ConvexOptimizationII-Lecture17

Instructor (Stephen Boyd):Hey. Today we're going to do our, I guess second – second to last topic. It's one of the new ones we were very happy to add. It's stochastic model predictive control, and although in the end it's going to look just like the model predictive control, it's actually worth going over this carefully because there's actually some real subtleties here.

This is actually much more sophisticated, in fact, than mere optimization. And I'll make that clearer as we go along with it. A lot of times, the subtleties are brushed over. I think in this case it's a serious mistake because it's really – it's quite complicated, and you really have to kinda be on your toes to know what things mean and stuff like that, so – okay. Although, in the end you'll find out what do you do about it? You use, like, model predictive control that we looked at last time. So in some ways you could argue that you don't need to do model predictive control unless it's a stochastic problem, so.

So let's – I'm gonna go over this slowly because it's actually a bit – it's quite subtle, in fact. So here's what we have. We have a linear dynamical system, so with a state Xt and an input Ut. And we could make the – we could easily make this time varying so that could be AT and VT. It's easy enough. And here's the new element – is a noise here – a process noise.

So what this is is that the next state depends on actually two things – well, three things really. It depends on the current state, so that's this. It depends on your action, and it depends on this random variable. Actually, at this point, it's not yet a random variable, but it's something – the idea is it will soon be a random variable – and it depends on something that you don't know fully. That's W of t.

So we'll use this notation that capital Xt is gonna be X zero up to Xt, and that's the entire state history up to time t. So it's a big vector with all the states. It's the first part of the state trajectory here.

Now when you talk about control or something like that, we're gonna make the requirement that the input must be causal state feedback. What that means is this, that this input Ut is a function of the state history.

Now, if you're a function of the state history, then you're also a function of the disturbance history, okay, because if I know the state, if I know the state sequence – that's what this says – if I know X zero, X one up to Xt here, then up here I know that. I know that. So I subtract those. I definitely know that because I did it. So I know W of t.

So in fact – in fact, this is just a linear transformation that maps – in fact, a linear bijection that would map X zero up to Xt to X zero up to W t minus one. So to say that you are – or to use the language of probability – to say that you're measurable on capital Xt on the state sequence means you're measurable on the disturbance sequence – actually

up to t minus one because you know what the last disturbance is, but you don't know what the next one, or current one is gonna be, or something like that.

Okay, now these functions are called – that's called the policy at time t. Then I guess you'd talk about the policy would be a collection of these functions, or something like that. We're just going to consider the finite horizon one. You could easily consider the infinite horizon version.

And there's a couple of important things to say about this. The distinction in something like stochastic control is that you're actually – what you're actually looking for are these functions five t. In particular, you are not looking for a particular U. In fact, it doesn't even make any sense to say that you're looking for U. I mean, in the end, of course, what'll happen is we're going to find U. That's true. But in fact, what you're really looking for is this U that depends on the state history.

And let me just say a couple of things about how this – why this is extremely different. It's different this way, and it's very important to kinda understand these distinctions because these are not, sorta, mathematical subtleties that don't really matter much. These make all the difference. They make a huge difference in practice, and here there are.

It's this: the basic idea here is that Ut, you actually have recourse. When you choose U three, you know what X three is. You know what X two, and X one, and X zero are, and you only decide U three at that point. Okay. So that's the – in other words, you wait until information is available. What do you know at, for example, t equals three that you didn't know at t equals zero? You know a lot. You know W zero, W one, and W two. So these are things you know at that point. And that's by virtue of here, of knowing the X sequence so you know the W sequence – so you know the initial state.

So that's actually gonna have a real difference. In fact, if you choose U ahead of time knowing nothing, you will do worse. You must do worse, of course, and the difference actually is gonna be sort of the value of recourse – recourse, or feedback, or whatever you wanna call it.

Okay, so let's look at stochastic control. That works like this. The initial state and the sequence of disturbances is a random variable, though we don't assume that they're independent or anything like that yet. We may later, occasionally. So, this is a random variable. Well, it's a stochastic process, although that's a little bit pedantic, to call it a stochastic process, since we're doing finite horizon. But if you want to call it a stochastic process, go right ahead.

And our objective is going to be this. It looks exactly like it did last lecture, so it's gonna be a stage cost. That's little lt of XtUt. And then there'll be a final state cost. So this is sort of the – this is the cost of winding up, or whatever – something like that in one of these things. In a supply chain that would be some salvage cost, or some who knows what, that kind of thing. Okay.

In a dynamic system problem, this might – in fact, the whole thing might – you might be interested in putting it in some final states, in which case this would be less important, and this would judge how far you are from some target state that you wanted to hit, for example.

Okay. So [inaudible] these are convex. And what's interesting about this is that J – this objective – now actually depends on these control policies. So these are really our variables, if you wanna talk about variables in the problem. So this is an infinite dimensional problem, and in fact, in some sense it's even more than merely like an infinite dimensional control problem because your variables are actually functions. So even if you do one stage, or whatever, that has a name – if you just do one stage, your variables – I mean, if there are variables here – there are – are these actual functions or policies.

By the way, that has nothing whatsoever to do with how we're gonna implement it. They implement it be actually solving specific problems, or working out functions, and all that kind of stuff. That's a different story. Here, what's important to serve is what depends on what and when decisions are made.

Okay, and we'll have constraints. Ut is in this script Ut, and the stochastic and full problem is this: choose these policies of phi zero up to phi t minus one to minimize J, subject to the constraints. So no matter how you write it down, that is actually a very complex problem, and it's worth actually thinking about what it means.

It is not the same as an optimal control problem. An optimal control problem basically says, "You're in this state. Here's your cost. Find me a sequence of inputs." That's a straight optimization. Maybe it's large dimensional.

Even in the infinite dimensional case, which we looked at last time, it's not that big a deal. I mean, you just take a finite horizon and that'll approximate it well enough. This is much more than that because you're actually optimizing over – in fact, not only are you optimizing over sort of these really infinite dimensional function spaces, but in fact this growing set.

