

## ConvexOptimizationII-Lecture16

**Instructor (Stephen Boyd):** That's okay though. I see it puts a nice emphasis on, I mean, it's probably okay to say here's our problem, actually, it's an approximation. Here's the relaxation. Then when we run the relaxation, of course, on the real thing you get one thing, but then – actually, when you run it on the true system the relaxation thing does better than the other one. That's not uncommon.

**Student:** Yeah.

**Instructor (Stephen Boyd):** Perfectly okay.

**Student:** Yeah.

**Student:** It's Kreloff.

**Instructor (Stephen Boyd):** It's what?

**Student:** Kreloff.

**Instructor (Stephen Boyd):** Kreloff. I just spent five days with a bunch of Russians. I guess I'll have to remember that for next year. The left monitor's going to be right to you that I'm on. Does that mean I'm on? Something about this room, I don't know. At least I haven't said anything too horrible while being taped or whatever. Okay.

Today we're jumping around a different order of topics, but this is maybe, I think, the next topic that some people are working on, it's obviously too late for them for their projects, but we can at least cover the material and for people who are doing this at least it'll make a lot of sense. For other people, it's actually very, very good stuff to know about. It's widely, widely used. So it's called Model Predictive Control. In fact, I've been reading a lot about it the last couple of days. To sit through very long airplane flights, read a couple more books on it. It has got tons of different names, all different. Basically all the different areas doing this don't know about the others.

Often the notation and stuff is absolutely horrible. So, anyway, here's the basic idea. We'll get this in a minute. It's basically, I mean, there's a bigger area. Model Predictive Control is the name that comes from a relatively small community. In fact, I read one book and heard, I guess from some sort of member of this community, as referred to as the control community. That was as opposed to some others. Other names would be used. One that is a very good name in a sense is embedded optimization, real time optimization. Actually, the way that fits in the future, I think, is very, very interesting. A lot of people have their view of optimization locked into a model that's from 1969.

Either because they learned it in 1969 or they learned it from someone who learned it in 1969. They think of optimization as this big, complicated, very big thing that requires several Ph.D.'s, a bunch of super computers, and things like that. You really wouldn't use

optimization except maybe to, I don't know, design a nuclear weapon, design the newest airplane, schedule United Airlines for tomorrow, or who knows what else. But these are the kinds of things you – or price derivative in the basement of Goldman Sachs. These are the things people think of when you think of optimization. You all ready know that a lot of these things actually are extremely reliable.

They're reliable enough to just work always, period, end of story. They're also very, very fast. By the time you say fast and reliable then it means you're automatically talking something candidate for imbedding in real-time systems, right? Because these, in fact, are the only two qualities you need to imbed in a real-time system. It has something that's gotta be fast, have bounded run time, and it has got to be extremely reliable. It can't require a Ph.D. to come over and say, "Oh, your algorithm's jammed. Let me change the number here. Let me restart it from a different point." You can do that if you're doing some big, big problem where it's important to run. That's fine. But if you're making a decision every 2.6 milliseconds you obviously can't do that.

The bigger name for this lecture would be imbedded or real-time optimization. Okay. So let's start. I think it's a very interesting area. I think you're gonna see a whole lot of it more in the future. Let's start with an optimal control problem and there's some very nice subtleties here and it's quite complicated. The idea, although, unfortunately, when it's taught many times the subtleties just aren't seen, but they're actually very important subtleties. So I'm gonna go over them a little bit because it actually takes some time to really understand what it means. So here's what we're gonna have. We're gonna have a linear dynamical system, so that's  $x(t+1) = Ax(t) + Bu(t)$  and this  $u$  is gonna be our input or control or action.

We start from a state  $z$ . These input and controls have to be in some set  $U$ . The  $X$ 's have to align some set script  $X$  and our objective is a sum over the entire trajectory of functions, this is called the stage cost or something like that, of this state and the control. Now, this is very traditional and you'd find this in like a vehicle control if you've taken a course on control or something like that, but, in fact, this describes tons and tons of problems, including supply chain management, things like scheduling and networking in networks and computer system. This just goes on and on and on. A lot of problems in finance look to be exactly like this. I mean, some don't, but many do. Even though I'm using the traditional symbols that you would find, for example, in the course in AeroAstro describing vehicle control, this can just as well be a finance or economic system or something else or a supply chain system.

So don't let the notation of the, as this book I was reading said, the control community fool you. This is much more general than controlling your local robot or something. Actually, it doesn't actually include your robot because the dynamics are linear. Okay. So the variables are an infinite number of inputs and we can also treat the states as variables too. In which case, we think of these as equality constraints. Now, the other option is to not think of the states as variables and to think of this as merely a recursion, which relates the  $X$ 's, which are expressions, and, in fact, affine expressions to the variables, but these are actually equivalent. Okay.

Now, the data to the problem are this dynamic matrix  $A$ , this input matrix  $B$ , this stage cost function, and we'll assume that it's zero at zero. So, in other words, if you're at zero that's some costless point or something like that. We'll assume that these are convex and they include zero. You can actually change all of these assumptions pretty much and it just makes things more complicated. Okay. So the first thing one can have is something called greedy control and it's sort of the most obvious thing. It works like this.

What you do is you simply pick the input  $U$  to minimize the current stage cost here, that's what this does, over your allowed control actions and the only part of the future you intend to take into account is the following: Because of your action  $W$ , the next state is gonna be  $Ax(t) + Bw$  and the only accounting of the future you'll take care of is that this must be in this state constraint set, so that's Greedy Control. The basic idea is that it minimizes the current stage cost and it ignores the effect on the future. Well, except for this. It says that you can't do anything that would get you into serious trouble, namely violating the state constraint, on the next step.

As to whether or not you have violated the next step, that's entirely possible and too bad. Now, this method typically works very poorly. Of course, we can make it work very well, no problem. If  $A$  here, for example, has a very small spectral radius or very small norm it means that the  $X$  at the next state doesn't depend much on the current state and then that tells you that basically the dynamic system has what people would call something like strongly fading memory. That means that, in fact, Greedy is not too far from optimal because although your current action does have an affect it propagates into the future the affect goes down fast. For example, if the norm of  $A$  is point one that would certainly be the case here. The Greedy Control would probably work well. Okay.

If you haven't seen optimal control in dynamic programming – actually, how many people have seen this somewhere? Good, that's good. That's how many people should have seen it. I mean, this is sort of like 48 transforms. You need to see it maybe three times minimum, maybe four. Maybe it's more important actually, but I guess that's open for argument or something like that. You should see it three or four times. So this will be your, whatever, third or fourth time for some of you. If it's your first time that's fine too. Here's the way it works. You define  $V(z)$  as the optimal value of this problem right here. It is the optimal value of this problem. That's an infinite dimensional problem.

