MachineLearning-Lecture18

Instructor (Andrew Ng):Okay. Welcome back. What I want to do today is talk about one of my favorite algorithms for controlling NVPs that I think is one of the more elegant and efficient and powerful algorithms that I know of. So what I'll do is I'll first start by talking about a couple variations of NVPs that are slightly different from the NVP definition you've seen so far. These are pretty common variations.

One is state action rewards, and the other is horizon NVPs. Using this semi-modified definition of an NVP, I'll talk about linear dynamical systems. I'll spend a little bit of time talking about models within dynamical systems, and then talk about LQR, or linear quadratic regulation control, which will lead us to some kind of [inaudible] equation, which is something we will solve in order to do LQR controls.

So just to recap, and we've seen this definition many times now. We've been defining an NVP as [inaudible] states actions, states improbabilities, [inaudible] reward function where – gamma's the discount factors, a number between zero and one. And R, the reward function, was the function mapping from the states, the rewards – was the function mapping from the states, the real numbers.

So we had value iteration, which would do this. So after a while, the value of the iteration will cause V to convert to V star. Then having found the optimal value function, if you compute the optimal policy by taking essentially [inaudible] of this equation above. Augments of A, of that [inaudible].

So in value iteration, as you iterate of this – you know, perform this update, the function V will [inaudible] convert to V star. So there won't be – so without defining the number of iterations, you get closer and closer to V star. This actually converge exponentially quickly to V star. We will never exactly convert to V star and define the number of iterations.

So what I want to do now is describe a couple of common variations of NVPs that we slightly different definitions of. Firs the reward function and then second, we'll do something slightly different from just counting. Then remember in the last lecture, I said that for infinite state of continuously in NVPs, we couldn't apply the most straightforward version of value iteration because if you have a continuous state NVP, we need to use some approximations of the optimal value function.

The [inaudible] later in this lecture, I'll talk about a special case of NVPs, where you can actually represent the value function exactly, even if you have an infinite-state space or even if you have a continuous-state space. I'll actually do that, talk about these special constants of infinite-state NVPs, using this new variation of the reward function and the alternative to just counting, so start to make the formulation a little easier.

So the first variation I want to talk about is selection rewards. So I'm going to change the definition of the reward function. If this turns out, it won't be a huge deal. In particular, I change reward function to be a function mapping from a state action pair to the real numbers.

What I mean by this is just the following. You sell off in some state in zero. You take an action A zero as a result of your state of action choice. You transition to some new state, S1. You take some action, A1. You transition to some new state, S2. You take some action, A2, and so on. So this is a [inaudible] state action sequence that you see.

So in an MPP where you have a state action reward, your total payoff is now defined as this, where your reward function is now a function both of the current state and of the action you took in the current state. So this is my total payoff.

Then as usual, my goal will be to find a policy – to find the function mapping from the state's actions, so that when I execute that policy, I can maximize the expected value of my total payoff. So this definition, it actually turns out that given an NVP with state action rewards, you can actually – so by [inaudible] with the definitions of the states, you can actually reduce this back to an NVP with only rewards that function in the states.

That may or may not be [inaudible]. Don't worry if it isn't. But using state action rewards allows you to more directly model problems in which different actions, we have different costs. So a running example is the robot. So [inaudible] a robot, and it's more costly for the robot to move than for it to stay still. If you give an action to stay still, and the action to stay still may have a lower cost because you're not using a battery power [inaudible] recharge it for that action.

Another example would be – actually, another navigation example would be if you have an outdoor vehicle. You need to drive over some sort of outdoor terrain, like very rough rocks or driving over grass. It may be costly, more difficult, than driving over, say, a paved road. So you may assign an action that requires driving over grass or driving over rocks to be more costly than driving over paved road.

So this really isn't a huge change to the definition of an NVP. I won't really bother to justify this a lot, but [inaudible] equations is generalizing the way that you probably expect it. V star of S is now equal to that.

So previously, when the reward function was just a function of the state, S, we could take the max and push it in here. But now that the rewards is a function of the action you're taking as well, the max comes outside. So this says that your expected total payoff, starting from the state, as [inaudible] policy, is equal to first your immediate reward, RFSA, for executing some action, A, in state S.

Then plus gamma times your future expected total payoff. So this is your expected total payoff if you take the action, A, from the current state. So while these [inaudible] optimal value functions. So your actually optimal expected total payoff is the max of all actions of this thing on the right.

Let's see. Value iteration, which I'm abbreviating VI, is really the same algorithm. B of S is updated as max over A, RFSA, same thing. Just [inaudible] on the right-hand side of those equations be updating V of S using [inaudible] equations. Again, you get value iteration, exactly the same way.