So, I mean, it's not that big a deal, but it's very important to understand what this – this is not – what you'll find, actually amazingly – and a lot of things is it just says the problem is to choose U to minimize J. And that's actually just, like, wrong because that's not the problem. It makes no sense to choose U. What you're actually choosing is you're choosing the method by which you choose U when you've arrived at a certain point in time, and you've made observations. That's actually what you're choosing, which is a policy.

Okay, so as I said, that's an infinite dimensional problem, and the variables are these functions, or policies. And there's many things you can do. For example, you can restrict these policies to a finite dimensional subspace. You can say all these policies are affine, for example. You can make it even more specific than that, and a lot of classical control

stuff that you learn about in, like, a control class or whatever, would be where these – they don't make a big deal about all of this, and these are very specific. You might like at some error, and have something proportional to the error, to the running sum of the errors, and maybe to the first difference. That would be a classical PID control.

By the way, if that's the type of control you're doing, where you're looking at the extremely simple policies like that, then it's pedantic to even introduce all this and talk about policies and things like that, but that's – okay – but we're gonna look at the sophisticate – the full case here.

Now, the very important point here is that you actually have recourse. Recourse means you can actually come back and do something about something after you learn more information. So that's what recourse means. It always gives you an advantage. It basically says – another name for this is, like – do I have it here? I don't. So multi – well, there's lots of names for this. One is called multistage optimization, sequential decision making. Actually, those things should be up there. Those are other names for this type of thing, and they all kinda make sense. Oh, the other one – the very big one – is just feedback, so.

I guess you're taught about this as an undergraduate. I taught undergraduate classes on this, and you give – you talk about feedback. I guess these ideas come up, but not the – not really fully what it means in an optimization context. And this is what it means.

Okay, so the point here is that you actually get to change U based on the observed state history. So if you look at a standard optimal control problem it says basically, "Here's the problem data. Please decide on what the state – what the input history should be to minimize the expected cost." That's a stochastic – sorry – that's a stochastic optimization problem. So even if there's random variables in it, that's no problem. But that's still just an optimization problem. In fact, in finite horizon one it's a finite dimensional problem. It might have some random variables and be a stochastic optimization problem, which some people would have you think is hard, but it's still just an optimization problem.

So this would be called instead of feedback or closed loop control, this would be called open loop, is what you'd call this. Okay, so now in the general case, the only way to really evaluate this objective here is gonna be here.

There's gonna be some special cases. These are quadratic, and so on. We're gonna get a sort of analytical solution, and, in fact, as usual, that will be essentially the only – that and 15 other special cases – will be the only cases where people can solve stochastic and full problems. That will be the – solve meaning like here's the solution. I mean, there'll be isolated ones in one dimension, and two, and things like that. We won't even look at them, but there are plenty of them.

Actually, you can also say the following, that if there's – I don't know – if the state dimension is one or two you can certainly solve it numerically, and get as close as you

like to the solution, and so on, and so forth, and it's very, very simple, but not in the general case.

Okay, so the only way, really, to evaluate this – except in these very special cases – is by Monte Carlo. The way you do that is you fix your policies. You have some policies, and the policies could be as simple as some affine policy. They could be open loop. You could compare it to an open loop policy. An open loop policy would just – these functions are very simple. Whenever they're called they just return a – they're returning constants. They just return some U. This returns U zero or something. That's what you're gonna do whether – no matter what you observe.

What you'll do is you'll simulate a trajectory of X zero and W, and you will calculate J, and you'll run Monte Carlo. And you can make a distribution of this cost, and then the expected value – in this case the empirical mean – will give you the objective modulo Monte Carlo error. So that's what you'll have to do to judge these.

Okay, so it's the solution, the dynamic programming, so you can actually say what the solution is here – or you can say something about the solution – and it does, in a few cases, actually just give you – it becomes sort of computable. So let's look at what it is.

It works like this. You're gonna let Vt of capital Xt be the optimal value of the objective from t on assuming – or conditioned on – the fact that you visited the states X zero up through X little t, which is to say that you have this state history capital Xt. That's your state history up to that point. So conditioned on that, you then let things play out on the other end. You would run, for example, to stimulate that you'd start from point conditioned if this is a distribution, you'd work out the conditional distribution for Wt, Wt plus one, all the way up to Wt, capital T minus one conditioned on this sequence. And then you would do simulations and calculate this optimal value of the objective.

Now, we can certainly say what this at the end. At the end it turns out that Vt, of course, there's no more action to take so there's nothing to do. In fact, there's nothing to do except pay the final bill, so you simply pay the final bill here, like that. That's deterministic. There's no – this is – in other words, this is X zero up to X capital T, and it only depends on this last component here. And if you take this back to zero, then the expected value of V zero of X zero – X zero is your random variable – this is actually exactly equal to the optimal value of the original problem.

I should say something about this before we move on, that you can do all – that you can compare all sorts of other stuff here, and there are actually two extremes that are actually worth pointing out. Let me just point those out now because it's good to keep these in mind here.

So there are two extremes. One is actually the prescient control. Prescient control basically says, "I will tell you ahead of time what the disturbance sequence is." Prescient means you know the future. "I'll tell you what the demands are gonna be, or I'll tell you what disturbances are gonna hit your vehicle," or something like that.

And if you do that, it's interested because now the – because in this case the control – it's not causal, of course. It knows the future. And if you actually solve the prescient version, it still depends on the initial state, but that, by the way, is a standard convex optimization problem, finite dimensional. You can solve it. If you run the prescient version Monte Carlo, starting from different X zeroes you will get the average prescient cost. That's clearly a lower bound here. So that's gonna be a lower bound. Everybody see what I'm saying here? If you – so –

Of course, in some cases knowledge of the future is gonna be a huge advantage. I mean, think finance for example. Even a small amount of knowledge of the future is gonna give you a huge advantage. Actually, in other cases it might not. So that's one down.

Now, at the other extreme – so that's the full prescient version – variation – which gives you a lower bound. You could also do the other one, which is the full ignorance policy.

