IntroToLinearDynamicalSystems-Lecture07

Instructor (Stephen Boyd):[Inaudible], if you go down to the pad, I wanna make a couple of announcements.

So the thing I wanted to announce – you're gonna have to pan up here a bit here – is that we have a couple of sets of notes on the course Web site. These are, you know, either two pages or barely three or something like that. One of them is a set of notes on least squares, or least-norm solutions. We haven't covered least-norm problems yet, so you can ignore that part.

But you must read these notes. They're very short and connect to these we've seen in the class, and is actually gonna be very important for you to do the problems and things like that. This is one of the sets of notes.

The other is, of note, that collects in one case. It's [inaudible] unambiguous and clear, unlike any [inaudible], which actually explains how you solve [inaudible] linear equations.

So the question there is given a and b, how do you find out [inaudible] x such that x equals b? How do you find – if there is such an x, how do you find one? And this will explain it. And it's connected to all the stuff we've been doing, so. So now that's a – that's official announced. A few parts here we haven't covered yet, but we will in the next week, or even we'll hit it today. I'm not sure.

So let's continue with least squares function fitting. And the specific case is polynomial fitting. This is very famous and something you should know about.

So here's the problem. I have a polynomial degree less than n – looks like this – that's characterized by n coefficients – a0 up to an minus 1. So these are these – these are essentially the variables for us. And we wanna fit this to data. So data are a little – are samples, basically, of this – of some function, or they're just data. So ti and yi. So they're little pairs which consist of a t value and a y value.

And we want to fit – wanna choose a0 through an minus 1 so that when we evaluate the ti's at these – when we evaluate this polynomial at the ti's, we get something close to the yi's.

So the obvious basis functions here – by the way, if you really wanna do polynomial fitting somewhere, these are among – this is about the poorest basis you could choose. But that's another story.

So the obvious base function are simply the powers. So the first function is simply the constant 1. The next is t, then t squared, and t cubed, and so on.

Here, the matrix a is gonna have this form. It's a very famous matrix. It's called a Vandermonde matrix. And it looks like this. So each row is actually a set of ascending powers of a number. So this is t1 to the 0, t1, t1 squared, and so on.

By the way, you should never – you should not look at this matrix and just say, "Yeah, so what?" It should be completely obvious what this matrix does. This matrix, when you multiply by a0, a1, an minus 1 – so this matrix maps [inaudible] polynomial coefficients into [inaudible] here, you get p of t1 [inaudible] to p of tm.

So if you had to have an English name for this matrix, or you wanted to tag it with a comment or a description, an extremely good description of this would be something like a polynomial evaluator matrix because that's exactly what this does. It maps coefficients into a vector of the polynomial evaluated a number of points.

So it's actually known as a Vandermonde matrix, I suppose, because Mr. Vandermonde had something to do with this or something like that.

So that's – and you have – you're almost done now because it's essentially a least-squares problem if your goal is to minimize the sum of the squares of the differences between the measured data or the data here – doesn't have to be measured – and the model data. So that's the – you get a least-squares problem.

Actually, this matrix here, we can deal with that right away. It turns out it is a full rank, provided, actually, that these points, the evaluator points are different. So as long as the evaluated points are different and the matrix is as skinny, this is full rank.

And to check this – [inaudible] it's a good exercise just to see how that worked. Assume that these tks are different from each other – the sample points. And let's suppose it's skinny. If you have a times a equals 0, that's very interesting. What it means is that this polynomial here vanishes at n points t1 through tm.

But if you have a polynomial of degree n minus 1, and it vanishes – in fact, at n points or more – distinct points, I should say – vanishes at n distinct points, then that polynomial is 0, period. So [inaudible] if you have a cubic or something like this, and it vanishes at four points, it is 0, period. It's identically 0.

But to say that p is identically 0 says that a is 0. And that's just a restatement of the fact that the matrix a, here, has 0 null space. In other words, you're saying that is the columns are independent, or a is full rank.

So that's just a quick thing to check. But let's look and see how polynomial fitting works and looks. It's a [inaudible] bullets meant to have any actual, particular use, but here it is. I have a function, which, by the way, is a perfectly simple function to evaluate, so [inaudible] coming up with [inaudible] polynomial model for it. But it doesn't matter.