Then finally, having found the optimal value function, V star, using the value iteration algorithm, you can then compute the optimal policy, pi star of S as same as before. The best action to take in the state, S, is the action, A, that maximizes the thing on the right-hand side. So having used value iteration to compute the optimal value function, you can then find pi star using that.

So this was the easier of the two variations of NVPs [inaudible] so far. Any questions? Actually, are there questions about this? So the other variation, the other alternative definition, will be finite horizon NVPs. So finite horizon NVP comprises of the [inaudible] SA [inaudible] transition, probably with these, and the parameter T and the reward function. Here, T is a parameter called the horizon time. Concretely, what this really means is that we'll be taking actions in the NVP only for a total of capital T times this. So we won't use this counting anymore. [Audio cuts out]

Instructor (Andrew Ng):[Inaudible] zero, take action A0. Get to some other state S1, take action A1 and so on. Eventually, you get to some state, STAT after T times [inaudible]. So my total payoff, now, will be given by this sum from time zero up to time T of my sum over rewards. Okay? My goal, as usual – so this is my total payoff.

My goal, as usual, is to maximize the expected value of my total payoff. We want to come up with a policy to maximize the expected value of this total payoff. The key difference is that the world only will exist [inaudible], and after that, there's no more rewards to be corrected.

So this turns out to be [inaudible] of a difference because it turns out that the optimal policy may be non-stationary. The term, stationary, means that it doesn't depend on time. Non-stationary means that it may depend on time. So non-stationary roughly means that my optimal action to take will be different for different time steps. That's what non-stationary means.

Just as an example of that, imagine that we have some robot. Let's say the robot is here. Let's say that there's a good [inaudible] over there with a plus one reward. Much further away, there's a plus ten reward. So depending on how much time you have left on the clock, it may be better to go after the plus one or the plus ten reward.

If it's still early in the game, you still have a lot of time, it may be better to head toward the plus-ten rewards junction and get a much larger reward. If you only have a couple of time sets left, if the clock has nearly reached the time, capital T, then you may not have enough time to get to a plus ten reward. You've be better off heading for the plus one reward that's much more close by.

So what this example illustrates is that when you're in that state, the best action to take could be to go left or to go right, depending on what time it is. So just as an example, illustrating how the actually policy can be non-stationary.

In fact, since we have non-stationary policies anyway in the sequence, what I'm going to do next, I'm going to allow non-stationary transition probabilities as well. So I'll just write that up there. What I mean is that so far, assuming that the state ST plus one, is joined from the state transition probabilities [inaudible] by the previous states and the previous action.

I've been assuming that these state transition probabilities are the same for all times. So I want to say [inaudible] and take some action, the distribution of an innate state doesn't matter. It doesn't depend on time. So I'm going to allow a study more general definition as well, in which we have non-stationary state transition probabilities so that the chance of where you end up [inaudible] may also depend on what time it is.

So as examples of this non-stationary state transition probabilities, one example would be if you model flying an aircraft over a long distance. Then as the aircraft flies, you burn fuel and become lighter. So the dynamics of the aircraft actually change over time. The mass of the aircraft can change significantly over time as you burn fuel. So depending on what time it is, your mixed state could actually depend on not only your current state and your action, but also on how much fuel you burn, therefore, what time it is.

Other examples, another aerospace one, is if you have the weather forecast for the next 24 hours, say, you know what the winds and precipitation are going to be like over the next 24 hours. Then again, if you fly the aircraft from, say, here to New York, it may cost different amounts to fly different [inaudible] at different times. Maybe flying over the Rockies may cost different amounts, depending on whether you do it now, when there's really great weather, or if you do it a few hours from now, when the weather may be forecast really bad.

For an example you see everyday, same thing for traffic, right? There's at least – depending on where you live, certainly here in California, there are times of day where traffic is really bad in lots of places. So the costs of driving certain roads may vary, depending on what time of day it is. Lots of other examples. Industrial automation, different machines in the factory may be available to different degrees at different times of day. They cost different amounts to hire different workers, depending on whether you pay over time [inaudible] or whatever. So the cost of doing different things in the factory can also be a function of time.

The state transition probabilities can also be a function of time. Lastly, while we're doing this as well, to make this fully general, we might as well have non-stationary [inaudible] as well, where you might also index the reward function of these times and prescripts, where the cost of doing different things may depend on the time as well.

Actually, there's more examples of non-stationary NVPs, so let's – so now we have a non-stationary policy. Let's talk about an algorithm to actually try to find the optimal policy. So let me define the following. This is now a slightly modified definition of the optimal value function. I'll just write this down, I guess.