The full ignorance policy goes like this. You can do it different ways, but the full ignorance policy is this. Choose U zero, U one, up to U capital T. Now, before you know any – any – of the Ws – see what I'm saying? By the way, that's covered by last lecture because that's nothing but – that's an optimal control problem, or what people call optimal control. That's the full ignorance one, and that will, of course, do worse than if you have recourse, and the extreme of recourse is to have full prescience. Okay. So this sort of in between here. Okay.

Okay, so let's get back to the solution here. So you have – this is the optimal value of the objective conditioned on this initial state history capital X sub t. Then you get a backward recursion, and it looks like this. It says – well, in fact, you can even say what the optimal thing to do is. In fact that's the first thing to do.

What you do is this. You have observed capital Xt. So what you do is this, is you should minimize the current cost of a current action – that's this. That's the bill you're gonna run up now. You know Xt because it's conditioned on capital Xt. You know little xt. For that matter, you know all the – the entire previous state history. And then what's gonna happen is as a result of your action V, you're gonna land in this state. AXt plus BV plus Wt. That's random because you don't know Wt yet because you don't know where you're gonna land.

Okay, so what you would – wherever you land, you'll have to finish out what you're doing. And if you do this optimally, the cost will be V t plus one of where you land here. Okay? So – and this is the state appended at the end of this thing.

So that's the cost of where you land, and this will be conditioned on – you have the expected value of this conditioned on Xt. In fact, this expected value goes outside here, but this thing is deterministic if you condition on capital Xt so it goes over here. Okay? So this is the picture. This is the so-called – I don't know – the Hamilton-Jacobi equation, the Bellman equation, the dynamic programming equation, and it's got a Russian name,

which I don't know. So, anyway, that's what it is. Does anyone know the Russian name? Pontrialigan, let's say.

Okay, so then you get this backward recursion. It's exactly the same as before, except we didn't have this conditional expectation before, so this was not here. This was simply simpler, and this was not here either. So what happened in the last lecture was this. When you chose an action, where you ended up was absolutely certain. It was simply AXT+BV, and you simply had to worry about balancing the current bill versus the bill you run up from landing in a certain state at the next step. Here it's more complicated because even when you commit to an action, you don't know where you're gonna land next step, and that's why you have this conditional expectation here.

Okay, now, you can show that these are convex functions. That's actually quite straightforward because any kind of averaging or conditional expectation is gonna preserve convexity. And this is minimizing. If you do partial minimization, you preserve convexity. So these are convex, these functions.

Okay, an important special case is when you have independent process noise. So that's when the noises are independent. By the way, that doesn't happen that often. Typically what happens is you transform the original problem so that the noises are independent. You do that by introducing additional states.

Actually, if you've taken these classes you know about this. But let me just give an example of – let's say in finance – a very common model of something would be a random lock. In a random lock, obviously the variables themselves are not independent. What's independent are the increments, are the differences. So you'd introduce a new variable, which is the running sum of the Ws. That would be a new state. So that's how this is done.

Actually, how many people have seen this somewhere? In which kind of classes? MS&E or –?

Student:[Inaudible] dynamics program.

Instructor (Stephen Boyd):Oh, okay. Perfect. Okay, great. How about –?

Student: Yeah, me too.

Instructor (Stephen Boyd):Same?

Student:MS&E 251.

Instructor (Stephen Boyd):Cool. Okay. All right. No one in [inaudible] 210? Did you take it?

Student:I took it a long time ago.

Student: Haven't taught 210 in years.

Instructor (Stephen Boyd):Oh. Well, okay. Fine, but had they taught it, this might have been in it.

Student: You see a little bit in 207 A or B, one of those.

Instructor (Stephen Boyd):B.

Student:B. Yes.

Instructor (Stephen Boyd):Okay. I thought it was in that one. Okay. Really? Years? Oh, well. All right. So, okay.

So this one actually gives you a solution in the typical case. I mean, actually, after a while you get used to this. If everything's like Gaussian – actually here you don't need anything Gaussian because it's just second order statistics, but with enough Gaussian independent arrival, exponential arrival, these kinds of assumptions of linearity, and quadratic functions, usually these problems get solvable, or something like that.

Okay, so we'll assume that here X zero, W zero – actually, let me go back because there's something I didn't – I didn't finish saying something about this, which I meant to say. Sorry.

In this case, when you have an independent process noise – again, an example of how you'd create an independent process noise in the general case – then Vt actually only depends on the current state. This is what you would – this is the way things simplify, and they look much simpler. So V sub t of capital X sub t you can write as V sub t of little xt, because that's all it depends on. And then you get this, and this is actually what you would see in books, more likely. You'd see something like that in books. And then the optimal policy would look like this.

By the way, there's an interesting thing here. The – because of this expected value, this actually has a smoothing effect on Vt. If you have a function V, even if it's sort of non-differentiable, right, [inaudible] corners, and things like that – if it's like an L one norm – you throw in an expected value, and that actually is gonna smooth it out. So these actually are gonna be quite smooth functions. That's actually a smoothing effect.

Okay, so let's do linear quadratic stochastic control, so no constraints. These variables are independent with zero mean and covariance is sigma and then Wt, something like that. That would be sort of the standard – a standard model. And then we'd take stage and final costs which are just quadratic – convex quadratic. And you don't really need this Rt to be positive definite, but it's a traditional assumption, and it makes sure – it means there's an equation that's going to come up. Well, the Bellman equation is a formula for it that just always works. The important point is that these are quadratic – these are convex and quadratic.

In this case, the value function itself is quadratic. It, by the way, is not a quadratic form. So it's not – it's a quadratic function so it's got – it's got a constant. Actually, it doesn't have the linear part. Although, if your original problem had a linear objective – a linear term of the objective, this would have a linear part, too. At the end, you have Pt is Qt. That's the final cost, and that little one is zero. And then Vt, you have a backward recursion from here. This is – this thing is V t plus one of AZ plus BV plus W. That's what this thing is.

Now, when you work this out, it's quite easy to do here because lots of things are zero mean. So, for example, the Wt transposed P times all of these two, that goes away because that's zero mean. And the only thing that really comes up here is you have the Wt transposed P t plus one W of t, and that's this. That you would recognize this way. Right, because the expected value of Wt transposed P t plus one W of t is that. Right? People have seen this before? This is the usual.