The variables here are the  $U$ 's and the  $X$ 's. It is infinite dimensional. It has an optimal value and it's a function of  $Z$ . That's an infinite dimensional one, but the optimal value of a convex optimization problem as a function of the right-hand sides of inequalities or equalities is convex in those parameters. The optimal value of this function is a convex function of  $Z$ . By the way, you know that rule very well. You know that rule in specific context, for example, the minimum of a quadratic function over some variables is another convex quadratic function. It's given by the sure compliment. So that's a very specific case where you can say much more about it.

In general, you know it's just true. Okay. So we formed this function  $V$  and that's called a value function. I should say that, there seems to be a weird – once again, MIT and other

places split in typical notation. So some people use  $J$  as the value of the – I'll have to hunt down the source of this.  $J$  is the objective and then a lot of people use then  $V$  for the value function and at MIT it's reversed. They don't use  $V$ , this is  $J$ . So if you look in books from MIT or something the value function is called  $J$ . However, it is always called the value function or the Bellman function or, I forget what the Russian name for it is, but I think the Russian symbol is  $V$ , so anyway. Just a comment on notation.

This is convex. It's actually extremely easy just to show it directly. The important part is it satisfies Bellman or dynamic programming or Hamilton-Jacobi equation, many other names, Punctreargon equation. It says this, it's actually quite interesting. I'm not gonna go into a lot of the details, but it makes perfect sense and, in fact, it's extremely easy to show. It looks deep. A lot of people want you to think it's very complicated and deep and it's actually not. It basically says the following: It says that if you want to make the right choice right now from state  $Z$  here's what you do. Well, when you take an action  $Z$  you're gonna run up the bill  $l(z,w)$ . That's immediate.

So the extremely greedy control, would say this: Choose  $W$  to minimize this. That would be the extremely greedy control. Well, that's not good because you have to at least take an action that will land you in an allowed state next step. That's the first plausible greedy method. As a result of choosing  $W$ , here's where you land:  $A(z) + B(w)$ . So far we've all ready said that's feasible, that's okay. But the question is how should you take into account the affect of your current action on where you land? Well, in fact, if you imagine where you land, if you then simply carried out the optimal control from them it couldn't get any better than that.

It turns out this is what you should do. You should add to the current cost, the current bill you run up, plus the bill you would run up for landing in this state and thereafter doing exactly the right thing. So that's what this says. It says if you add these two together and minimize over  $W$  then this will actually equal  $V$ . It's the optimal thing and not only that, but the argmin of this thing is the optimal  $U$ , right? It looks complicated, it's really not. If you take an infimum, you can minimize in any order and it doesn't make any difference, right? So that's really what it is.

So this is the dynamic programming equation. By itself it says absolutely nothing, doesn't help in the slightest because then you'd say so what's  $V$ ? What is interesting is that in some cases you can actually find  $V$ . Once you've found  $V$  you actually have the solution here. And there's one thing I want to point out about this, this optimal input is, in fact, a function of the state  $x(t)$  because although it's complicated that's a function of the current state. So that's what it is. In other words, that the optimal input has to form some function  $\phi$  of  $x(t)$ . State feedback form would be the common way to say that. That's quite interesting.

And here, as I said, the interpretation is this: That this term  $V(Ax(t) + Bw)$  is very important. It's basically the accounting term needed to completely accurately account for the effect on the future cost of your current action, so that's exactly what this thing is here. Okay. So one famous case where this actually leads to a numerical solution, but

basically a solution, is this: You have linear quadratic optimal control, so the stage cost is  $X^T Q X$  plus  $u^T R u$ . It's easy to add a cross term between  $X$  and  $u$  and it's easy to add a linear term and all that kind of stuff and this is just like exercises.  $Q$  is traditionally positive, semi definite and  $R$  is positive, definite. That's just to keep the formulas simple. Okay.

Now, in this case you can actually solve the problem by dynamic programming. It turns out the value function's quadratic. That's hardly a surprise, right? Because you basically have an infinite dimensional quadratic problem, an infinite number of variables. You know that for a finite dimensional quadratic function if you minimize over some variables the result is quadratic in the others. In fact, the result is a sure compliment in the others. You know that all ready. This merely extends that to the infinite dimensional case and, in fact, sure compliment is the right thing because it's very, very close. In fact, if you stare at this long enough you'll see a sure compliment here. I should have written it that way. I don't know why I didn't. You could take the  $A^T$  and the  $A$  out of these two terms and you will see, I don't know how good your sure compliment visualization skills are, but you should see a sure compliment.

Anyway, it's plausible, right? Because a sure compliment is blah minus something, something inverse, something transpose where the third something was the same as the first something or something like that. Anyway, you know what I mean. So, anyway, that's a sure compliment, so it shouldn't be surprising. The value function is quadratic in this case and, in a sense, it is a sure compliment of an infinitely big positive quadratic function and you minimize over infinitely many variables leaving just  $N$  left and the result is, in fact, a giant infinite dimensional sure compliment, but as an explicit solution. This is the Bellman equation now written out. Because they're quadratic forms it just says the two matrices are the same and you get this.

Now, by the way, there's no reason to believe that you can solve this. It's a quadratic matrix equation, so there's lots of ways to solve this. One just by iterating and things like that, so I'm not gonna get into that, but this equation can be solved quite effectively. In this case, the optimal policy is a linear state feedback. It just looks like this.  $u^* = -k(x(t))$  where  $k$  is this thing here. You can actually see all sorts of various interesting things here about what happens when  $P$  gets big and when it's small and all that kind of stuff. You can actually work out a lot of the behavior. This thing, I think a bunch of people have – actually, how many people have seen this? I'm just sort of curious. Okay.

So, if you've seen this you probably know this is extremely widely used. Even in cases where systems aren't linear and so on like that and it is unbelievably effective. I mean, state feedback gains like this just work really well. In particular, it's sort of like least squares. I mean, it's almost like magic in the sense that when you synthesize state feedback gains like this – I mean, this is not interesting if this data is small and  $U$  is small, but if you've got 20 dimensions and five inputs that's 100 numbers in there and 100 numbers just come out just like calculating a pseudoinverse and then doing image processing. It would take you a long time to adjust those hundred numbers by hand to get something that even works let alone works so well. This is widely used. Okay.

Now we're gonna look at this finite horizon approximation and this is quite straightforward. What we do is instead of solving this infinite dimensional problem what we're gonna do is we're simply gonna run it out to time capital T, that's some horizon. What we'll do is we'll insist that when you arrive at the end of this horizon you drive the state to zero. Now, once the state is zero you have the option of putting in input zero, zero, zero after that. We're essentially looking at a finite dimensional subspace of possible inputs. The inputs go out to a certain amount of time, they drive the state exactly to zero, that's this. Thereafter you apply zero to the whole thing.

