi^*}(s) = R(s) + \max_{a \in \mathcal{A}} \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi^*}(s')$$

Recall that from the optimal value function, we were able to recover the optimal policy π^* with

$$\pi^*(s) = \text{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{sa}(s') V^*(s')$$

In this set of lecture notes we'll place ourselves in a more general setting:

1. We want to write equations that make sense for both the discrete and the continuous case. We'll therefore write

scribe: Guillaume Genthial

$$\mathbb{E}_{s' \sim P_{sa}} \left[V^{\pi^*}(s') \right] \qquad \text{instead of}$$
$$\sum_{s' \in S} P_{sa}(s') V^{\pi^*}(s')$$

meaning that we take the expectation of the value function at the next state. In the finite case, we can rewrite the expectation as a sum over states. In the continuous case, we can rewrite the expectation as an integral. The notation $s' \sim P_{sa}$ means that the state s' is sampled from the distribution P_{sa}.

2. We'll assume that the rewards depend on **both states and actions**. In other words, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. This implies that the previous mechanism for computing the optimal action is changed into

$$\pi^*(s) = \text{argmax}_{a \in \mathcal{A}} R(s, a) + \gamma \mathbb{E}_{s' \sim P_{sa}} \left[V^{\pi^*}(s') \right]$$

3. Instead of considering an infinite horizon MDP, we'll assume that we have a **finite horizon MDP** that will be defined as a tuple

$$(\mathcal{S}, \mathcal{A}, P_{sa}, T, R)$$

with $T > 0$ the **time horizon** (for instance $T = 100$). In this setting, our definition of payoff is going to be (slightly) different:

$$R(s_0, a_0) + R(s_1, a_1) + \cdots + R(s_T, a_T)$$

instead of (infinite horizon case)

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots$$
$$\sum_{t=0}^{\infty} R(s_t, a_t) \gamma^t$$

What happened to the discount factor γ? Remember that the introduction of γ was (partly) justified by the necessity of making sure that the infinite sum would be finite and well-defined. If the rewards are bounded by a constant $\bar{R}$, the payoff is indeed bounded by

$$|\sum_{t=0}^{\infty} R(s_t)\gamma^t| \leq \bar{R}\sum_{t=0}^{\infty}\gamma^t$$

and we recognize a geometric sum! Here, as the payoff is a finite sum, the discount factor γ is not necessary anymore.

In this new setting, things behave quite differently. First, the optimal policy π^* might be non-stationary, meaning that **it changes over time**. In other words, now we have

$$\pi^{(t)} : \mathcal{S} \to \mathcal{A}$$

where the superscript (t) denotes the policy at time step t. The dynamics of the finite horizon MDP following policy $\pi^{(t)}$ proceeds as follows: we start in some state s_0, take some action $a_0 := \pi^{(0)}(s_0)$ according to our policy at time step 0. The MDP transitions to a successor s_1, drawn according to $P_{s_0 a_0}$. Then, we get to pick another action $a_1 := \pi^{(1)}(s_1)$ following our new policy at time step 1 and so on...

Why does the optimal policy happen to be non-stationary in the finite-horizon setting? Intuitively, as we have a finite numbers of actions to take, we might want to adopt different strategies depending on where we are in the environment and how much time we have left. Imagine a grid with 2 goals with rewards +1 and +10. At the beginning, we might want to take actions to aim for the +10 goal. But if after some steps, dynamics somehow pushed us closer to the +1 goal and we don't have enough steps left to be able to reach the +10 goal, then a better strategy would be to aim for the +1 goal...

4. This observation allows us to use **time dependent dynamics**

$$s_{t+1} \sim P^{(t)}_{s_t,a_t}$$

meaning that the transition's distribution $P^{(t)}_{s_t,a_t}$ changes over time. The same thing can be said about $R^{(t)}$. Note that this setting is a better model for real life. In a car, the gas tank empties, traffic changes, etc. Combining the previous remarks, we'll use the following general formulation for our finite horizon MDP

$$\left(\mathcal{S}, \mathcal{A}, P_{sa}^{(t)}, T, R^{(t)}\right)$$

Remark: notice that the above formulation would be equivalent to adding the time into the state.