Let me just write that down just to make sure it's completely clear. It's this – it's a completely standard trig. It's sort of like if you wanna write the expected value of Z is – well, actually all I need is this. All I need is this. If I have expected value of Z equals zero, and expected value of ZZ transposed equals capital Z, then the expected value of Z transposed PZ, you write it this way. The first thing you do is you rewrite this thing as the trace of P times ZZ transposed, which looks kind of pedantic and weird here.

You do that – by the way, the way you know that this is right is that trace AB is trace BA. So I could write this as trace PZ times Z transposed is the same as trace of Z transposed times PZ. But then it's trace of a scalar, so it's the same thing. But this thing, the expectation floats inside to the ZZ transpose, you get capital Z, and you get the expected value of PZ like that.

By the way, that's sort of – people write it down that way. That's perfectly good. Everyone understands what this is, but probably there's better ways to write it. For example, I think this is probably a better way – which would be one better way. So that's another way to – that's a pretty symmetric form. So this would be a better way. It won't matter, but it's for aesthetics.

Okay, so this term Wt transposed P t plus one Wt expected value, that turns out to be this thing here, and the rest turns into a familiar Racadi recursion here because here you just get some quadratic in Z and V. You minimize the quadratic in two blocks of variables over one, and you get a sure compliment.

So this is actually a sure compliment, although you can't really see it that well. You would see it if you pulled out – I don't know – certainly if you pull out an A transpose out of either side. Actually A transpose would be fine here. You pull out an A transpose on either side, and a sure compliment will emerge here for something.

Okay, so that's the – what's nice about this is this gives a completely – this is the one case where you can actually solve a stochastic control problem. It's quite sophisticated

actually. The optimal policy is the linear state feedback. So in other words it just takes the current state, multiplies it by state feedback gain, which you get by propagating this Racadi equation – this Racadi recursion – backwards in time.

The first thing you notice about the state feedback is that it actually apparently has nothing to do with the statistics of what is gonna – of what you're up against – I mean, the disturbance, and also the initial state, which seems kinda weird. So it says basically that you do the same – you take the same recourse no matter what the – no matter what the – or the policy is the same. It doesn't depend on what kind of disturbance you're up against, or the covariance of it, which is a bit odd at first.

And the optimal cost is the expected value of V zero of X zero, but that's trace sigma zero of Q zero, but if you go back and look what Q zero is, Q zero is nothing but – you can see that Qt is just a running sum of these things. And so Q zero is simply the sum of these things, and that's – that gives you this formula here.

Now, this gives a – all of this can be interpreted quite nicely. This is exactly the optimal controls we looked at last lecture, but that was not a – that was an optimal control problem, not a stochastic control problem. It just said, "Find me the optimal U to minimize this cost," and all that sort of stuff, given what state you were in. And there was – it was a mere observation that you could write the optimal controller in a linear feedback form, and it was utterly irrelevant in the last lecture. You could give U as a trajectory. Immediately you could write it in state feedback form. They were exactly the same.

Here, that's not true. It is very important here because the point is that now you don't get – you don't know what X little xt is until time t, and that's when you make your decision as to what Ut is, and it still has its linear form.

So these actually have some very nice interpretations. Let me see if we wrote them down. I didn't. Let me tell you what the interpretations of these terms are. This one is quite beautiful. This one is it says this: it says that, "I'm going to choose at random an X zero from the distribution of X zero. I'll choose that X zero. Then after that there will be no more disturbances, none. So I'll just start you in a weird initial state, and I'll ask you to maneuver that – X zero – not quite X zero – whatever minimizes the cost running forward."

If you do that that is exactly the average costs you run up. That's what this term is. So this term is the cost you would get if the initial condition were generated according to this distribution, but there was no further disturbance of the state. In which case, by the way, you could work out what the optimal U is as a function of the initial state. So that's exactly what this term is.

This term is actually quite beautiful. It basically says that – it's the same thing, but moving forward it says, for example, at time three – or something like that – it says, "Imagine that you start from an initial state depending on – which is chosen for the

distribution W three, but thereafter, there's no more disturbance." So that would give you these. And it basically says that the sum of these gives you the optimal cost for this thing. And that's not remotely obvious at all. In fact, when you kinda look at it, and think these two are the same, it's just some weird accident having to do with linearity, and quadratic costs, and things like that. It's certainly completely false, in general, that this would be the case.

Okay, well now I can tell you what – how people – what are the methods for solving these problems. One, maybe it's the simplest one – it's not the simplest one – I mean, methods for stochastic control are – well, for one thing it includes all of traditional control like proportional plus integral – proportional integral derivative – all that stuff applies, but a simple sophisticated – the simplest of the sophisticated methods would be certainty equivalent for model predictive control.

It works like this. It's just model predictive control, but it's gonna work like this. It's time t. You have observed the state trajectory. That's utterly irrelevant up here, except in one term only. You've observed a state trajectory zero, X zero, X one up to X little t. Based on that, you form a prediction of what the future disturbances are gonna be. Is there independence? That's just like the mean, or something. It could be.

By the way, these absolutely do not have to be anything like conditional expectations. So they don't have to be. They can be conditional expectations. By the way, that's no better or worse because, in fact, the optimal thing to do is not to use conditional expectations here, but the point is they don't have to be conditional expectations. They can be. If you can calculate the conditional expectation, great.

These could actually be – these are just forecasts. I mean, they can come from analysts who say, "I think the prices are gonna look like this," or something like that. It just doesn't make any difference. So this is some prediction of what W tau is gonna be.

And so what you do is you form this optimal control problem. Now, as of – like last lecture – and you're gonna run out capital T steps – sorry – you're going to run from where you are to the end of the game. You'll run up – you'll do the accounting for your stage costs and your final cost. You'll require – you have to satisfy the input constraints. And what you'll do is for the disturbance you're simply gonna put in these forecasts or approximations. You're just gonna plug in the forecast.

It's really, like, quite stupid. It basically says – it says, "Do a planning exercise. Do exactly what you would do if your forecast were perfect." Right, which is quite ridiculous. So, but that's what it is.