So this would give you a valid input on infinite horizon. That's what this is. Okay. Now, then you have a finite dimensional convex problem. We can solve that easily and this will give you a sub-optimal input for the original optimal control problem. Why? Because it's basically you're looking over a finite dimensional subspace. It's the subspace of signals of inputs and states that banish after time T. That's a finite dimensional subspace. You solve the problem over subspace you certainly get an approximation. You get an upper bound. Let's just do an example of a system with like three states and two inputs. It really doesn't matter.

It's just completely random. Just quadratic stage cost and we'll have a limit on the state. None of the components can be bigger than one. Same for the inputs, they have to be less than point five. Who knows why. And then here's our initial input and the optimal cost turns out to be 8.83. If you're curious, I can tell you why that's not an approximation. That's the exact number. I'll explain it in a minute. Not that it matters, but let me get back to that in a minute. So here's the cost versus horizon. In this problem it turns out it takes ten steps. Notice if you say get to the origin it's infeasible for ten steps because you have some state, it propagates forward in time, you have an input, which is limited between plus/minus point five. There's this limited amount you can do.

By the way, those of you who are in 263 I should say that I made fun of it then, but I'll make fun of it even more now. This very traditional idea of like controllability and all of this. I don't know if you remember this. If you've blocked it out that was the right thing to do because, I mean, it's perfectly okay. It's the first thing you ask. Can you wiggle the input to take the state to zero or something and you get these kind of silly things involving ranges and all that and that's all fine, but the real question is not can you maneuver the state to zero in capital T steps? The real question is, can you do it with an input signal that's less than minus point five? That's actually a real question. That one's of interest and no stupid theory of controllability is gonna tell you that. Okay?

In my opinion, this is the real theory of control. This is the first theory that's actually useful. This is with  $b$ ,  $a$ ,  $b$ ,  $a$  squared and  $b$  and all that. The Kreloff subspace, as we now know it. Being full rank and everything, I mean, I don't know. It's classical and, as far as I know, it doesn't really have any actual use, but okay. So here's a real controllability analysis. It tells you can you hit the origin. It says that for even in nine steps you can't do it. It's infeasible, so the cost is plus infinity. In ten steps you can do it at a cost of 11.5. Then in, I don't know what this is, in 11 you can do it down this way. You see something here that this is converging very, very quickly. This is very common.

It's not universal. I should say, obviously this is monotone decreasing, but, in fact, there's a lot that would tell you that this thing is gonna converge very quickly. In other words, it makes sense because the optimal inputs, I'll show you what they look like. Here's the actual infinite dimensional optimal input and you can see very carefully what happens here. The input looks like this, but then after a while it's dying down like this and then that's dying down like that and, of course, this is falling off exponentially. Here it's not gonna take – if I tell you, in fact, you have to be zero or  $T=30$  it takes very little perturbation of this input signal to drive this state to zero here and not have it be merely small. Takes just minor perturbation.

That's why these things converge very, very quickly. Okay. So this is the cost versus horizon. Actually, people who have had these courses, I don't know, 207, 210, anyone know how we calculated that line? This is just for those who have the background. I mean, look it would be perfectly okay to just do this, actually. Perfectly normal would be to solve this for  $T=50$  or  $100$  and numerically say that's it. That's perfectly acceptable. But, in fact, you can calculate it exactly. I mean, up to this numerical round off here. Do you know how we did it? Do you know?

**Student:**[Inaudible].

**Instructor (Stephen Boyd):**This is with constraint. Yeah. So here's how you do it and this is just an aside, but just for fun if you have the background. You can show in a problem like this that you can compute an ellipsoid in state space such that if you're in the ellipsoid the optimal control is, in fact, coincides with the linear one because you'll never saturate, never in the future. Therefore from then on it's given by an explicit formula. In fact, it's this formula right here. All right. This is an advanced topic, so just ignore me if you don't know what I'm talking about. Also, I'm going the wrong way here. There we go. So it's given by this formula and once you have that you can actually evaluate the cost to go by solving the Lyapunov equation. Okay.

If you knew what I was talking about, fine. If you don't, just ignore what I just said. Okay. And if you look here you can see all sorts of things. This is the smallest amount of time you can possibly drive the state to zero and you can see this is really quite different. The input is quite different although, actually, curiously it doesn't start off that differently here, right? I mean, it basically says I'm just showing you  $U$ , but I think maybe there's several  $U$ 's. I forget and I don't know why we're showing two inputs. Okay. So this is one of the  $U$ 's. I don't know which it is. We'll fix this typo. It's either  $U_1$  or  $U_2$ . I mean, not that it matters, right?

You see it does something and then it pulls back. This is all because you have to drive the state to zero. There are two other states as well that get here from the zero up here at ten. Okay. All right. Now this brings us to Model Predictive Control. This is something everybody should know about this. Unfortunately, it's considered part of control. It's not even mainstream control because it's considered part of industrial control. I think it's really stupid. It's incredibly effective for all sorts of things. It goes like this. Here's the way it's gonna go. There are some subtleties, so I'm gonna take some time to do it right.

It's at time  $T$  and here's what you do. You solve the following problem. You have a planning horizon capital  $T$ . From time  $T$  forward, capital  $T$  periods you are going to assess a cost, you will – all of this is essentially a planning exercise and it's gonna work like this. You're gonna propagate forward the dynamics of the system. You are gonna start at the current state. So you're in a current state at  $X=T$ . You will do a planning exercise, which will actually propagate the state forward, capital  $T$  samples. You will insist that capital  $T$  samples in the future you must hit zero. By the way, there are variations on this so-called terminal constraint, but the easiest is just to put that there.

You'll do this planning exercise. Literally, at time  $T$  you will work out an optimal control that would take the state to zero in capital  $T$  steps starting from there and so on. By the way, at this point the  $U$ 's here they constitute a plan for the future. If you were to execute that plan, then the state would indeed be at capital  $T$  plus capital  $T$  it would go to zero. The clever part is this, you're gonna do a complete planning exercise and you're simply gonna use the first input. Then you're gonna step  $T$  forward one unit and you're gonna re-plan, again. Your horizon has just moved and that's how this works. It's always like capital  $T$  is 30. You're always planning to drive the state to zero in 30 steps at all times. That's the way it's working.

By the way, another name for this is receiving horizon control, finite look ahead control, dynamic linear programming because in a lot of supply chain problems these are linear or piecewise linear, so it's got lots and lots of names, including it probably comes up in lots of places where it doesn't have a name because it's kind of so obvious. Okay. So that's the idea. Now, the fact is this is, actually, also a state feedback. It's a fantastically complicated state feedback. Well, it's not that complicated. To tell you the truth it's piecewise linear. We'll get to that in a minute. It is quite complicated because it basically takes a bunch of data, namely  $x(t)$  and solves a big QP in order to calculate a full trajectory.