The value function at time t for a policy π is then defined in the same way as before, as an expectation over trajectories generated following policy π starting in state s.

$$V_t(s) = \mathbb{E}\left[R^{(t)}(s_t, a_t) + \cdots + R^{(T)}(s_T, a_T) | s_t = s, \pi\right]$$

Now, the question is

In this finite-horizon setting, how do we find the optimal value function

$$V_t^*(s) = \max_\pi V_t^\pi(s)$$

It turns out that Bellman's equation for Value Iteration is made for **Dynamic Programming**. This may come as no surprise as Bellman is one of the fathers of dynamic programming and the Bellman equation is strongly related to the field. To understand how we can simplify the problem by adopting an iteration-based approach, we make the following observations:

1. Notice that at the end of the game (for time step T), the optimal value is obvious

I'm guessing you need to compute all s, for V_t(s)

$$\forall s \in \mathcal{S}: \quad V_T^*(s) := \max_{a \in \mathcal{A}} R^{(T)}(s, a) \tag{1}$$

wait it says right here!

2. For another time step $0 \le t < T$, if we suppose that we know the optimal value function for the next time step V_{t+1}^*, then we have

Still the question is how to model the unknown distribution`

In the video, Andrew assumes it is determinstic

If only this a takes a deterministic model

$$\forall t < T, s \in \mathcal{S}: \quad V_t^*(s) := \max_{a \in \mathcal{A}} \left[R^{(t)}(s, a) + \mathbb{E}_{s' \sim P_{sa}^{(t)}}\left[V_{t+1}^*(s')\right]\right] \tag{2}$$

With these observations in mind, we can come up with a clever algorithm to solve for the optimal value function:

There is also the no-smart one at which you can use the old value iteration, same procedure

1. compute V_T^* using equation (1). This means knowing the end reward?

2. for $t = T-1, \ldots, 0$:

   What about the distribution?

   compute V_t^* using V_{t+1}^* using equation (2)

Side note We can interpret standard value iteration as a special case of this general case, but without keeping track of time. It turns out that in the standard setting, if we run value iteration for T steps, we get a γ^T approximation of the optimal value iteration (geometric convergence). See problem set 4 for a proof of the following result:

Theorem Let B denote the Bellman update and $||f(x)||_\infty := \sup_x |f(x)|$. If V_t denotes the value function at the t-th step, then

$$\begin{aligned} ||V_{t+1} - V^*||_\infty &= ||B(V_t) - V^*||_\infty \\ &\leq \gamma ||V_t - V^*||_\infty \\ &\leq \gamma^t ||V_1 - V^*||_\infty \end{aligned}$$

?? what is V*

In other words, the Bellman operator B is a γ-contracting operator.

2 Linear Quadratic Regulation (LQR)

In this section, we'll cover a special case of the finite-horizon setting described in Section 1, for which the **exact solution** is (easily) tractable. This model is widely used in robotics, and a common technique in many problems is to reduce the formulation to this framework.

First, let's describe the model's assumptions. We place ourselves in the continuous setting, with

$$\mathcal{S} = \mathbb{R}^n, \quad \mathcal{A} = \mathbb{R}^d$$

and we'll assume **linear transitions** (with noise)

$$s_{t+1} = A_t s_t + B_t a_t + w_t$$

where $A_t \in R^{n \times n}, B_t \in R^{n \times d}$ are matrices and $w_t \sim \mathcal{N}(0, \Sigma_t)$ is some gaussian noise (with **zero** mean). As we'll show in the following paragraphs, it turns out that the noise, as long as it has zero mean, does not impact the optimal policy!

We'll also assume **quadratic rewards**

$$R^{(t)}(s_t, a_t) = -s_t^\top U_t s_t - a_t^\top W_t a_t$$

Time variant???

where $U_t \in R^{n\times n}, W_t \in R^{d\times d}$ are positive definite matrices (meaning that the reward is always **negative**).