So I'm going to define the optimal value function, and this going to be indexed by T, with a subscript T. The optimal value of a state for time, T, we're going to define as your optimal sum of rewards for if you start the NVP at that state, S, and if the clock starts off at time, lowercase T.

So the optimal value of a state will depend on what time it is and how much time you have lest to run this NVP. Therefore, the sum on the right sums only for time T, time T plus one, time T plus two up to time, capital T. I'll just state in English again, this is your expected optimal total payoff if you start your system in a state, S, and if the clock is already at time, lowercase T.

So it turns out then there's a [inaudible], you can value that [inaudible]. Let me just write out the value [inaudible] algorithm for this. It turns out you can – well, let me just write this out. I'll write this below. It turns out you can compute the optimal value function for the NVP using the following recursion, which is very similar to what we have for value iteration. We're going to set V of S to be equal to [inaudible] over A, same as before, right?

Okay? So if I start the clock at time T and from state S, my expected total payoff is equal to the maximum [inaudible] actions I may take of my immediate reward. Taking that action, A, in that state, S. Them plus my expected future payoff. So if I take action, A, I would transition with [inaudible] P, subscript SA, S prime to some new state, S prime.

If I get to the state, S prime, my total expected payoff from the state S prime would be these [inaudible] now subscript T plus one, that's prime. Subscript T plus one reflects that after I've taken one action, my clock will have advanced from time T to time T plus one. So this is now V star subscript T plus one.

So this expresses V star of T in terms of V star T plus one. Then lastly, to start off this recursion, you would have V star, capital T is equal to – it's just equal to that. If you're already at time, capital T, then you just get to take one action, and then the clock runs out. So this is V star capital T. Your value of starting in some state, S, with no time – with just one time step, with no time left on the clock.

So in the case of finite horizon NVP, this actually gives up a very nice dynamic programming algorithm in which you can start off by computing V star of T. Then you use this backward [inaudible] to compute V star of capital T minus one, capital T minus two and so on. We compute V star of T and T minus one and so on. It recurs backwards onto your computer, V star for all of your time steps.

Can you see this board? Cool. Then the final step is – previously, we said that pi star of S – I'm going to come back and change this a bit – was the [inaudible] A of R plus A plus [inaudible] PSA – this is sort of what we had. In the finite horizon case, the ultimate action may depend on what time it is. So the ultimate action to take it, time T, is [inaudible] actions, A.

This is basically the augmat of exactly that same thing on the right-hand side as we had in our dynamic programming algorithm. So you do this for every time step, and now you compute it, your optimal policy for different time steps. Again, this is a non-stationary policy because pi star of S my depend on what time it is.

So [inaudible] the difference between this and the early version of the earlier version of value iteration is that – so what you do is you complete V star of T. Then using the backwards recursion of that [inaudible] algorithm, you computer V star T minus one. Then V star T minus two and so on down to V star of zero. Then from these, you compute pi star.

So one – there's not a huge difference, but one minus difference [inaudible] the infinite horizon discounted case is that by running this recursion once, you now have exactly the right value function. So this just computes the value function, rather than merely converging [inaudible]. This just gives you the right value function with one pass.

Cool. Any questions there? Yeah. Interviewee:

[Inaudible].

Instructor (Andrew Ng):This computation's much shorter than valuations. So sort of yes and no. It depends on how large capital T is. Interviewee:

[Inaudible] the normal NVP, could [inaudible] and then use this case for that case?

Instructor (Andrew Ng):I see. So for the normal NVP, can you assume capital T and then assume this? So it actually turns out that – that's a great question. Let me just answer this in a hand-wavy way. So it actually turns out for a discounted infinite horizon NVP where some discount factor gamma. So what you can do is you can say, after T times X, gamma to the T would be really small. It would be like [inaudible] something. I don't really care what happens after that many times because the rewards are multiplied by gamma to the T. After that, I don't really care.

So you can ask, can I take my infinite horizon discounted NVP and approximate that with a finite horizon NVP where the number of times, steps T, is chosen so that [inaudible] true. So it turns out you can do that. Then you end up with some value for T. You can solve for T so that this holds true. It turns out you can prove that if you took the original value iteration algorithm and if you run the original value of the [inaudible] algorithm – the version for discounted NVPs. If you run that for this same number of time steps, you will end up with an approximation to the value function that is about this close, up to some small constant factors.

So to do that, you end up with roughly the same amounts of computation anyway. Then you actually end up with a non-stationary policy, which is more expensive to keep around. You need to keep around the different policy every time step, which is not as nice as if you had the stationary policy, same policy for all times.