So I'm just trying to think if I should draw a picture. Well, why not. I mean, it's the basic idea. I guess it's kind of obvious, but just to make sure. The way it works is here's time and you're right here. What you do is you make yourself a plan of input, something like that, that will take a state to zero at this time. So this is the plan that you execute at this time step. You then simply execute this. That's the  $U$  that you actually apply. You then step forward one to this point, but now your horizon has extended. There are other versions where it doesn't. Your horizon has extended here and now there's no reason to believe that you'll do this. If you did this, you would actually take it to zero, but now, actually, what's happened is your requirements have relaxed. You had 30 steps to take the state to zero before.

You step forward now and now instead of 79 it's 80 you have to hit and, in fact, this one might relax. It might be I won't be so aggressive. It'll be slightly lower, so you can actually watch this, and then it would take it to here. Okay. So this is the basic idea. It's complicated. I mean, it's really interesting. In this case, you would solve a convex problem every step. For that baby example we have, here's the way it works. Well, this is the MPC compared to the optimal. You see something that's totally insane here, which is

basically if you tell this thing plan ten steps in the future – by the way, if you ask it to plan nine steps in the future it can't even hit. It's not even feasible.

So you'd say plan at all times I would like you to act as if I want you to file a full, legitimate plan that ten steps in the future would take the state to zero. Here's how well it works compared to optimal. But if you say like 11 or 12 it's basically optimal. Yeah?

**Student:** So the way the graph is drawn it seems a little bit unfair to put them both on the same horizontal scale because one is [inaudible] than the other isn't it?

**Instructor (Stephen Boyd):** Yes. Right, right. Yeah. We do have to explain a few things. This one was calculated, as I said, by solving a QP big enough and then adding on an  $N$  from a Lyapunov equation. That's correct. And this one here, each point on that curve was done by solving one QP. Very good point. This thing was done by solving a QP for whatever 40 steps or 50 steps or something. You are right. The computation involved here is much more. That's correct. Yeah. Actually, even though it's a baby example and it's not always this dramatic there's a hint here that things like this work really, really well. I mean really well. The reason is actually pretty simple.

It does make sense. It goes something like this. As you pointed out, you're solving 50 QP's here basically. You're solving 50 different quadratic programs or something like that. All the time the QP you are solving is that you're making a plan of action for  $T$  steps in the future. Let's make it 12, so you're like out here. Basically, you're doing flawlessly. Basically, at all times, you're filing a plan of action to take the state to zero in 12 steps. What happens next? You then execute only the first input. You step forward and now instead of saying, well, it has to be zero at that time, that time when it has to be zero has shifted. Okay?

Let me explain why this works well. There is a sense in which at each step you basically compute a whole action. Basically it's a whole action plan is what it is. It means that if you were to get the signal all over, done, quit, stop, everything like that, then you would execute that plan in 12 steps or whatever it is. Capital  $T$  steps later you'd bring the thing to zero. So at all times you have sort of a viable plan that would halt the system in 15 steps. That's kind of the – or whatever capital  $T$  is, right? So, in fact, we have  $T=15$ . You're calculating 15 values of the input, but only one you use. Only the first one. The other 14 values of the input is nothing but a planning exercise and its only reason for being there is so that you don't do something stupid now that messes you up for the future.

That is the only reason for that planning to be there. By the way, once you realize that that's the basic idea, that the reason you're making a full planning is just so you don't do something now stupid with respect to the future, then it starts making sense that you could do a crappy job of planning and you would expect it to actually work. I mean, this sort of makes sense. Imagine this in some kind of a cash flow problem, right? Where you're managing some cash flow and that'll actually be something for next lecture, which is Stochastic MPC, but in a cash flow problem you would have inputs and now I've got

expenses, I've got income, and stuff like that. What you do is you simply do some planning now. What you do now has a huge effect on the future, right?

You invest in some fixed income security that matures in three months it's gonna be a cost now, but it'll be a benefit in three months. You run your cash balance down real low now to pay for something, then you're gonna have to borrow and that'll be expensive when you have expenses come up. Everybody see how – I mean, it's the exact same problem. So MPC there would do just what you would imagine. You'd basically do, in fact, what you're required to do if you're a CFO or something like that. You, actually, have just a plan. You make a plan for the next 12 quarters on cash flow.

Now, so far everything's deterministic, so we, sort of, assume nothing's gonna change. We'll see that the real advantage comes when, in fact, things are not what you predict them to be. That's Stochastic control. That's the next topic.

**Student:** Is there a way to solve this problem with [inaudible]?

**Instructor (Stephen Boyd):** Uh, if they are changing, but linear, yes. That's no problem. But if they are non-linear it's not a convex problem and you know what my recommendation would be then? I don't know. Lecture whatever it was, sequential convex programming. It'll work quite well. Yeah?

**Student:** If the dynamics are changing, do you have to keep solving more feasibility problems to figure out your horizon? It seems like before you'd do this problem you have to figure out the feasibility problem to get your minimum –

**Instructor (Stephen Boyd):** Right.

**Student:** – or you can overshoot it, right?

**Instructor (Stephen Boyd):** Right. We'll get to that. Generally speaking, if you're running into feasibility issues your horizon is too short. The other thing is that when you run this, believe it or not, it'll work perfectly well if you violate feasibility. So if instead of saying  $x(t) + T$  is equal to zero, I put a big charge on capital  $X(t)$  like ten times its absolute value or one norm or something like that. This would work well now even, believe it or not, or tolerably it would work out here because it would just try its best to get that state zero. I mean, it wouldn't get great performance, but it wouldn't just like stop, right? So, yes. When you do these methods, you have to deal with gracefully, right?

Because if you do a plan for the future and someone says let me see your plan and there it is. It's like nan, nan, nan, nan, nan, right? That's, yeah. So you don't – you have to relax all the things. This just makes it easiest to present.

**Student:** Just a question. Has anybody looked at like different [inaudible] forms so you could reuse some of the computations you had done?

**Instructor (Stephen Boyd):**Excellent question. Okay. When this is running there should be something that's very clear. That at each step, if you're the CFO, you worked out basically how to wind up operations in two and a half years, right? But do so, by the way, minimizing your total expense or maximizing your lifetime or your time to live or whatever it was. I mean, it's a very simple thing. Now you step forward and you're actually solving almost the same problem, right? That's your point. That can be exploited unbelievably well by using a warm start method. Then that gets back to your question, which is the computational complexity.