Remark Note that the quadratic formulation of the reward is equivalent to saying that we want our state to be close to the origin (where the reward is higher). For example, if $U_t = I_n$ (the identity matrix) and $W_t = I_d$, then $R_t = -||s_t||^2 - ||a_t||^2$, meaning that we want to take smooth actions (small norm of a_t) to go back to the origin (small norm of s_t). This could model a car trying to stay in the middle of lane without making impulsive moves...

Now that we have defined the assumptions of our LQR model, let's cover the 2 steps of the LQR algorithm

step 1 suppose that we don't know the matrices A, B, Σ. To estimate them, we can follow the ideas outlined in the Value Approximation section of the RL notes. First, collect transitions from an arbitrary policy. Then, use linear regression to find $\text{argmin}_{A,B} \sum_{i=1}^{m} \sum_{t=0}^{T-1} \left\| s_{t+1}^{(i)} - \left(A s_t^{(i)} + B a_t^{(i)}\right)\right\|^2$. Finally, use a technique seen in Gaussian Discriminant Analysis to learn Σ.

WHy do we need sigma?

where did the noise go

?? This can either come approximation or actual physical model. Like the self-drviing car model.

step 2 assuming that the parameters of our model are known (given or estimated with step 1), we can derive the optimal policy using dynamic programming.

In other words, given

In other words, A and B can be fixed as well as time varying

Compute A,B Sigma for every t?

$$\begin{cases} s_{t+1} &= A_t s_t + B_t a_t + w_t \qquad A_t, B_t, U_t, W_t, \Sigma_t \text{ known} \\ R^{(t)}(s_t, a_t) &= -s_t^\top U_t s_t - a_t^\top W_t a_t \end{cases}$$

we want to compute V_t^*. If we go back to section 1, we can apply dynamic programming, which yields

Piecewise time interval update of optimal policy to take

1. **Initialization step**

   For the last time step T,

$$\begin{aligned} V_T^*(s_T) &= \max_{a_T \in \mathcal{A}} R_T(s_T, a_T) \\ &= \max_{a_T \in \mathcal{A}} -s_T^\top U_T s_T - a_T^\top W_t a_T \\ &= -s_T^\top U_t s_T \qquad \text{(maximized for } a_T = 0) \end{aligned}$$

How do we know ST? From the approximation model at time = T?

1. Setup the goal

2. **Recurrence step**

   Let $t < T$. Suppose we know V_{t+1}^*.

   ?? how

   Fact 1: It can be shown that if V_{t+1}^* is a quadratic function in s_t, then V_t^* is also a quadratic function. In other words, there exists some matrix Φ and some scalar Ψ such that

$$\begin{aligned} &\text{if } V_{t+1}^*(s_{t+1}) = s_{t+1}^\top \Phi_{t+1} s_{t+1} + \Psi_{t+1} \\ &\text{then } V_t^*(s_t) = s_t^\top \Phi_t s_t + \Psi_t \end{aligned}$$

   $\boxed{\text{For time step } t = T\text{, we had } \Phi_t = -U_T \text{ and } \Psi_T = 0.}$

   Fact 2: We can show that the optimal policy is just a linear function of the state.

   Knowing V_{t+1}^* is equivalent to knowing Φ_{t+1} and Ψ_{t+1}, so we just need to explain how we compute Φ_t and Ψ_t from Φ_{t+1} and Ψ_{t+1} and the other parameters of the problem.

$$\begin{aligned} V_t^*(s_t) &= s_t^\top \Phi_t s_t + \Psi_t \\ &= \max_{a_t} \left[R^{(t)}(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P_{s_t, a_t}^{(t)}} [V_{t+1}^*(s_{t+1})] \right] \\ &= \max_{a_t} \left[-s_t^\top U_t s_t - a_t^\top V_t a_t + \mathbb{E}_{s_{t+1} \sim \mathcal{N}(A_t s_t + B_t a_t, \Sigma_t)} [s_{t+1}^\top \Phi_{t+1} s_{t+1} + \Psi_{t+1}] \right] \end{aligned}$$

   where the second line is just the definition of the optimal value function and the third line is obtained by plugging in the dynamics of our model along with the quadratic assumption. Notice that the last expression is a quadratic function in a_t and can thus be (easily) optimized[1]. We get the optimal action a_t^*