So there are other reasons, but sometimes you might take an infinite horizon discounted problem and approximate it to a finite horizon problem. But this particular reason is not the one. That makes sense. More questions? Interviewee:

[Inaudible]?

Instructor (Andrew Ng):Is there a gamma in this? So if you want, you can actually change the definition of an NVP and use a finite horizon discounted NVP. If you want, you can do that. You can actually come in and put a gamma there, and use this counting the finite horizon. It turns out that usually, for most problems that people deal with, you either use discounting or you use the finite horizon.

It's been less common to do both, but you can certainly do as well. One of the nice things about discounting, it makes such your value function is finite. Algorithmically and mathematically, one of the reasons to use discounting is because you're multiplying your rewards exponentially. It's a geometrically [inaudible] series. It shows that the value function is always finite. This is a really nice mathematical properties when you do discounting.

So when you have a finite horizon anyway, then the value function's also guaranteed to be finite. So with that, you don't have to use discounting. But if you want, you can actually discount as well. Interviewee:

[Inaudible].

Instructor (Andrew Ng):Yeah, yes, you're right. If you want, you can redefine the reward function to go downward into the to the reward function, since we have non-stationary rewards as well.

So that was finite horizon NVPs. What I want to do now is actually use both of these ideas, your state action rewards and your finite horizon NVPs to describe a special case of NVPs that makes very strong assumptions about the problem. But these assumptions are reasonable for many systems. With these assumptions, what we come up with, I think, are very nice and very elegant algorithms for solving even very large NVPs.

So let's talk about linear quadratic regulation. We just talked about dynamic programming for finite horizon NVPs, so just remember that algorithm. When I come back to talk about an algorithm for solving LQR problems, I'm actually going to use exactly that dynamic programming algorithm that you just saw for finite horizon NVPs. I'll be using exactly that algorithm again. So just remember that for now.

So let's talk about LQR. So I want to take these ideas and apply them to NVPs with continuous state spaces and maybe even continuous action spaces. So to specify and NVPs, I need to give you this fivetuple of state actions, [inaudible] in the reward. I'm going to use the finite horizon, capital T, rather than discounting.

So in LQR problems, I'm going to assume the following. I'm going to assume that the [inaudible] space is [inaudible] RN. And I'm going to assume, also, a continuous set of actions lie in RT. To specify the state transition probabilities, PSA, I need to tell you what the distribution of the mixed state is, given the current state and the current action. So we actually saw a little bit of this in the last lecture. I want to assume the next state, ST plus one, is going to be a linear function of the previous state, AST plus BAT plus WT – oh, excuse me. I meant to subscript that.

Where WT is Gaussian [inaudible] would mean zero and some covariance given by sigma W. Subscripts at A and B here with subscripts T to show that these matrixes could change over time. So this would be non-stationary dynamics. As a point of notation, unfortunately compiling ideas from multiple literatures, so it's sort of unfortunately that capital A denotes both a set of actions as well as a matrix.

When you see A later on, A will usually be used to denote a matrix, rather than a set of actions. So [inaudible] overload notation again, but unfortunately the notational conventions when you have research ideas in multiple research communities, often they share the same symbol. So just to be concrete, AT is a matrix that's N by N. [Inaudible] matrixes that are N by D. Just to be completely clear, right, the matrixes A and B, I'm going to assume, are fixed and known in advance. So I'm going to give you the matrixes, A and B, and I'm going to give you sigma W. Your job is to find a good policy for this NVP.

So in other words, this is my specification of the state transition probabilities. Looking ahead, we see this later, it turns out this noise term is not very important. So it turns out that the treatment of the noise term is not very important. We'll see this later. We can pretty much ignore the noise term, and we'll still do fine. This is just a warning in the sequel, what I do later, I might be slightly sloppy in my treatment of the noise term. In this very special case, it would be unimportant.

The last thing I have to specify is some horizon time, and then I also have some reward function. For LQR control, I'm going to assume that a reward function can be written as this, where UT is a matrix that's N by N. VT is a matrix that's D by D. I'll assume that UT and VT are both positive semi-definite. Are both PSD. So the fact that UT and VT are both positive semi-definite matrixes, that implies that ST transpose, UT, ST [inaudible] zero. Similarly, ST transpose are VT, AT, [inaudible] zero. So this implies that your rewards are always negative. This is a somewhat depressing NVP in which there are only costs and no positive rewards, right, because of the minus sign there.