It is true that you're solving 50 QP's here, but you're solving 50 QP's and each is only a very minor perturbation on the other and, in fact, if you use a warm start method I guarantee it'll be two Newton steps to go. Basically, what happens is if someone says, "Okay. Now T is 17. Let's figure out what to do." And you say, "Okay. Well, what I have to do is I have to do a plan from T=17 to T=29." That's if capital T is 12. Okay? But you just did a plan from 16 to 28. So you take that plan and that's your initial point for your new plan. Two Newton steps I promise you, three, I don't know. It's all over, done. So, in fact, these things are fast. Okay? We'll get to that.

**Student:**[Inaudible] about the [inaudible]?

**Instructor (Stephen Boyd):**Yeah.

**Student:**This is very deceiving because what you're saying when the finite horizon control problem relate in 15 steps you are going –

**Instructor (Stephen Boyd):**Oh, you mean compare this one to this one? Yeah. Right.

**Student:**Another one is a strategy to compute, like, a somewhat different strategy to compute the optimal –

**Instructor (Stephen Boyd):**I agree.

**Student:**And you are actually going to reach near zero or some region near zero within terms of the radius near zero in some, like, 100 steps or 50 steps, not really in like 10 steps –

**Instructor (Stephen Boyd):**Right. No, I, no, but technically – okay, that is true. But let me put something out. Technically all three of these are showing the same thing. This one doesn't depend on horizon because it's the exact global optimal.

**Student:**Right.

**Instructor (Stephen Boyd):**Okay? Now, this one says bring that state to zero in 15 steps and then stop.

**Student:**Right.

**Instructor (Stephen Boyd):** That's a viable U. I mean, it's a U, it brings a state to zero, and it just sits there. It runs up a bill.

**Student:** But if you look at –

**Instructor (Stephen Boyd):** It runs up this bill. What's that?

**Student:** If you look at your first MPC and you actually look you will see that as if you are incurring the cost of 11.5.

**Instructor (Stephen Boyd):** Right.

**Student:** And then it would suddenly realize that I only used the most important then my cost is going to use –

**Instructor (Stephen Boyd):** Right. Well, what happened was at this point what happened for this MPC thing, on the first step, it's actually quite rushed, right?

**Student:** Yeah.

**Instructor (Stephen Boyd):** Because it's basically saying you minimize this cost, take this state to zero in 11 steps. It basically in nine steps is impossible. It's not even feasible to do, so it's feeling quite rushed and it kind of has got some input that just barely does it. Then what happens is, in the next step the horizon moves another step forward, right?

**Student:** Right.

**Instructor (Stephen Boyd):** So I think they're perfectly okay. I don't know. I think I'm not – they're showing different things, but it's okay. Here are the trajectories generated. This is where MPC with  $T=10$  and, I guess, the point is here that at least visually, I mean, you know all ready you've seen the plot, they're kind of the same.

**Student:** Do you build anything like games like this where you have two players –

**Instructor (Stephen Boyd):** Yes.

**Student:** – and both of them –

**Instructor (Stephen Boyd):** Yeah. Yeah. So people have done games too. That's right. Yeah. Yeah, infinite versions of this and all sorts of other stuff. Okay. Here if you look carefully, I mean, you can see that these are – well, they have to be close, right? Because the costs were close, but, I mean, they're shockingly close. But the point is that this is calculated in a very different way. This is calculated by solving one QP essentially infinite horizon, let's leave it that way. Here. That's what this does. This says that this point was found by saying wind up in ten steps. Then it moves forward. It creates a whole

plan for winding up in ten steps, moves forward in one step, now it's  $T=2$  and it says now given the state you're in plan to wind up in ten steps.

So what's shocking is that each planning exercise is stressful in the sense that it's not even feasible for nine steps to get there. Well, it might actually start being feasible once the state has been reduced, but at first it's not even feasible. Each planning exercise is stressful, but the resulting input generated by sequence of stressful planning exercises is actually superb. I mean, this obviously doesn't always happen, but this is pointing out something which is correct in practice and that is that these things work like spookily well. I mean, if you have to solve things like this you should do this. Okay.

Now, I've all ready said some of this. It's got lots of names in the 70's. This was propagated all across chemical process plants and it had the name dynamic matrix control. You might want to ask why. The matrix means that they wrote a big matrix that related the future inputs to the future outputs and I'm not kidding. That's what it is, so that was heavy technology, but widely used. There was like four companies. They're used just everywhere now. So you go to a big chemical processing plant it's almost certain this is what they're using. Let's see. And they can get quite fancy. By the way, I remember talking to someone from Shell Canada and he said that they actually have in these things that are – so I'm talking about big giant process plants, right? By the way, this is very impressive because these are also very dangerous, right? You send the wrong signals to a giant cracking tower and you can be very, very sorry.

Also, they're extremely high value assets as you can imagine, right? So he said that, actually, in the QP, there's coefficients that go into that QP into the cost function come from analysts and things like spot futures prices and things like that for the future. Then I pointed out to him, I said that means that some guy in Chicago who raises a card, it's not done that way, but, anyway, raises a card on something, actually, more or less directly ends up affecting valves and pumps in Ottawa. He thought about it for a minute and he said yes, that's true. So I think it's kind of cool actually. I think it's cool anyway. Okay.

It has lots of names, rolling horizon planning. A lot of people do it and don't even say it so that's the other thing. Now, actually, what's interesting is that it's propagated into industries where the dynamics are slow. So, in fact, I just recently found out it's what's used, although not called MPC. It's used in so-called revenue management. Do you know what that is? That's basically these sleazy ways companies get to part with most of your money possible and you'll be happy to know this is what's doing it. This is how they release the number of airline seats at this price and then they change it and then more and then more and then they sense desperation like you say, "No, I need to be at this meeting in Chicago." They go like, "Oh, it's really expensive. I'm sorry about that." And you sit next to someone who paid 1/12 the price. The euphemism for that is revenue management. They're managing it all right. At least according to my friend at business school, these are the methods used. Although no one doing it would recognize the terms Model Predictive Control. They'd say never heard of it. They would call it revenue management and it would be this. Okay. All right. Let me say a few more things about it. Actually, there is something very important about this that I should say and I think it's

extremely relevant. I think you hit it right. You went right to the key point. The key point is in these systems in these methods if you're running control you're solving a convex problem at every step.

Now, if you think about control and things like that, things that run in real-time, that is very non-traditional, right? So the way controls that were implemented in many industries up until recently, and included in many now, is the control which runs at real-time is just a very small amount of calculation, right? You spend a billion dollars on all sorts of stuff to figure out some control gains, but then that actual control code is like 14 lines of C. Got a couple of switches in it, couple of flops, and things like that and then it can be checked for correctness like a zillion times or whatever. Then it's installed in a flight controller. Not, by the way, because it's updated so fast. These things are only updated in like 50 hertz or something like that, so you have 20 milliseconds. It's just for tradition and various other things. Everybody see what I'm saying?