So the only way in which capital Xt comes into this – and in fact, this whole thing is actually – this is the policy, the Arg min of this subject to that – that's a policy, which is to say it's a function of capital Xt, and it comes in right through there and nowhere else. So you solve this problem.

Okay, so you called it – this is your plan, and it's a bit silly. These are all – actually, not one of these will actually come true. I mean, this is sort of your – if someone says, "What's X little t plus one out to the end?" You say, "Well, that's my plan for what the state trajectory is going to do." You have no reason to believe that's actually what's gonna happen. In fact, that's not what's gonna happen. That's all quite – in fact, if that's what's gonna happen, that means your forecasts are perfect, and this is the wrong lecture for you. You have to look at the previous one, which is simpler because then it's not stochastic control.

Okay, so you work this out. Then what you're gonna do is you're simply gonna do this. And then this is – that's your policy. That is the certainty equivalent model predictive control. That's your policy.

So it looks dumb. It works actually shockingly well in a lot of cases. It sort of balances two – it makes two simplifying assumptions which kinda cancel – null each other out. One is that it assumes your forecasts are perfect. The second is that you'll have no recourse in the future. Actually, if your forecasts are perfect, there's no need for recourse. So this combination of two weird, bad assumptions kind of ends up doing something quite good.

Okay, so this is widely used, for example in revenue management. This is a euphemism for extracting as much money as possible from you and me, like the airlines use, or something like that. They decide how many seats to release at some price, and that kind of thing. So that's revenue management. I like that term. They're definitely managing the revenue. And I'll give you a hint. It's not in our favor.

It's used actually in all sorts – it's also used in finance for a lot of things, for things like optimal execution and all sorts of other stuff, when you have to sell a giant block of asset, or pick up a block – a big block – some big block of stock, or something like that. So it's used in lots and lots of things. In fact, it will be used in more and more. It's – right now it's used in things where things go pretty slowly. I mean, where it's down to a second, or even minutes, or something like that – like in supply chain optimization.

In real supply chain optimization, that's stochastic control. Unless your forecast says, "I'm gonna tell you the demand for this product in three weeks," and then you say, "Well, how sure are you?" and then the forecaster says, "Totally," then you're in last lecture.

By the way, the funny part is you'd do the same thing. The same code would run. So it's really just subtleties here. The difference is in that case it's just a cheap way of just looking ahead and getting out of solving a big problem, or something like that, which is actually kind of silly given that the cost of solving problems like this scales linearly with the horizon size.

Okay, so it's used in revenue management. As I said, it's already – it's based on two very bad approximations that kind of cancel each other out in a weird way. So you trust your

forecast perfectly, and you have no recourse. Actually, if you trust your forecast perfectly, there's no need for any recourse because it means that the future is completely known, and therefore you can decide. There's no advantage to changing how much of some product you order next Tuesday based on a week's worth of new data because your forecast is perfect, so there's no need to. And it works really, really well.

By the way, combined with the stuff we looked at last week, which tells you that these things can actually run very faster – faster by, let's say, a factor of 1,000, or 100, than anyone – than almost all people know or suspect. It tells you that there's actually plenty of opportunities for this stuff to be fielded in various things because not many people – I mean, there are places using this. One of the problems is when you looked around and said, "People don't use this name."

The only name – the only people that would recognize model control would maybe be people running chemical process plants, and you said we're using – that's how you solve your stochastic control problem, they would say, "What's that?" And if you looked in finance or revenue management, they would say, "I'm doing revenue management," or, "I'm doing dynamic linear programming," is another name for it, or something like that. So you get – okay.

So let's look at an example. It's a system with three states, two inputs – I mean, none of this matters – a horizon of 50. It's a completely random system. You'll have a quadratic stage cost. This is just – the point is just made up completely just to see what happens – and the constraint set is – it says that each set, that each input has to be between plus or minus point five. Then we'll just make all of these inputs initial state. It's all – they're all just IID with a standard deviation of point five. They're all plus or minus point five variables, all of them.

Okay, so here's a sample trajectory. A sample trace of X one and U one, and this shows you the state, and here's the input. You can see that it is actually saturated some fraction of the time here. You can see various times where it hits the limit like this. And this is running – this is with the MPC trajectory.

Now here, the way you'd get this is you'd generate a random X zero, W trajectory. You'd then actually run MPC, which is to say that you'd solve a QP for every one of these points, and you'd propagate it forward, and get – by the way, the – what's our forecast for our Ws? What's W hat of t given tau – tau given t? What is it?

Student: It's zero.

Instructor (Stephen Boyd):It's zero. Yeah, it's zero because they're independent, and yeah. By the way, if for some reason you don't want it to be zero, you're perfectly welcome to make it non-zero. If you wanna add a hunch in there, you're welcome to. But we're using zero.

So the way you judge this now is it's – unlike last lecture you can't just run it and say, "Here's the objective value." You actually have to do this by Monte Carlo. There's no other way. Unless – I mean, if there are no constraints and it's – then you get a formula, that's the trivial case. In a case like this, just with – all it takes is input constraints to make this – there's no analytical solution in this case. There are a few problems with analytical solutions, but –

And so you run Monte Carlo simulations. Each Monte Carlo simulation requires you to solve 50 QPs in this case, or whatever. So you run – I don't know –a couple hundred or thousand Monte Carlo simulations. In fact, that looks like 1,000. Is that the right number? Is 1,000?

Student:[Inaudible].

Instructor (Stephen Boyd):Really, because that's 100 right there. Of course, these numbers are totally irrelevant, right? But if you add up all these you'd find out – you might be right. That's 500?

Student:A thousand.

Instructor (Stephen Boyd):It doesn't matter, 500 or 1,000. Anyway, it's some reasonably appropriate number – and the average value is here. So of course this objective is a random variable. It's got some outliers. It's got some things it did very well here because you've got a very benign input disturbance here.

By the way, as the horizon gets longer, of course, this gets tighter. It doesn't matter. We're just doing the 50-step version. So this is – I'll explain what these two are in just a minute. In fact, I'll explain what all these things are. In fact, let me do that first, and I'll come back.