But the mean isn't zero

[1]Use the identity $\mathbb{E}\left[w_t^\top \Phi_{t+1} w_t\right] = \mathrm{Tr}(\Sigma_t \Phi_{t+1})$ with $w_t \sim \mathcal{N}(0, \Sigma_t)$

Scales like cube. LQR scales better, scales polynomialy. Disretization scales expontially.

Under the assumption we made, the Value function is a quadratic function of state,
the optimal policy is a linear function of current state. Even though the states are continous.

How did it get to this step>??

I need to look at my control textbook to see the derivation

$$\begin{aligned} a_t^* &= \left[(B_t^\top \Phi_{t+1} B_t - V_t)^{-1} B_t \Phi_{t+1} A_t\right] \cdot s_t \\ &= L_t \cdot s_t \end{aligned}$$

This is saying that the optimal policy is a linear function of current state, Not approxiamtion

In the lecture we assume A_t and B_t is time invariant

where

$$L_t := \left[(B_t^\top \Phi_{t+1} B_t - W_t)^{-1} B_t \Phi_{t+1} A_t\right]$$

which is an impressive result: our optimal policy is **linear in** s_t. Given a_t^* we can solve for Φ_t and Ψ_t. We finally get the **Discrete Ricatti equations**

$$\begin{aligned} \Phi_t &= A_t^\top \left(\Phi_{t+1} - \Phi_{t+1} B_t \left(B_t^\top \Phi_{t+1} B_t - W_t\right)^{-1} B_t \Phi_{t+1}\right) A_t - U_t \\ \Psi_t &= -\operatorname{tr}\left(\Sigma_t \Phi_{t+1}\right) + \Psi_{t+1} \end{aligned}$$

Fact 3: we notice that Φ_t depends on neither Ψ nor the noise Σ_t! As L_t is a function of A_t, B_t and Φ_{t+1}, it implies that the optimal policy also **does not depend on the noise**! (But Ψ_t does depend on Σ_t, which implies that V_t^* depends on Σ_t.)

Then, to summarize, the LQR algorithm works as follows

1. (if necessary) estimate parameters A_t, B_t, Σ_t
2. initialize $\Phi_T := -U_T$ and $\Psi_T := 0$.
3. iterate from $t = T-1 \ldots 0$ to update Φ_t and Ψ_t using Φ_{t+1} and Ψ_{t+1} using the discrete Ricatti equations. If there exists a policy that drives the state towards zero, then convergence is guaranteed!

Not sure what this means, the value converges?
Let's ignore it for now

Using Fact 3, we can be even more clever and make our algorithm run (slightly) faster! As the optimal policy does not depend on Ψ_t, and the update of Φ_t only depends on Φ_t, it is sufficient to update **only** Φ_t!

Special Property of LQR:

don't depend on noise term, covaraince matrix. YOu can even ignore the noise
in the updating equations for state and you still get the SAME optimal policy.

3 From non-linear dynamics to LQR

It turns out that a lot of problems can be reduced to LQR, even if dynamics are non-linear. While LQR is a nice formulation because we are able to come up with a nice exact solution, it is far from being general. Let's take for instance the case of the inverted pendulum. The transitions between states look like

$$\begin{pmatrix} x_{t+1} \\ \dot{x}_{t+1} \\ \theta_{t+1} \\ \dot{\theta}_{t+1} \end{pmatrix} = F\left(\begin{pmatrix} x_t \\ \dot{x}_t \\ \theta_t \\ \dot{\theta}_t \end{pmatrix}, a_t\right)$$

where the function F depends on the cos of the angle etc. Now, the question we may ask is

Can we linearize this system?