So the most common type of control is like PID, which is Proportional Integral Derivative. Basically requires about three multiplies per step or something, right? You get the new output compared to your target. That's your error. You do have to add that to the previous integral error, right? And you multiply by three numbers and that's what you do. Everybody see what I'm saying? When computers were slow this all made a lot of sense. People who did traditional control ignored this entirely through the 70's and 80's because they said it's a special thing. I mean, it works because chemical process plants have very slow dynamics that go for like 24 hours and if you have 15 minutes you can go ahead, solve a QP every 15 minutes. What do we care?

If you're doing revenue management solve a QP every 15 minutes. What do we care? LP in that case, right? Go right ahead. So people who did sort of the real-time control said, "Oh, no, I work on robots. I work on jet engines. I do networks. Our decisions have to be made at the millisecond or microsecond level. I have to schedule processes. I don't have 15 minutes to figure out what to schedule next. I've got to make decisions every millisecond." So there's a whole bunch of areas like that where whether those people know it or not this is coming for them. Mostly they don't know it, but this is coming. Okay.

So let's look at some variations. I've all ready mentioned this that instead of insisting that the final state be zero you can make a final state cost – make up some  $V$  had equals  $V$ . By the way, if the final state cost were exactly the cost to go function then, no matter how short the horizon, you would actually get the exact optimal input. In other words, even if it was one step – well, that's literally Bellman equation. The Bellman equation says let's do some planning, ready? Let's plan only what you're gonna do right now, but then you say, well, how do I count for the future cost? And you go by the exact effect on the future cost that would be  $V$ . If you put that accounting in and optimize Greedy locally in time you get the exact optimal. Everybody see what I'm saying?

So you might imagine, for example, the end point condition  $x(T)+T$  is zero. That is effectively a  $V$  had. It's the following  $V$  had. It's zero when  $X$  is zero. That's plus infinity

otherwise. Not exactly what you'd call a superb estimate of the cost to go. Anyway, if you can put in an approximation of a cost to go here it'll work really well. For example, if a problem is like linear quadratic and zero is in the interior of the thing, you can actually solve the Riccati equation and put  $X^T P X$  as this thing and you'll just get amazingly good controls at that point. So that's just one thing. It also deals with the non-feasibility issue quite gracefully.

In other words, that way when someone says can I see your plan it's not gonna be nan, nan, nan, nan as inputs right is. Okay. Again, on that topic, you convert hard constraints to violation penalties because you don't want to just say I can't make the plan. What you do is you just violate some penalties. What's really weird about this is you'd think things would work poorly. You can actually run MPC where all of your planning exercises are, actually, infeasible. If you freeze it and you say, "What are you doing?" You say, "I'm planning my actions for the next 14 steps." You go, "Great, what are you gonna do?" "Well, I wanted to make the final state zero, but it's infeasible. Also, I'm violating a few things. I'm violating the dynamics equations and things like that." And you're like, "You call that a plan?" And you go, "Hey, it's real-time."

Shockingly those things actually work really well. It kind of makes sense because if you think of it this way, if you think what's the plan for? The plan is there only so that you don't do something stupid now with respect to the future. So if you're planning is not perfect – well, I guess if it's infeasible it's a little bit worse than not perfect. I mean, at least it's some kind of rough plan. You couldn't execute it, but anyway. All right. Another variation is this. You can solve the MPC problem every capital  $K$  steps. Basically what happens is that at  $T$  you plan out capital  $T$  steps and you plan out a whole trajectory and you execute the first five. You just do those. Then after five steps you see what state you're in, there's no disturbances here so the truth is you know what state you're in.

Your horizon also shoots back five steps and you re-plan, so that's another variation on these. There are lots of variations. Okay. Now, let me talk about explicit MPC. This is a special case for these. The loss function is quadratic and then these are polyhedral. The state and input constraint sets are polyhedral. Then it turns out that phi MPC is actually piecewise affine. Okay. It looks shocking or cool or something, but it's not. It turns out that the solution of any quadratic program is piecewise affine in the right-hand sides of the constraints. That comes straight from the kickety conditions.

By the way, this comes up in machine learning also that when you do an L1 regularized problem and you put your lambda there, you can say lots of things about the  $X$ , like that it's piecewise affine in lambda and you can say various other things about it, but it's a good thing to know and it's not hard to see. Theoretically, it says that if you're gonna solve a QP with many instances of the right-hand side you can actually pre-compute an explicit lookup table for the solution. Now, you can do this for any QP for any application. Now, what happens then is you simply calculate these regions and then your code looks like this. Lets call your parameter  $P$ .

P comes in and the first thing you do is you find out which region P is in and then you simply apply  $KP + G$  where the K and G are the correct ones for that region. Okay? So that's how that works. This is practical for very small problems only. I mean, certainly an empty C. I mean, these are things like two and three and things like that. Small number of constraints. Now, people have pioneered in the last, like, five or six years methods that actually analytically construct these things and they make very pretty pictures. You can type explicit MPC into Google or something and you'll see beautiful pictures with a two-dimensional states base and some kind of tessellation and 180 regions, and they actually use these things.

They've used them in anesthesia. They've used them also in switching power supply. Obviously, these things can get slow very quickly, which is kind of the – the whole point was to make it fast or something. Again, this comes from 1969 mentality. 1969 mentality was solving a QP is a big deal. That's 1969 mentality. Big thing. You need a big super computer, you need expensive software, a couple of Ph.D.'s to sit around, big 208 volt three phase, 50 amps. People just think of it like it's a big deal and, as you all know, it is nothing. You can write one in Matlab in 20 minutes. You could actually. It's just not that big a deal.

You can write one in C in half an hour or something like that. So a lot of it is sort of demotivated by that. Now, you can do things to simplify these things and get quite a good control. There is another advantage to an explicit form and the explicit form is actually quite traditional in control. That would be called gain scheduling or something like that. The person who is traditionally trained to make sure that before some control code is downloaded into something dangerous or expensive they'll be very familiar with a switch, right? With a big C statement that lists a bunch of cases and in each case dials in a gain. That's completely standard. Okay.

But, in fact, again, if you think past the 1969 mentality here's what you would see. The MPC problem is highly structured, so there's a whole section on this, but the Hessian is block diagonal. I'm not gonna go into the details. The equality constraints are block banded. It's weird because the equality constraint matrix isn't square, so it's not clear what banded means, but if there were a meaning for banded square matrix this would be it. But, actually, you know why you can guess this and all you needed was your intuition to know this? This is one of the things I would hope you would eventually get. Probably by the end of this year you'll have it.