So here's the function. It's 4t over 1 plus 10t squared. [Inaudible] that, most assuredly not a polynomial. And we're gonna fit it with a polynomial just to see how this works. We'll take 100 points between t equals 0 and t equals 1. And we're gonna fit this thing with a polynomial. And the RMS errors for the bits of degree of 1, 2, 3, and 4 drop as .135 and .076, .025, and then .005. And here's a picture showing this.

So the first – this is the first 1-degree polynomial is the dash line here. Actually, it looks like it's almost the same as a 0-degree polynomial, which is a constant. But in fact, it has a very slight slope. You can see that the dashed function starts off below .5 over here, and it's slightly above over here. So it's got a very small coefficient of t in it.

This is the quadratic fit, and you can see that it sort of picked up this kind of dominant curvature over here. The cubic allows you to sort of get this little – this skewness, actually, quite literally. It allows you to get the skewness here. And the [inaudible], as you can see, is now at least visually fitting extremely well.

There's a question?

Student: What's like the best way to deal with a polynomial [inaudible] same [inaudible] 0?

Instructor (Stephen Boyd):That's an excellent question. And that's exactly what I was gonna talk about next, which is not in the notes. I was thinking your question while I was talking. So let's talk about it.

So the question was, is this a good way to get a polynomial fit to a function? And the most obvious other time you've heard about polynomial fits is, of course, calculus. So that is, in fact, what calculus is, although it generally stops somewhere around the second-order fit. So here's my short description of calculus. It says, take a function, and it says, develop a first- and second-order polynomial fit of that function near a point t0. Everybody know what I'm talking about? So that's what it does.

How does it measure the fit? It actually doesn't measure it by a sum of squares of errors or anything like that. It measures it by how close it is as you get closer and closer to the point. This is not making any sense, so I'm gonna draw some pictures.

So here's my function, like this. Here's my point. And the first order of the series will give you this function here. And if you were to look at here, you'd see you're pretty close here.

The second-order fit is gonna be – is – get something like this. It's gonna be f of [inaudible] plus m of t bar times t minus t bar. That's your first order Taylor expansion. [Inaudible] have f prime, prime of t bar times t minus t bar squared, like that. And this is t bar here. So that's your [inaudible]-order fit.

Now, one is gonna capture some of the curvature, like this. And the second-order fit nearby is gonna be not just very good – sorry, I should say it's gonna be very, very good. That means that's an order, then it'll fit very well.

So if you wanna – and this is, by the way, not the same you'll get from least squares. It is absolutely not the same. [Inaudible] if you take a little, tiny interval and chop it up into – and sample it at a bunch of point [inaudible], you should get really close to the first and second derivative. But they won't be the first and second derivative. They absolutely will not be.

The reason is this: In calculus, the fit is local. That's what it is. You're finding a second-order – you're finding a quadratic function that fits the function extremely well locally. How local is local? Well, that depends. I mean, you have to ask – you have to – it depends on the function.

In contrast, something like this [inaudible] get an excellent fit over a big – that's a macroscopic interval here. Oh, by the way, for this function, we can look at the Taylor series. What would the – what's the first order of Taylor series look like? Well, it's not – I mean, it looks like this. You draw a line – that there. That's the – there's the first order of Taylor series. And you can see right away, these are very, very different.

Now, this first order of Taylor series, though, will have an – a very, very low error if you're near 0. I could do this in another place, too. I could try the first order of Taylor series here and get something else.

The second order of Taylor series is gonna look like this. And then oh, let's – I would say there's not much of a curvature there, so it's gonna kinda – it's like this, and it's gonna do [inaudible] that.

It's [inaudible] – what it – what will [inaudible] this second order of Taylor expansion here is that near 0, it will be very, very, very good.

So we come back to your question, which is better? And the answer is, it depends on what you wanna do? Are you looking for a [inaudible] fit over an interval? You shouldn't be using Taylor series.

Are you looking for something that gives you the best fit when you zoom in arbitrarily close? You should probably be using Taylor series in calculus.

So actually, what I'm telling you now has some – has serious implication. The typical mathematical indoctrination now just drills into people derivatives, calculus, and so on. And so any time the – you're sort of conditioned – we're all conditioned – that includes me – whenever somebody says something like, "Give me a linear or an afine or a quadratic model of a function or a point," I just – it's in our – well, we've been totally indoctrinated. We just say, "Okay, it's the Taylor series or something like that."