3.1 Linearization of dynamics

Let's suppose that at time t, the system spends most of its time in some state $\bar{s}_t$ and the actions we perform are around $\bar{a}_t$. For the inverted pendulum, if we reached some kind of optimal, this is true: our actions are small and we don't deviate much from the vertical.

We are going to use Taylor expansion to linearize the dynamics. In the simple case where the state is one-dimensional and the transition function F does not depend on the action, we would write something like

$$s_{t+1} = F(s_t) \approx F(\bar{s}_t) + F'(\bar{s}_t) \cdot (s_t - \bar{s}_t)$$

In the more general setting, the formula looks the same, with gradients instead of simple derivatives

$$s_{t+1} \approx F(\bar{s}_t, \bar{a}_t) + \nabla_s F(\bar{s}_t, \bar{a}_t) \cdot (s_t - \bar{s}_t) + \nabla_a F(\bar{s}_t, \bar{a}_t) \cdot (a_t - \bar{a}_t) \qquad (3)$$

and now, s_{t+1} is linear in s_t and a_t, because we can rewrite equation (3) as

should this be a_t

$$s_{t+1} \approx As_t + Bs_t + \kappa$$

where κ is some constant and A, B are matrices. Now, this writing looks awfully similar to the assumptions made for LQR. We just have to get rid of the constant term κ! It turns out that the constant term can be absorbed into s_t by artificially increasing the dimension by one. This is the same trick that we used at the beginning of the class for linear regression...

3.2 Differential Dynamic Programming (DDP)

The previous method works well for cases where the goal is to stay around some state s^* (think about the inverted pendulum, or a car having to stay in the middle of a lane). However, in some cases, the goal can be more complicated.

We'll cover a method that applies when our system has to follow some trajectory (think about a rocket). This method is going to discretize the trajectory into discrete time steps, and create intermediary goals around which we will be able to use the previous technique! This method is called **Differential Dynamic Programming**. The main steps are

step 1 come up with a nominal trajectory using a naive controller, that approximate the trajectory we want to follow. In other words, our controller is able to approximate the gold trajectory with

$$s_0^*, a_0^* \to s_1^*, a_1^* \to \dots$$

step 2 linearize the dynamics around each trajectory point s_t^*, in other words

$$s_{t+1} \approx F(s_t^*, a_t^*) + \nabla_s F(s_t^*, a_t^*)(s_t - s_t^*) + \nabla_a F(s_t^*, a_t^*)(a_t - a_t^*)$$

where s_t, a_t would be our current state and action. Now that we have a linear approximation around each of these points, we can use the previous section and rewrite

$$s_{t+1} = A_t \cdot s_t + B_t \cdot a_t$$

(notice that in that case, we use the non-stationary dynamics setting that we mentioned at the beginning of these lecture notes)

Note We can apply a similar derivation for the reward $R^{(t)}$, with a second-order Taylor expansion.

In the video slide, the reward function is R(s) = -||s_t - s_desired||^2

$$R(s_t, a_t) \approx R(s_t^*, a_t^*) + \nabla_s R(s_t^*, a_t^*)(s_t - s_t^*) + \nabla_a R(s_t^*, a_t^*)(a_t - a_t^*)$$
$$+ \frac{1}{2}(s_t - s_t^*)^\top H_{ss}(s_t - s_t^*) + (s_t - s_t^*)^\top H_{sa}(a_t - a_t^*)$$
$$+ \frac{1}{2}(a_t - a_t^*)^\top H_{aa}(a_t - a_t^*)$$

I think we specific problem, we have to design our reward function

Even though the reward function is defined like this I still don't get why the Helicopter follows the gold traj

where H_{xy} refers to the entry of the Hessian of R with respect to x and y evaluated in (s_t^*, a_t^*) (omitted for readability). This expression can be re-written as

I think this is similar to giving the traj space a heuristic

$$R_t(s_t, a_t) = -s_t^\top U_t s_t - a_t^\top W_t a_t$$

for some matrices U_t, W_t, with the same trick of adding an extra dimension of ones. To convince yourself, notice that

$$\begin{pmatrix} 1 & x \end{pmatrix} \cdot \begin{pmatrix} a & b \\ b & c \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \end{pmatrix} = a + 2bx + cx^2$$

step 3 Now, you can convince yourself that our problem is **strictly** re-written in the LQR framework. Let's just use LQR to find the optimal policy π_t. As a result, our new controller will (hopefully) be better!