So as a complete example for how you might want to apply this, you've seen my helicopter videos, right? So one thing is, for example, suppose you have a helicopter, and you want the state ST to be as close to zero as possible. Then you might choose UT to be equal to the identity matrix. This will make R of STAT be equal to ST transpose ST. But that's just – I'll just write that down. [Inaudible] oh, excuse me – minus. The squared negative of the squared [inaudible] vector.

So this would be penalizing the system quadratically for having a state that's half of zero, assuming that zero's the origin state. So if it goes to make a helicopter hover around the state zero, then you might choose this sort of reward function.

It turns out it's also very common for action to choose a cost for the action. So suppose I choose VT to be equal to an identity matrix. I get minus AT transpose AT here. Then minus [inaudible] actions as well. Including a quadratic cost for actions, it's also a fairly common thing to do. In practice, this tends to be effective of discouraging your system from jerking the controls around. This discourages making very huge control commands.

Having a term like this reward function often makes many systems behave better. Of course, [inaudible] choose different values, we have different values on the diagonal to give different state variables, different weight and so on. So lots of possible choices for U and B. This is one example.

So for the next few steps, I'm going to write out things, I'm going to derive things, for the general case of non-stationary dynamics. I'm going – as I write out more math and more equations for LQR, I'm going to try write it out for the fairly general case of time varied dynamics and time varied reward functions. So I'm [inaudible] function. But for purposes of understanding this material, you might want to think of just ignoring many of the subscripts, in terms of T.

So for the sake of [inaudible] material, you might want to mentally assume that there are some fixed matrix, A, so that A is equal to A1, A2, equals A3 and so on. Similarly, there's some matrix B. Okay? So write it out for the fully general, non-stationary case, but you might just want to ignore many of the time subscripts and imagine the stationary case for now.

Quite a bit later, we're going to talk about an extension of this called differential dynamic programming that will actually use a non-stationary [inaudible] to a very powerful effect for a specific algorithm. But for most of what we're about to do, just pretend that NVP is stationary.

Okay. So before I talk about models, let me jus say a couple of words about how you would go about coming up with the linear models. The key assumption in this model is that the dynamics are linear. There's also the assumption the reward function is quadratic, but let's talk about the assumption that the dynamics are linear.

ST plus one equals AST plus VAT. Maybe time varying, maybe stationary. I'm just writing stationary for now. So how do you get models like this? We actually saw one example of this already in the previous lecture. If you have an inverted pendulum system, and you want to model the inverted pendulum using a linear model like this, maybe [inaudible]. I'm not going to write that down.

One thing you could do is run your inverted pendulum, start it off in some state as zero, take some action, A0, have it get to some state, S1. Take action A1 and so on, get to some state ST. Our index is one to denote that this is my first trial.

Then you can repeat this a bunch of times. You can repeat this N times. I'm just executing actions on your physical robot. It could be a robot, it could be a chemical plant. It could be whatever. Trying out different actions in your system and watch what states it gets to.

So for the linear model to your data, and choose the parameters A and B, that minimize the quadratic error term. So this says how well does AST plus BAT predict ST plus one. So you minimize the quadratic penalty term. This would be one reasonable way to estimate the parameters of a linear dynamical system for a physical robot or a physical chemical part of whatever they may have.

Another way to come up with a linear model consistently, if I want to control, is to take a nonlinear model and to linearize it. Let me show you what I mean by that. So you can linearize a nonlinear model. So let's say you have some nonlinear model that expresses ST plus one as some function of ST and AT. In the example in the previous lecture, I said for the inverted pendulum [inaudible]. By referring to the laws of physics. It was actually by downloading off the shelf software for doing physics simulations. So if you haven't seen [inaudible] before, you can go online. You can easily find many open-source packages for simulating the physics of simple devices like these.

Download the software, type in the specifications of your robot, and it will simulate the physics that you use. There's lots of open-source software patches like that. You can just download them.

But something like that, you can now build a physics simulator that predicts the state as a function of the previous state and the previous action. So you actually come up with some function that says that – the state [inaudible] next time. The [inaudible] vector will be some function of the current state and the current action, where the action in this case is just a real number that says how hard you accelerated to the left or right.

Then you can take this nonlinear model. I actually wrote down a sample of a model in the last lecture, but in general, F would be some nonlinear function. [Inaudible] of a linear function. So what I mean by linearize is the following. So here's just a cartoon. I'll write down the math in a second.

Let's say the horizontal acces is the input state, ST, and the output state, ST plus one, as I said. Here's the function at F. So the next state, ST plus one, will be some function of the previous state, ST and the action AT. So to linearize this model, what you would do is you would choose a point. We'll call this bar T. Then you would take the derivative of this function. For the [inaudible] straight line to that function.