So here are – several control laws are gonna be checked. We're gonna do this. The stupid control law is to calculate the linear quadratic one. By the way, this is how this would sort of probably work if people implemented it in a dynamical system or something like that. You'd have the state feedback gain, which would tell you U is something or other. If there were no constraints on the input, that would actually – that linear feedback control law would be exactly optimal, period.

Now the problem is when you form KtXt, occasionally that's outside the bounds of your input, and so you simply saturate that. So that would be this one, and this gives you an average cost of 275.

If you run MPC here, your average is 224. So you can see it's actually measurably different. I think you can see that from this picture here. This is this very, very simple thing, and this is MPC here. So that's a little disturbing, but that's okay. That's the nature of all this.

Actually, if you cared about this – suppose someone said, “I don’t like all of that compared to that,” what’s the answer to that? How do you respond if someone says that? There is a correct response, and there’s a lot of incorrect responses. What’s a correct one? What’s the correct one, actually?

Well, the correct one goes like this. When this problem started, you agreed to this tossed bunch, the expected value of the cost worked out with this. If, for some reason, you don’t like it when this thing has – when that cost has outliers that are big – then you should not have agreed to that cost function. And, in fact, you shouldn’t have done something like this. You could put a risk averse term, and how would you put risk aversion into this?

We talked about it last time, actually. I think we did. How would you put risk aversion? Just suppose you look at this, and someone says, “Yeah, I’ve run MPC,” and they go, “That’s terrible. There’re lots of times when the cost is high. The stupid saturated controller does better, actually.” What would you do about it? I think we talked about it.

Student: Have like, standard deviation –

Instructor (Stephen Boyd):You have standard deviation; you know how you’d do that in a convex way?

Student:X transpose covariance X.

Instructor (Stephen Boyd):No, that won’t do it. That won’t – because you want – what you wanna do is you wanna penalize large excursions of Lt – L little t and L big T. You wanna penalize large ones, and you’re less worried about the smaller ones, right? It’d be like what you do in finance, or something like that. What would you do?

Student:[Inaudible]

Instructor (Stephen Boyd):Huber would be much worse. If I replaced those quadratics with Huber, I’m basically saying, “I’m giving you license to have large residuals,” and that would do just the other thing. But you’re close, actually. What would be better than that? What’s the opposite of Huber?

Huber says, “Relax on the outliers.” Right, because it just goes linear. That’s the most relaxed you can be and still be convex in a penalty function. Barrier. Barrier would do the trick.

By the way, if I did a Barrier here, what would happen? Actually, can I do a Barrier here? It’s sort of a subtlety, and it would be a practical issue. You cannot because the disturbance is Gaussian, and so with small probability, however big you set your barrier, there’s a positive probability that the disturbance – through nothing you can do about it – would be outside the barrier and run up a bill of infinity.

You can't do Barrier, but you can add, like, a quartic. But I'll show you the right way to do it, in fact. The correct way to do it is to do risk aversion. And you would simply do this. You'd have this thing. I'll – it doesn't matter. Let me just do this one. You don't like excursions in X, then you'd do this. You do – this is your new – that's your new cost. Okay? That's your new cost.

By the way, it does the right thing. If gamma is really small, then this is basically like that. But what this does is it penalizes positive variations more than negative ones. Everybody – this does the right thing, and when you put the expected value here, that's your new cost function. It does just the right thing, and it's very traditional to do this like that. And this would be a cost function that is now risk – it would be called a risk averse quadratic – or exponential of quadratic, and whatever. So that's what you would do.

By the way, the first term in the expansion is what, I think – I can't remember who suggested it – what you wanted. The first term in the expansion here is the variance. So you would get this cost plus the variance. And if you were to do this, it would tighten this up very, very easily. How easily could we do this in –? Well, I believe – is the source code online?

Student:[Inaudible].

Instructor (Stephen Boyd):Not yet. If the source code – how fast could we do this, could we add this, could we modify this – the source code?

Student: Very fast.

Instructor (Stephen Boyd):Very fast, right. You would just go back in, and wherever there's – wherever you see the – you find, you search, you grab for minimum – you find the minimized statement, and you replace it with this thing, and it's done. Right.

So the incorrect answer is to complain about these things. That's wrong. By the way, the same is true, but would actually never happen in finance if you said, "I wanna do something," and then you work it out, and then look at the histogram, and there's some outliers, like in this case negative ones meaning loss, or something like that. You don't like those? No problem. Then go back and put in a nice, strong concave function that charges you for losses. You don't like losses. And don't complain about it. That's the key.

Okay, this is all an aside. So I guess the point here is that when you started the problem, we agreed it's the average of these that matters, and this beats this quite a bit. This relaxed number – let me show you what that is. The relaxed number is simply this: it basically does the following. It ignores the input constraint. So it's simply – it's calculated via the LQR – via the linear quadratic regulator. So it just propagates backwards, and it actually runs up wherever this formula was. It just calculates this formula. That's not by Monte Carlo. That's just – you run through – add up all these guys, and then that's the exact amount.

That's obviously a lower bound because here the difference between the two is merely that you are more constrained. It simply corresponds to commenting out this constraint, and then you get the global solution because it's one of those special cases. And that says that no one could possibly do better than this here.

And what this gap shows you is that – well, it kinda says that you're kinda close. By the way, a more sophisticated relaxation method will make this bound for this instance – what is it – like 175 is a more sophisticated bound. You get 175.

Actually, at that point it's quite interesting. I would be happy to argue at that point that for all practical purposes this is pretty good. Actually, this would be for the quadratic case. And the reason I would say that is this: if that's 175, and that's 225, you're off by 50, or something like that, but you have to take the square root because these are sorta like squares. So you're maybe something like – at that point you could argue that you're no more than 15 percent suboptimal, period. That's it.

In this case, for problems like this when you see objectives that look like this – this would be typical in dynamic systems with vehicles, or traditional control. Usually, people don't really mean this. They just mean, "I want this." This is a code word for, "Please make the state small, not big." Right, because no one really cares about least squares type things.