Note: Some problems might arise if the LQR trajectory deviates too much from the linearized approximation of the trajectory, but that can be fixed with reward-shaping...

step 4 Now that we get a new controller (our new policy π_t), we use it to produce a new trajectory

\pi_0(s^*_0) = a*_t

s_0 might be old

$$s_0^*, \pi_0(s_0^*) \to s_1^*, \pi_1(s_1^*) \to \quad \to s_T^*$$

The F here is the system dynamics

note that when we generate this new trajectory, we use the real F and not its linear approximation to compute transitions, meaning that

I think my question is, how do gurantee that we end up in our desired ST state

$$s_{t+1}^* = F(s_t^*, a_t^*)$$

then, go back to step 2 and repeat until some stopping criterion.

4 Linear Quadratic Gaussian (LQG)

Often, in the real word, we don't get to observe the full state s_t. For example, an autonomous car could receive an image from a camera, which is merely an **observation**, and not the full state of the world. So far, we assumed that the state was available. As this might not hold true for most of the real-world problems, we need a new tool to model this situation: **Partially Observable MDPs**.

A POMDP is an MDP with an extra observation layer. In other words, we introduce a new variable o_t, that follows some conditional distribution given the current state s_t

$$o_t | s_t \sim O(o|s)$$

Formally, a finite-horizon POMDP is given by a tuple

$$(\mathcal{S}, \mathcal{O}, \mathcal{A}, P_{sa}, T, R)$$

Within this framework, the general strategy is to maintain a **belief state** (distribution over states) based on the observation $o_1, \ldots, o_t$. Then, a policy in a POMDP maps belief states to actions.

In this section, we'll present a extension of LQR to this new setting. Assume that we observe $y_t \in \mathbb{R}^m$ with $m < n$ such that

$$\begin{cases} y_t &= C \cdot s_t + v_t \\ s_{t+1} &= A \cdot s_t + B \cdot a_t + w_t \end{cases}$$

So C, A, B is known I assume

where $C \in R^{m \times n}$ is a compression matrix and v_t is the sensor noise (also gaussian, like w_t). Note that the reward function $R^{(t)}$ is left unchanged, as a function of the state (not the observation) and action. Also, as distributions are gaussian, the belief state is also going to be gaussian. In this new framework, let's give an overview of the strategy we are going to adopt to find the optimal policy:

step 1 first, compute the distribution on the possible states (the belief state), based on the observations we have. In other words, we want to compute the mean $s_{t|t}$ and the covariance $\Sigma_{t|t}$ of

$$s_t | y_1, \ldots, y_t \sim \mathcal{N}\left(s_{t|t}, \Sigma_{t|t}\right)$$

to perform the computation efficiently over time, we'll use the **Kalman Filter** algorithm (used on-board Apollo Lunar Module!).

step 2 now that we have the distribution, we'll use the mean $s_{t|t}$ as the best approximation for s_t

step 3 then set the action $a_t := L_t s_{t|t}$ where L_t comes from the regular LQR algorithm.

Intuitively, to understand why this works, notice that $s_{t|t}$ is a noisy approximation of s_t (equivalent to adding more noise to LQR) but we proved that LQR is independent of the noise!