And that's used in a lot of cases. A lot of – in a lot of applications now, people are getting away from the Taylor series, and they're actually looking at things like least-squares fits. In fact, some of the best estimation methods for nonlinear – this is, by the way, just for fun, now. This is not covered in the – this is not real material in the class.

Some of the absolute best estimation methods for nonlinear systems are now based on, literally, at each step, spring a bunch of points into – take a bunch of random points, plugging them into some horrible nonlinear function, taking those points, and doing a least-squares fit, not using calculus.

So – and those are – and those methods work unbelievably well. They are so much better than the ones based on calculus, it's not even funny, in practice.

Interested in these applications and finding a function that is close to the function you have when you're very, very close. You have a pretty good idea of the interval over which you wanna fit. And so these things work very well.

They're called particle filters. That's – if you care, – but I think that was a very long answer to your question. But thanks for bringing it up.

Couple of other topics we're gonna look at. One is this – I give growing sets of regressors. Actually, this comes up in [inaudible] problems. It's very interesting.

So it's this: What's gonna happen is, I'm gonna give you a list of vectors. These can be the columns of a matrix. And this is very important. They are ordered. So I give you a 1, a 2, a 3 in – they're ordered.

And this says, you have a family of least-squares problem, which fit a vector y as a linear combination – as a mixture of – well, first, just – and I'll vary p. So it's a sequence of questions. I say, here's my set of – fact, to this, we just did it with polynomials. [Inaudible]

[Inaudible] squared [inaudible], one of these 2, 3, 4, 5, and so on. Oh, and I don't mean 5, any 5, I mean the first 5. So let me say it right. I'll rewind and say it again.

You're given a sequence of vectors, and you're asked, what's the best fit with the first one, the first and second, the first, second, and third, first, second, third, and fourth, and so on? So it's in a specific order.

Now, it actually, in this context, these are called regressors. So in statistics, in a lot of other areas, you'll hear these vectors referred to as regressors. We might call them columns. And actually, in fact, what you're doing is you're solving a bunch of least-squares problems where you're actually taking a leading columns of the matrix.

So if we were to write this as ax minus – or a with the p up here, what a is, is it is [inaudible]. Well, I might have some master a here. That's my [inaudible] p, p [inaudible] of a. And that's what we're solving. That's the idea.

Geometric idea is basically, you're projecting y, a given vector, onto the span of a growing set of vectors. That's the idea. And I guess the verb you'll sometimes hear is to regress y on a1 through ap. That's a verbal – that's the syntax used in statistics. You would say that you would – you – we would say for doing least squares or something like that, or you're calculating a projection, or you could say lots of things. They would say something like you're regressing y on these regressors.

So obviously here, as p increases, you get a better fit. You can't fit – you can't find the best least-squares fit with seven columns and have it come out worse than the best least-squares fit with the first six columns because I mean, if – I mean, you could, by the way, have them the same. And that would only be the case in the optimal fit, with seven columns. That mixture calls for 0 usage of the seventh column. That can actually happen, of course.

But the point of this is, you get a better fit. So the optimal residual decreases. Now, of course, for us, this is quite straightforward. For us, you just do this. In fact, it's even more embarrassing than that. The way you might do this is you might do this. I'll just write the code for you right here if you don't mind. So right, that's how you would write it, say, in MATLAB systems as well. But it's pretty straightforward. But it's just [inaudible].

So you take – this says to take all rows, and take the first p columns. And then do a least-squares fit. So this would give you this.

And the formula's just this. Now, what I'm about to say was highly relevant in 1967. It's less relevant today. And let me actually have – I need to have a little digression, so little digression here about how all this stuff – where this all fits in, in terms of least squares, about actually doing it.

If you need to do least squares, let's say with 100 – let's take 100 variables – and let's say, I don't know, 1,000 equations. So that's the size [inaudible], and I'm gonna do least squares. How long does that take to do nowadays? I just wanna order of magnitudes. Are we talking minutes, hours, seconds? What are we talking?

Student: Seconds.

Instructor (Stephen Boyd):Seconds. Anybody who knows about scientific computing wanna refine that number?

Student: Milliseconds.

Instructor (Stephen Boyd):Thank you. I heard it. Milliseconds. That's measured in milliseconds, maybe a couple tens, maybe more. Do not be confused by the interpreted

overhead in MATLAB. [Inaudible] longer [inaudible] for it to figure out, you just type [inaudible] in, and call it the right allay pack function, in most cases, then to solve it.