It has to do with going from problem structure in the actual problem to the problem structure in the numerics and what are the implications, so let's just talk about that for a minute. Lets say you're solving an optimal control problem and let me just ask you about the state  $X(t)$ . What does it connect to? Where do the other variables it's, sort of, directly constrained with or connected to?

**Student:** T plus one.

**Instructor (Stephen Boyd):** That's what?

**Student:**  $T$  plus one and  $T$  plus –

**Instructor (Stephen Boyd):** It's  $X(t)$  is connected to  $U(t)$ . Lets say it's connected to  $U(t)-1$  because the previous decision is what landed you in that state and it's connected to  $X(t)+1$  and  $X(t)-1$ . That's it. Whenever you can make a statement like that – now, of course,  $X(t)$  is related to  $X(t)+50$ . Okay? But indirectly because it's got to propagate through all these dynamics and things like that. Otherwise, it's just a separate problem, but the key is whenever you can identify blocks of variables and say that – in fact, what we just argued means it's banded. If you stack  $X$  and  $U$  and  $X$  and  $U(t)$  you only depend kind of on  $X$  and  $U(t)$  the previous and post that smells like banded. That says banded.

I mean, you have to go through and work out all the matrices and all that, but the point is at least you know what you're looking for and you're looking for banded and what does banded mean about solving it? It means blazing fast. It means linear in the time step. Okay? So that's the other thing you should know. Banded – how fast does it take a Newton step if you have a band of 20 and it's a million long?

**Student:**  $NK$  squared.

**Instructor (Stephen Boyd):** Hmm?

**Student:**  $NK$  squared times –

**Instructor (Stephen Boyd):** Yeah, yeah, time is the answer. Super-fast. And it's not easy solving million variables, million equations generally. Okay? I just want to talk about the ideas. You should be used to this. Structure comes up other ways. In image processing what structure comes up? What are pixels related to typically?

**Student:** The surrounding.

**Instructor (Stephen Boyd):** Yeah, some neighbors. Okay. So did you get banded?

**Student:** Yes.

**Instructor (Stephen Boyd):** No. You get banded across a line, but you're also related, so you get something called the chronicle product of banded. You're related to your left and right neighbors for sure. Things like signal processing control, anything with time in it, banded. Images you get actually a chronicle product of band. You get weird bands because you're related to the guy to your left, the guy to your right, but you're also related to your neighbor above and your neighbor below. Yeah, okay. Make your more complicated nine point. That hardly changes anything, but it means that you're related here and that's, by the way, an offset of  $N$ , right?

So the sparsity pattern there you would expect to see would be bands. Actually, it's a banded matrix of banded matrices. By the way, it doesn't matter. The main thing you should recognize is structure. That's structure and that means fast if you know what

you're doing. Okay. So you can actually solve this. I guess we've gone all over this. I don't have to tell you this. It's  $t(n+m)^3$ . So to do this it's linear in  $T$ . Here's a fast MPC. In fact, it was just put together by Young. We've all ready talked about some of the things here. First of all, you don't have to do a good job in the planning, so you might as well do an interior point method and fix the barrier parameter and then you can use warm start, which I think – who asked? You asked about that, right?

So you could do warm start and then it turns out you only need to do like two Newton steps or something like that. You can limit the number of Newton steps and this is just a simple C implementation, not even optimized, and you can compile it with whatever it was, O3 or whatever it's called. These would be the numbers involved here. Here would be one. Here's a problem with, I don't know, four states, two inputs, a time step of ten, and that goes down in 300 microseconds. You can have a problem with 30 states, eight inputs, 30 horizon. That's a problem with 1,000 variables and 3,000 constraints. That goes down in 23 milliseconds. Okay?

So I don't mind – I mean, I just came back from Zurich where there's people who have been doing MPC for a long time and those who had the 1969 mentality successfully convinced other people that solving a QP is a big deal. By the way, these are also the same kind of people who just used tools and don't know how they worked inside. If you fire this up in CVX you'll get actually some very little CVX overhead time. You'll get 3.4 seconds. All right. This is actually why I think it's very important if you're gonna use things like convex optimization. Maybe you'll go into an area where it doesn't matter, and you can use high-level tools and you can use other people's solvers. Good for you. Very good for taste. Very good judgment to go into a field like that.

It requires a combination of modest number of variables and no real-time constraints. If you're considering fields go into a field like that. However, the point is if you go into something that is huge you're gonna have to write your own solver. We went over that a couple of weeks ago. If you do anything that's real-time you're also gonna have to write your own, but it's really easy. I mean, it is not hard provided you know what you're doing. If you've taken 364a and b you know what you're doing hopefully. You should know what you're doing or something like that. By the way, I do want to make a big deal over this. There's not a lot of people outside this room who actually know these numbers. Even though everybody should know them, I know all the experts who write all the big codes, if I asked them how fast would you solve – actually, no one even thinks about solving small problems.

So even the experts in the world when you ask them how fast can you solve that? They'll say a couple of seconds. Right there. That won't help you if you're controlling a robot or something like that or doing navigation updates or whatever it is. I personally happen to think there's a lot of opportunity for MPC running at rates like 50 hertz, 60 hertz, and so on. By the way, there's other areas where this comes up. There's a student who did a project in 364 three, four years ago that had to do with optimizing guard bands and wireless transmission. He used an SOCP with, I don't know how many variables, 200 variables, and during his project he got a C program to run it down to 500 microseconds

and then it ultimately went to hardware and it was right in there at packet time. It was in the microsecond range. I forget what it was.

These real-time things are actually quite interesting and fun. I think, actually, they're gonna have a lot of impact. Okay. So let's look at examples. This example essentially is supply chain management because it's completely generic, so here's what it is. I have a bunch of nodes and these are warehouses or buffers, it depends on if you're storing goods or storing packets or something like that, for fluid or whatever. This is gonna be single commodity. It's very easy to make this once you understand how to do this. Totally trivial to make it multi commodity.

Then you have  $m$  links that go between the nodes in the warehouses. These could be communication links. They could be airfreight or something like that. It doesn't matter. Pipes.  $X_i(t)$  is gonna be the amount of commodity at node  $i$ . In period  $t$ ,  $u_j$  is the amount of commodity transport along the link  $j$ . Then you will split the traditional graph node incident matrix into its negative and positive parts and we'll call those  $A_{in}$  and  $A_{out}$  and this basically tells you whether link  $j$  enters or exits node one.