If this was a more intricate function in a finance problem, or something like that, or a revenue management problem, then things being like 20 percent within the global optimum is not – then 5 percent means something. So when you see an objective function that's likely to actually be the revenue, or the loss, or the cost, or something like that, then you can't be so cavalier about it. But when you see things like this, it's just somebody saying, "I'd like the state small."

And you can go back and ask questions about that. Like you can say, "Are you sure you really meant this? Are you sure you didn't mean one point one here, or something like that?" You can ask a couple of key questions, and they'll maybe admit it.

Okay, so this finishes up this topic. It's quite interesting. I think there's a lot of, sort of, opportunities for doing this in a whole bunch of different areas. I guess people are already doing it, maybe not in a totally organized way, but in lots – people in lots of fields have already found that. Maybe you'll find it in some others.

Okay, so any questions about this? Otherwise, we'll go on to our – we'll start our last topic.

Let's see. How do you do this?

Okay, so now we're going to start the last topic for the class, which is on global – it's the very – it's sort of your intro lecture to global optimization. That's, by the way, a huge field. You can and do have whole classes on it. I don't think here, but you can have

whole classes on this. People do have whole classes. There's like books, and books, and books written on this.

So this is going to be the simplest, simplest, simplest method just to show you what these methods look like – the simplest methods of what it looks like to solve a non-convex problem noneuristically, to be able to say that's the global solution within this, or something. That's branch in bound.

So let's go over how that looks. So just to remind you how this works, we've already looked at this earlier when we looked at sequential convex programming. The basic idea is if you have a convex problem, then methods – and you – it's some standard problem – if you solve it you always get the global solution. That's not an issue. And it's always fast. That's sort of true both in theory and in practice.

When you have a non-convex problem, you have to give up one of these. That's what you have to do. Now, if you give up the always global – you're talking about local methods. We've already looked at one, which is sequential – or a family of local methods based on sequential convex programming. That's – these are local methods. There are lots and lots of those that you can take classes on here, entire classes.

In global methods, what you're gonna do is we're not gonna drop this. So you're always gonna get the global solution, and you're gonna lose this. They do actually, every now and then, in some application they'll actually work quite well.

Okay, so we'll look at global optimization methods. They'll find the global solution, and they'll certify it, but they're not always fast. In fact, in many cases they're slow.

I should say a few things about this. Just to remind you, in convex optimization there's the idea – there's the certificate. A certificate of optimality, or sub optimality, or something like that is provided by a dual variable. So you solve the problem. I mean, after a while you get used to it. You just solve the problem, and you trust it. You solve an LP, and you get the answer, and you don't check to see if, like, STPT three or sudumi calculated the right solution for you. And it wouldn't have come back with a status of solved, or whatever, in CVX had it not certified that to some reasonable number of digits. It actually calculated an exact – a solution within that tolerance.

So the certificates in convex optimization are very simple. They are dual feasible points. So when I give you back a solution, I say, "Here's the solution." And the simplest method of certification goes like this. I'll give you a point which is suboptimal. You verify that it satisfies the constraints. I also give you a dual feasible point. You verify that it satisfies the dual constraints, and you evaluate the objective of the first one, and the dual objective of the second, and if they're close, then it means that – in fact, it simultaneously certifies both. So certificates in convex optimization are based on duality.

What's gonna be really interesting is what does a certificate look like for a general, non-convex problem. And there's different kinds of certificates, but we'll see what they are. It's a more complicated object.

Okay, so branch in bound. It's a family of global optimization methods. You'll find lots of variations on this. There's like branch and cut. They've got all sorts – that's another huge name for this, and they have all sorts of names for this.

Actually, how many people have actually taken something on this, or seen –? No one. Okay.

Okay, so they're gonna provide – they're gonna maintain provable lower and upper bounds on the global objective value, just like in convex optimization. So you will get plots that look very familiar. You could even show something called the gap. It's not the duality gap, which you're used to seeing the duality gap versus iteration. It'll be a gap, and the gap will be – the gap between the best point found so far, and the best lower bound found so far. That's gonna be the gap at any given time.

Now, these have exponential worst-case performance, and in fact, they will often be very slow. Actually, occasionally, if you're very lucky, they work well.

I should add that I branch and bound algorithms, you're gonna see like – in solving convex optimization problems, there's not much room for personal expression. I mean, not much. I mean, I can solve it faster than you can, or something. I mean, there is, but it's kind of very small. You get to non-convex problems like local methods, huge room for personal expression.

By the way, that's a way to say these things are totally unreliable, and need to be babysat, and all that kind of stuff. But if you wanna think of it on the positive side, it means that my local method can be way better than yours, which just is weird.

And then you judge it in many ways. You judge it by how fast it is. Also, it's entirely possible in a local method that we come up with different solutions. Sometimes yours is better. Sometimes mine is better. If mine is often better, then mine is better. And I can always, when I'm doing this, choose the examples where mine is better. So that's why I'm saying it's nice for – you get nice personal expression.

You get personal expression in branch and bound method, too. We'll see where the personal – where the ability for you to put your stamp on this comes in.

By the way, that work will be simple. What happens now is we can't disagree. No one can do better than another person. If the rule is to solve the problem globally, to certify 100 percent accuracy, then you can't do that any better than I can do it because we're both doing it, and there's no lying and all that. We have to do that.

What can change is how long it takes, and indeed, your method of branching, or whatever, your bounding method could take four seconds, and mine might take two days to run, or something. That's how you would judge yourselves, is basically just how long it takes to run.

Okay, so the basic idea actually is very cool and very simple. Actually, the basic idea is quite useful in a lot of cases. In fact, I'll – we already assigned the last homework, didn't we? Hm. Do we have recourse? I would say so. We'll see. I'll think about it. I just thought it was good – there was a problem I've been thinking of that I wanted to assign. I even wrote some of it down somewhere. I'm gonna find out. So we'll see. That will be recourse in action. It'll be simple.

Okay, the basic idea is it relies on two sub routines that hopefully efficiently compute a lower and upper bound on the optimal value over a region. So that's what it is.