So this allows you to express the next state, ST plus one. You can approximate the next state, ST plus one, as this linear function of the previous state, ST. So to make this cartoon really right, the horizontal access here is really a state action pair. You're linearizing around. So this is just a cartoon. The horizontal access represents the input state and the input action.

So just to write this out in math, I'll write out the simple case first and the fully general one in a second. Suppose the horizontal access was only this state. So let's pretend interactions they [inaudible] now. ST plus one is just some function of ST, than that linear function I drew would be ST plus one. We're approximating as F prime evaluated at some point as bar T times ST times S bar T. Plus S bar T. So with this, you'd express ST plus one as a linear function of ST. Just note that S bar T is a constant. It's not a variable.

Does that make sense? S bar T is a constant. F prime of S bar T is gradient of the function F at the point S bar T. This is really just the equation of that linear function. So you can then convert this to A and B matrixes.

Jumping back one board, I'm going to point out one other thing. Let's say I look at this straight line, and I ask how well does this straight line approximate my function F, my original simulator, my original function F. Then you sort of notice that in this neighborhood, in the neighborhood of S bar, there's a pretty good approximation. It's fairly close. But then as you move further away, moving far off to the left here, it becomes a pretty terrible approximation.

So when you linearize a nonlinear model to apply LQR, one of the parameters you have to choose would be the point around which to linearize your nonlinear model. So if you expect your inverted pendulum system to spend most of its time in the vicinity of this state, then it'd be reasonable to linearize around this state because that means that the linear approximation would be a good approximation, usually, for the states that you expect [inaudible] to spend most of this time.

If conversely, you expect the system to spend most of its time at states far to the left, then this would be a terrible location to linearize. So one rule of thumb is to choose the position to linearize according to where you expect the system to spend most of its time so that the linear approximation will tend to be an accurate approximation in the vicinity of the states [inaudible]. Just to be fair, it is about choosing the point, S bar, A bar, that we'll use to come up with a linear function that we'll pretend it's a good approximation to my original nonlinear function, F.

So for an example like the inverted pendulum problem, this problem, if you expect to do pretty well in this problem, then you would expect the state to often be near the zero state. If S equals zero corresponds to X being the center of the railway track that the inverted pendulum lives on. You expect to do fairly well. You expect the pole to mostly be upright [inaudible] upright at zero degrees or 90 degrees, I guess. So you choose whatever state corresponds to having the pole upright. The zero velocity [inaudible], near zero velocity in the middle of the track. So you usually choose that as a state to linearize your inverted pendulum dynamics around. That's a region where you might want your approximation to be good.

So I wrote this down. To come back to this formula, I wrote this down for the special case of a one D state variable. If there are no actions. The general formula for the linearization approximation is ST plus one were approximate as F of S bar T. A bar T plus –

Okay? Where these upside down triangles are an unusual symbol for taking the derivative of F with respect to [inaudible] vector value, second argument. So by choosing an appropriate state as bar T, A bar T to linearize around, you'd now express ST plus one as a linear function of the current state and the current action, AT. Again, these things, S bar T, is a constant that you choose ahead of time. Same for A bar T.

Lastly, having linearized this thing, you can then convert it to matrixes like that. So the ST plus one is now a linear function of ST and AT. Questions about this?

So just one tiny detail, and it's really not a huge deal, is that this thing below is technically an [inaudible] function. There might actually be an extra constant there, but this is not a difficult generalization of a linear dynamical system definition. One way to deal with that constant is actually to do something like take your definition for this state, let's say XX dot theta theta dot. You can then augment your state vector to have an extra interceptor, one. With the interceptor one and working out the A matrix, you can then take care of the extra constant, C, as well. So you can deal with this thing being – technically it's an affine function because of this extra offset, rather than a linear function. But this is just a little bit of bookkeeping [inaudible] for yourself and shouldn't be a huge deal.

So to summarize, you see I have this up, you can learn a model, you can take a nonlinear model. Your nonlinear model can be a physics model or a nonlinear model you learned and linearize it. Now I'll post an LQR problem in which we have specification of the NVP in which the states are in RN, the actions are in RD, and the state has zero probabilities given by the [inaudible] linear equation. SD plus one equals ATST plus BTAT. Our rewards are going to be these quadratic functions.

So the specification of the NVP means that we know the A matrixes, the B matrixes, the U matrixes and the V matrixes. Our goal is to come up with a policy to maximize our finite horizon sum of rewards. So our goal is to come up with a policy, first, to maximize the expected value of this finite horizon sum of rewards.