It's something like  $A_{in}$  minus  $A_{out}$  would be the traditional graph node incidence matrix. That's the one that's got the one and minus one on every edge. The one tells you what it points in to and the minus one tells you where it comes from or the other way around depending on which standard you use. The dynamics is this. It's linear. It says that the amount you have at a certain time is equal to  $x(t)$  plus this is the amount that came in and then minus that's the amount that flowed out. And let me just for fun ask a question. Suppose you had spoilage. In fact, suppose I told you that this is a perishable good and at each step 10 percent of the commodity goes bad. Please modify the equations.

**Student:**[Inaudible].

**Instructor (Stephen Boyd):**Yeah. Thank you. Would it be linear then still? Yeah, sure. Okay. I'm just pointing out there's a lot of stuff you could do. I guess you did that, right? In your thing. If you have somewhat spoilage in airplanes that's not good.

**Student:**No.

**Instructor (Stephen Boyd):**No, okay, okay. All right. All right. Here are the constraints and objective. These are just sort of generic. One is to say that the  $X$  is the amounts go between zero and some  $x_{max}$ . You may not have to have this  $x_{max}$ . That would be a buffer limit or something like that. By the way, it's also very common in some cases to let  $x$  go negative. In which case, it represents a backlog. It's the same ideas having a negative investment in asset. It means you have the obligation to produce it or something like that. If a customer makes an order you can actually take the order and now you have a backlog, which just says how many units do you have on stock. You can say -12 means 12 are backordered. I mean, you're gonna have to add a penalty for backlog if you do that. You have to add a steep penalty for backlog, but that's another version of this.

Link capacities go between zero and some maximum value. This is a vector of the amount removed from each node or warehouse and that's gotta be less than  $x(t)$ . Some people call this actually a  $\alpha$  – do you remember what they? In fact, I just saw some presentation where this had some name to people who do supply chains. It means that what you can't do is you can't ship ten units to a warehouse and then ship it out on the same time period. Everybody know what I mean? So if you didn't have this it would mean you could actually do it all. Like a digital circuit, like a two-face clock, or something like that basically. On the rising edge all the trucks leave and this is that.

This says the truck can't leave with stuff that's not there and then on the following edge of that clock they all arrive. That's my model of it. Okay. You can have a shipping transportation cost that's  $S(u(t))$ . This could be positive, negative. It can include sales revenue, manufacturing cost, all sorts of stuff in it. Whereas a storage cost is just a charge on  $W$ . If you allow  $x$ 's to go negative, you would have, in fact, a back order charge, which could be real or just an internal accounting. Your objective would be to minimize this. I should mention one thing. There are some very common shipment and storage cost charges, which are not convex. That would be like a fixed charge. We're not gonna look at those, although you could.

They would be things like this. The cost to run a truck from here to there is fixed. So if you're gonna put two little boxes in the truck it's gonna cost you the same as filling it. Okay? That would have a cost that would go up like that and then, actually, it would jump up again. These would be highly non-convex. Obviously every time you jump from zero to one, one to two or trucks or something like that. We're not gonna look at those, although one could. Now we'll just look at an example. It's an example with five nodes, nine links here, and you start with everybody has some amount of commodity at different nodes. Presumably, this is sold or something like that, so we'll get revenue from shipping stuff out on these two links.

At each step I can ship whatever I like along these edges. I mean, it has to be there for me to ship it somewhere. I'll have a buffer limit of one. The most I can ship out is .05. My storage cost will look like this. It's linear plus a quadratic term. The shipping cost will just be linear. I pay linearly to ship among warehouses. Now, my shipping cost, I don't know if it's fair to call it shipping, but when I pull things out of those bottom two nodes I actually get paid for it. That's why this is negative. That's revenue from delivering product here. The initial stock is 10011 and we'll just run MPC with five steps or something like that. The optimal cost, by the way, is 68.2 and the MPC cost is 68.9, so, once again, just five steps is enough planning horizon to actually do very well.

And lets just see what this looks like here. What I'll do, I guess, is I'll start shipping these out immediately. That's clear. But then I won't have much space here, so I'll actually spread the product around and then ship it. Actually, the point is even baby examples like this what to do? Not obvious. Not remotely obvious. And you can imagine what would happen if you have 40 links and 50 and it goes on and on. Okay. So here's the example.

So the optimal was obtained just by solving a very long problem and this just shows you a couple of things. I guess the first one shows you – this is  $X_1$ , which starts at one. You can see stuff is shipped out, then held, and then it's shipped down until about 30. This is  $X_3$ . It goes from zero. It goes up and then down again. Sorry, that's – well, it's the same thing. There's the optimal and there's the MPC. They're not quite the same, but they're very close and you can see here these are the shipments made. The optimal says that, for example, U3 should ship a bunch then, sort of, stop, something like that, and then U4 should ship nothing, and then somewhere around 12 start shipping stuff out like that.

And then this is the MPC one. You can see, actually, it's not the same, but it's good enough to give you within two percent the optimal cost. In fact, if the horizon were any longer they'd just be indistinguishable, the two. By the way, just the pictures of these tell you this is not obvious. This is not remotely obvious what to do here. Maybe we'll make a movie. Did you make a movie? I can't remember.

**Student:** Yeah, I did, but it's not for –

**Instructor (Stephen Boyd):** Yeah. Presumably, you could make a movie or something that would kind of show you the right thing to do where after several steps you'd spread your product all around and then after one thing is done the stuff starts pumping out the bottom or something like that and that minimizes your various costs. Okay. Let me just mention a couple variations on the optimal control problem. We've all ready said this. You can have time varying costs, dynamics, and constraints.

One very extremely common thing, especially in finance, is discounted cost where the cost in the future is actually discounted by a discount rate, like you divide by 1.05 to the  $t$  in the future. What's funny about that is the people who do control have discount costs with negative, right? They actually put higher cost on the future, which induces the control to be more aggressive to bring something to zero faster. No one in control would ever, ever use a discount cost and everyone in finance would use a discount cost. If the whole thing is going down over a long enough period. We've all ready seen this. You can have a couple stay input constraints. You can add slew rate constraints. I'll just ask you one question, then we'll quit for today, and then that's our next topic, but let me ask you this.

If I added constraints like  $U_{t+1} - U(t)$  is less than that it's a slew rate constraint. It says that your input can't change by more than some amount. Tell me what that does to the problem structure.

**Student:** [Inaudible].

**Instructor (Stephen Boyd):** It what? It smoothes out the solution, sure, because it says that you can't change very radically. But what did it do to the problem structure that you need to solve? It preserves locality in time. In other words, it's another constraint, but it affects only things close in time. That should go in your brain. The whole center of your brain that controls sparsity patterns and things like that should be lighting up like banded,

banded, banded, banded, banded, and the other one should be going like fast, fast, fast, fast. Okay. We'll quit for today.

[End of Audio]

Duration: 81 minutes