Step 1 needs to be explicated. We'll cover a simple case where there is no action dependence in our dynamics (but the general case follows the same idea). Suppose that

$$\begin{cases} s_{t+1} &= A \cdot s_t + w_t, \quad w_t \sim N(0, \Sigma_s) \\ y_t &= C \cdot s_t + v_t, \quad v_t \sim N(0, \Sigma_y) \end{cases}$$

As noises are Gaussians, we can easily prove that the joint distribution is also Gaussian How we prove this?

$$\begin{pmatrix} s_1 \\ \vdots \\ s_t \\ y_1 \\ \vdots \\ y_t \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma) \qquad \text{for some } \mu, \Sigma$$

then, using the marginal formulas of gaussians (see Factor Analysis notes), we would get

Conceptually but computationally expensive

$$s_t | y_1, \ldots, y_t \sim \mathcal{N}\left(s_{t|t}, \Sigma_{t|t}\right)$$

However, computing the marginal distribution parameters using these formulas would be computationally expensive! It would require manipulating matrices of shape $t \times t$. Recall that inverting a matrix can be done in $O(t^3)$, and it would then have to be repeated over the time steps, yielding a cost in $O(t^4)$!

The **Kalman filter** algorithm provides a much better way of computing the mean and variance, by updating them over time in **constant time in**

This is not the true kalman filter, instead the hidden Markov model.
In Kalman filter,
hidden state variables take values in a continuous space
(as opposed to a discrete state space as in the hidden Markov model).???

t! The kalman filter is based on two basics steps. Assume that we know the distribution of $s_t|y_1,\dots,y_t$:

predict step compute $s_{t+1}|y_1,\dots,y_t$

update step compute $s_{t+1}|y_1,\dots,y_{t+1}$

and iterate over time steps! The combination of the predict and update steps updates our belief states. In other words, the process looks like

$$(s_t|y_1,\dots,y_t) \xrightarrow{\text{predict}} (s_{t+1}|y_1,\dots,y_t) \xrightarrow{\text{update}} (s_{t+1}|y_1,\dots,y_{t+1}) \xrightarrow{\text{predict}} \dots$$

predict step Suppose that we know the distribution of

$$s_t|y_1,\dots,y_t \sim \mathcal{N}\left(s_{t|t},\Sigma_{t|t}\right)$$

This term here contradict to what I learned for Kalman filter. Shoudn't it be y_t only ?

then, the distribution over the next state is also a gaussian distribution

$$s_{t+1}|y_1,\dots,y_t \sim \mathcal{N}\left(s_{t+1|t},\Sigma_{t+1|t}\right)$$

where

$$\begin{cases} s_{t+1|t} &= A\cdot s_{t|t} \\ \Sigma_{t+1|t} &= A\cdot\Sigma_{t|t}\cdot A^\top + \Sigma_s \end{cases}$$

update step given $s_{t+1|t}$ and $\Sigma_{t+1|t}$ such that

$$s_{t+1}|y_1,\dots,y_t \sim \mathcal{N}\left(s_{t+1|t},\Sigma_{t+1|t}\right)$$

we can prove that

$$s_{t+1}|y_1,\dots,y_{t+1} \sim \mathcal{N}\left(s_{t+1|t+1},\Sigma_{t+1|t+1}\right)$$

where

$$\begin{cases} \boxed{s_{t+1|t+1}} &= s_{t+1|t} + K_t(y_{t+1} - Cs_{t+1|t}) \\ \Sigma_{t+1|t+1} &= \Sigma_{t+1|t} - K_t\cdot C\cdot\Sigma_{t+1|t} \end{cases}$$

with

there's some notational difference between each literature. S_{t+1} | {t+1} = S_hat

$$K_t := \Sigma_{t+1|t} C^\top (C \Sigma_{t+1|t} C^\top + \Sigma_y)^{-1}$$

The matrix K_t is called the **Kalman gain**.

Now, if we have a closer look at the formulas, we notice that we don't need the observations prior to time step t! The update steps only depends on the previous distribution. Putting it all together, the algorithm first runs a forward pass to compute the K_t, $\Sigma_{t|t}$ and $s_{t|t}$ (sometimes referred to as $\hat{s}$ in the literature). Then, it runs a backward pass (the LQR updates) to compute the quantities Ψ_t, Ψ_t and L_t. Finally, we recover the optimal policy with $a_t^* = L_t s_{t|t}$.

Some of the details doesn't make sense to me

Check video description, the text here is missing something:
1:10