Okay. So our approach to solving this problem will be exactly that finite horizon dynamic programming algorithm that we worked out a little earlier in this lecture. In particular, my strategy for finding the optimal policy will be to first find V star of T, the capital T, and then I'll apply by a recursion to find V star of T minus one, V star of T minus two and so on.

In the dynamic programming algorithm we worked out, V star subscript T of the state ST, this is the maximum [inaudible] actions you might take at that time of R of STAT. Again, just for the sake of understanding this material, you can probably pretend the rewards and the dynamics are actually stationary. I'll write out all these superscripts all the time [inaudible] if you're reading this for the first time.

The reward is equal to max of AT of minus – right? I hope this isn't confusing. The superscript Ts denote transposes. The lowercase Ts denote the time index capital T. So that's just a definition of my next quadratic awards. So this is clearly maximized as minus ST transpose UTST because that last term is – this is greater than or equal to zero. That gives me my assumption that VT is [inaudible] semi-definite. So the best action to take in the last time step is just the action zero.

So pi star subscript T of ST is equal to the [inaudible] of actions of that same thing. It's just zero. It's by choosing the zero action, AT transpose VTAT becomes zero, and that's how this reward is maximized. Any questions, or is something illegible?

Okay. So now let's do the dynamic programming step where my goal is given VT plus one, I want to compute VT. Given V star T plus one, I want to compute V star of T. So this is the dynamic programming step. So the DP steps I wrote down previously was this. So for the finite state case, I wrote down the following.

So this is exactly the equation I wrote down previously, and this is what I wrote down for finite states, where you have these discreet state transition probabilities, and we can sum over this discreet set of states. Now we're going to continue as an infinite state again, so this sum over state should actually become an integral. I'm going to actually skip the integral step. We'll just go ahead and write this last term here as an expectation. So this is going to be max over actions AT plus – and then this becomes and expectation over the random mixed state, ST plus one, [inaudible] from state transition probabilities given by P of STAT of V star T plus one, ST plus one. So this is the same equation written down as an expectation.

So what I need to do is given a representation of V star T plus one, I need to find V star of T. So it turns out that LQR has the following useful property. It turns out that each of these value functions can be represented as a quadratic function. So concretely, let's suppose that V star T plus one – suppose that this can be expressed as a quadratic function, written like so, where the matrix phi T plus one is an N by N matrix, and psi T plus one is just a real number.

So in other words, suppose V star T plus one is just a quadratic function of the state ST plus one. We can then show that when you do one dynamic programming step – when you plug this definition of V star T plus one into your dynamic programming step in the equation I had just now, you can show that you would get that V star T as well, will also be a quadratic function of the same form. [Inaudible] here, right? The sum-appropriate matrix, phi T and sum appropriate real number, psi of T.

So what you can do is stall off the recursion with – well, does that make sense? So what you can do is stall off the recursion as follows. So previously, we worked out that V star capital T, we said that this is minus ST transpose UTST. So we have that phi of capital T is equal to minus UT, and psi of capital T is equal to zero. Now V star T of ST is equal to ST transpose phi of T, ST plus psi of T. So you can start out the recursion this way with phi of T equals minus UT and psi of T equals zero.

Then work out what the recursion is. I won't actually do the full [inaudible]. This may be algebra, and you've actually done this sort of Gaussian expectation math a lot in your homework by now. So I won't do the full derivation. I'll just outline the one-ish G step. So in dynamic programming step, V star ST is equal to max over actions AT of the median reward.

So this was R of SA from my equation in the dynamic programming step. Then plus an expected value over the random mixed state, ST plus one, drawn from the Gaussian distribution would mean ATST plus BTAT and covariant sigma W. So what this is, this is really my specification for P of STAT. This is my state transition distribution in the LQR setting. This is my state transition distribution [inaudible] take action AT in the state ST. Then my next state is – distributed Gaussian would mean ATST plus BTAT and covariant sigma W. Then of the this state.

This, of course, is just A star T plus one of ST plus one. I hope this makes sense. This is just taking that equation I had previously in the dynamic programming step. So the V star of T, ST equals max over actions of the immediate rewards plus an expected value over the mixed state of V star of the mixed state with the clock advanced by one. So I've just plugged in all the definitions as a reward of the state [inaudible] distribution and of the value function.

Actually, could you raise your hand if this makes sense? Cool. So if you write this out and you expand the expectation – I know you've done this many times, so I won't do it – this whole thing on the right-hand side simplifies to a big quadratic function of the action, AT. So this whole thing simplifies to a big quadratic function of the action AT. We want to maximize this with respect to the actions AT. So to maximize a big quadratic function, you just take the derivatives of the functions with respect to the action AT, set the derivative equal to zero, and then you've maximized the right-hand side, with respect to the action, AT.