Now, how do you find an upper bound on a minimization problem over a region? That's easy. You could choose any point in the region and evaluate the objective function – I mean, if it's feasible. Well, if it's not feasible the upper pound is plus infinity, which is a valid upper bound. It's just not – well, it's not useful, but it's valid. You could run a local optimization method. It really doesn't matter.

The lower bound of the region you're gonna find by convex relaxation, duality, you could argue from Lipschitz or other bounds, or something like that. The way this whole thing's gonna go together is this. You're gonna partition the feasible set into convex sets, and then you're gonna find lower and upper bounds on each in the partition. From the partition you will then form a lower and upper bound on the global problem. And if they're close enough you quit; otherwise, you will partition and repeat.

Actually, before we go on, let me just write down the example, and I'll show you what it is. Maybe we won't put it on the homework. How about that? But I'll find this problem. Met with approval? I guess so.

All right. Let's just solve a problem like this. Let's minimize. Note the period. C transpose X subject to AX less than B. Then let's do something like this. Let's say that X one is either zero or one. Okay? So that's my problem. How do you solve that?

First of all, is it convex? No, not remotely. I guess the classical method is to – when you see that it's not convex, you back off slowly. Do not turn your back. You slowly back away, and then make a safe retreat. But, actually, how do you solve that? It's extremely easy to solve. How do you solve that problem?

Student: Plug in zero [inaudible] solve it.

Instructor (Stephen Boyd):Yeah, you'd plug in zero and you solve it, and you plug in one and solve it, and it's done.

Okay, so how about this one? How about that one? How do you solve that?

Student: Relax it, and hope [inaudible].

Instructor (Stephen Boyd):Yeah, that's – I mean, methods that don't involve prayer, or hope, or aspirations. But that would work. That would give you lower bound. What would you do for this one?

Student: Do all the combinations.

Instructor (Stephen Boyd):You could do all combinations if you had to. This is 1,000. You'd run 1,024, right? So you solve 1,000 – what would the answer be, by the way?

Student:[Inaudible].

Instructor (Stephen Boyd):What would it be? How do you get the answer? You'd run 1,024 LPs, and you just take the best one. Okay, so that's what you'd do. So branch and bound is gonna go something like this. This is sort of the basic – it's basically this idea, but doing it in an organized way where you're gonna scale it past ten. You can go to 15 this way, and so on.

By the way, just for fun – and then we'll get to the main thing next time – let me ask one more question about this. Suppose you had to solve that problem right there. You had to solve 1,024 LPs, but you had to do it fast. It was taking too long, your whatever – 15 line matlafting was taking too long. How would you – could you speed that up, solving 1,024 LPs like this?

Heck, let's talk about that. Could you?

Student:[Inaudible].

Instructor (Stephen Boyd):Oh, you did it in parallel. That's probably the easiest way to speed it up, is in indeed to find 1,023 friends.

Student:[Inaudible], and then kinda figure out which ones would be zero, and then kind of fix –

Instructor (Stephen Boyd):Oh, yeah. You could get euristic real fast by relaxing. You could also relax and get a bunch of them. By the way, if you relaxed, and some of these were slammed up against zero, and some against one, what can you conclude?

When you solve in relaxation, if they were all zero or one, what could you conclude?

Student: That's the optimal.

Instructor (Stephen Boyd):That's the optimal. If they were all zero or one, but three of them were just sitting in the middle, what could you conclude?

Student: Nothing.

Instructor (Stephen Boyd):That's the right answer: nothing. It is very likely that the ones that in the relaxation came out zero are indeed zero in the global solution, but it is absolutely not guaranteed. That would be one method.

Actually, in something like this, you could do a lot of stuff. Let's – we'll just mention a couple. First of all, you can do warm start. Whenever you have to solve a problem with the same structure over, and over, and over again, with nearly the same – not nearly – I guess it's not that near when you're taking the first five variables, and making them zero or one. But if there's 1,000 variables there you could use a warm start from the previous one.

Here would be another trick that would actually be quite useful. Suppose you've solved a bunch of these LPs. Your code is gonna have to have a logic in it that keeps track of the best one you've found so far, right? So you're gonna have – you're gonna maintain something like X best. And it remembers simply – it remembers some pattern of these, and then it solved it, and it's the best one. The others, by the way, are out. Anybody worse than X best is out of the running. Okay?

When you're actually now solving this problem using an interior point method, you get at each iteration a dual value. That's a lower bound. Everybody see? Right? You put logic in there that says the following: if you take a pattern like zero, zero, one, one, zero, whatever it is, something or other – you take that pattern, and you start solving it, the instant the dual value goes over this, what can you do? You can quit even though you haven't solved that one of your 1,024 LPs.

Why can you quit? Because by convex analysis, you have a proof or certificate that the optimal value of that LP exceeds this. But actually, that means, you don't even have to finish solving it to know that the answer will be worse than the best you've already seen. Therefore, you break. Everybody see this?

That will work really well because a bunch of these are gonna be way, way – it's gonna be obvious, right? It won't take the dual value very long to get above the best you've seen so far.

Student: Does that mean go above C transpose X best, or will it be –?

Instructor (Stephen Boyd):Absolutely, sure, because C transpose X best, that's the best you've found so far, right? Now you choose a different initial bit pattern. This was obtained with zero, one, one, one, zero – something like that. Now you're testing this pattern: one, zero, one, one – who knows. Like that. What happens is the optimal value of this thing may be like over – actually, if the optimal value of this is lower than this, then

that’s the new winner. That’s the new X best. But if it’s above it, then the dual value will rise above this value, at which point you break.

Now, if it’s way above it – which is to say that this is a rather stupid, obviously stupid choice of initial bits here, then the dual value will go up, and be two iterations, and it’s all over. These are just practical matters, but they would allow you to do this a lot faster than it looks like it would have to be if you just did – if you didn’t do it, if you just wrote a five line thing that said, “For all 1,024 patters, evaluate the – solve the LP, and compare the optimal costs.”

So, I guess we’ll – how did that happen? Well, we’ll quit here, and we’ll continue this as our last topic next time.

[End of Audio]

Duration: 77 minutes