It turns out – I'm just going to write this expression down for completeness. You can derive it yourself at any time. It turns out if you actually maximize that thing on the right-hand side as a function of the actions, AT, you find that [inaudible] AT is going to be that times ST. Don't worry about this expression. You can get it from [inaudible] and derive it yourself.

But the key thing to note is that the optimal action, AT, is going to be some big matrix. We're going to call this thing LT times ST. In other words, the optimal action to take in this given state is going to be some linear function of the state, ST. So having done dynamic programming, you remember also when we worked out the dynamic programming algorithm for finite horizon NVPs, we said that the way you compute the optimal policy, pi star of T of ST. This is always the [inaudible] of the same thing.

[Inaudible] of actions AT of the same thing. STAT plus your expected value of [inaudible] PSTAT, P-star, T plus one, ST plus one. This thing on the right-hand side is always the same thing as the thing we maximized [inaudible]. So what this means is that when I said this a value of A to the maximize of this. So what this means is that the optimal action to take from the state of ST is actually equal to LT times ST.

What was shown is that when you're in some state, ST, the optimal action for that state is going to be some matrix, LT, which can compute, times the state, ST. In other words, the optimal action is actually a linear function of the state. I'm just going to point out, this is not a function of approximation here, right. What we did not do, we did not say, let's find the optimal linear policy. We didn't say, let's look at the optimal policy, and then we'll fit this straight line to the optimal policy. This is not about approximating the optimal policy with a straight line.

This derivation is saying that the optimal policy is a straight line. The optimal action is a linear function of the current state. Moreover, when you've worked out this is a value for AT that maximizes this thing on the right-hand side. So you take this and plug it back in to do the dynamic programming recursion. What you find is that – so you take AT and plug it back in to do the maximization. It will actually get you this formula, so V star TST.

So you find that it will indeed be a quadratic function like this of the following form where – and I just write out the equations for the sake of completeness. Don't worry too much about their forms. You can derive this yourself.

So just to summarize, don't worry too much about the forms of these equations. What we've done is written down the recursion to the expressor phi T and psi T as a function of phi T plus one and psi T plus one. So this allows you to compute the optimal value function for when the clock is at time lowercase T, as a function of the optimal value function for when the clock is at time T plus one.

So to summarize, GSELQG here's a finite horizon of – actually, just to give this equation a name as well. This recursion, in terms of the phi Ts, this is called the discrete time Bacardi equation. [Inaudible] recursion that gives you phi T in terms of phi T plus one.

So to summarize, our algorithm for finding the exact solution to finite horizon LQR problems is as follows. We initialize phi T to be equal to minus UT and psi T to be equal to zero. Then recursively, calculate phi T and psi T as a function of phi T plus one and psi T plus one with the discrete time – actually, excuse me. So recursively calculate phi T and psi T as a function of phi T plus one and psi T plus one, as I showed, using the discrete time Bacardi equation. So you do this for T equals T minus one, T minus two and so on, down to time zero.

Then you compute LT as a function of – actually, is it phi T or phi T plus one? Phi T plus one, I think. As a function of phi T plus one and psi T plus one. This is actually a function of only phi T plus one. You don't really need psi T plus one. Now you have your optimal policy. So having computed the LTs, you now have the optimal action to take in the state ST, just given by this linear equation.

How much time do I have left? Okay. Let me just say one last thing about this before I close. Maybe I'll do it next week. I think I'll do it next session instead. So it actually turns out there's one cool property about this that's kind of that is kind of subtle, but you'll find it out in the next lecture. Are there question about this before we close for today, then?

So the very cool thing about the solution of discrete time LQR problems – finite horizon LQR problems is that this is a problem in an infinite state, with a continuous state. But nonetheless, under the assumptions we made, you can prove that the value function is a quadratic function of the state. Therefore, just by computing these matrixes phi T and the real numbers psi T, you can actually exactly represent the value function, even for these infinitely large state spaces, even for continuous state spaces.

So the computation of these algorithms scales only like the cube, scales only as a polynomial in terms of the number of state variables whereas in [inaudible] dimensionality problems, with [inaudible], we had algorithms of a scale exponentially dimensional problem. Whereas LQR scales only are like the cube of the dimension of the problem. So this easily applies to problems with even very large state spaces.

So we actually often apply variations of this algorithm to some subset, to some particular subset for the things we do on our helicopter, which has high dimensional state spaces, with twelve or higher dimensions. This has worked very well for that. So it turns out there are even more things you can do with this, and I'll continue with that in the next lecture. So let's close for today, then.

[End of Audio]

Duration: 76 minutes