So let me repeat. I should've actually worked out the exact number and just tested it on my laptop. You can do this.

Student:[Inaudible]

Instructor (Stephen Boyd):What is it?

Student: Twenty-one milliseconds.

Instructor (Stephen Boyd):Twenty-one milliseconds. That was Jacob. Now, again, you're the class who couldn't care less when I had a 3 [inaudible] a to d. See, I'm still pissed off about that this – for the record. I had a 3 [inaudible] a to d, and subsequently was making predictions of the input to [inaudible] six bits. You're like, "That's cool. That's fine."

So this probably means nothing to you. Anyway, these are things that actually – well, you know, you can just back – you can just propagate Morse law backwards and figure out what it would've taken, what can your [inaudible] to do this.

So the point is, is it's all just a nonissue. So if you needed to do this for p equals 1 to 30, the whole thing would be over and done with in a loop [inaudible] for a second. Boom, half second, it's all done. This is not the case 30 years ago or something like that.

So by the way, let me say what I think the implication of this is. The fact that you can do this – by the way, this is absolutely an enormous amount of signal and data processing. Think about what this is. This is – you can match 1,000-long vector with a linear combination of 100 things in 21 milliseconds. That's on Jacob's laptop.

So this is absolutely amazing, and it's something you should fully appreciate. It means that a lot of the stuff we're looking at now actually can penetrate into areas where people have absolutely no idea can penetrate into. So I wanna make that point that a lot of people who think of least squares – somebody who's older, took – statistics, "Oh, we use least squares all the time. Oh, yeah, we used to regress like ten things on 15 measurements," or something idiotic like that.

This is a totally different era now, totally different era. You can solve – and that is actually a small problem – but think about what this would involve. But just think of the data – the data here is 100,000 – well, a 101,000 real numbers.

That's not a small amount of stuff. So I just wanted to press upon you, you now live in an era where the stuff we're talking about, it's not just that you can do it. You can do this unbelievably fast. And my opinion is that very few people in the world actually appreciate the significance of this. That means this could be done – there's lots of places

where you could be doing all the methods in this class. They could – they're now down to things like frame rate for video. They're gonna hit – in many cases, if they're much smaller, they're already at audio rate. And a lot of people don't appreciate. Of course, some do, but sorry.

That was just an aside, but I wanted to say it because it's actually very important to say how all these things fit into the big picture. Now, having said that, there's actually something very cool, which, as I said, is no longer relevant, in my opinion, but is much relevant in the '60s when this would be a really big deal. So if you get a paper from 1960 or something like that, this would be a really, really big problem, about as big as things would – this would take a long, long time, so.

So it turns out that if you form the qr factorization of a, then it turns out whether you like it or not, you have actually, for free, calculated – you've actually calculated the qr factorization of all the leading columns of a, so these a's of ps, like this.

So – and what that means is absolutely amazing. It says that if you do least squares using qr, it says that you can actually calculate the least-square solution. You don't have to have a loop that says, for p equals 1 to n, carry this out. And in fact, you can do it much, much faster. In our case, as you can see by the numbers involved here, it really hardly matters.

By the way, it would matter if you're doing some kind of [inaudible] thing, doing this in microseconds or something, or you're solving your problem with 10 million variables. Then things start mattering. But [inaudible] – what this says is you get – then this used to be a big deal. It's not a big – it's not – I think it's not a big deal now, so.

So right now, they – really no shame in your writing code that looks like this, as long as you put a comment there to preserve – just as a safety factor. You put a comment that says yeah, yeah, I know this can be done much faster, but I'm lazy. Then put your initials after the comment. [Inaudible] then you're totally protected.

Now, much more important is actually to understand what you can do when you do least squares with varying numbers of residuals. Let's take a look at it. Well, the most obvious thing to do is this is you take a plot, and you plot the number of regressors you take. That's the number of columns of a. And then what you do is you simply plot the norm of the residual. By the way, 0 has meaning, and it says, please fit my vector y with a linear – with a optimal linear combination of no columns. Well, you're at – your fit is gonna be there is no fit. And so your error is simply the norm.

So this, by the way, sort of sets everything here. So right at p equals 0, you get the norm of y. Then this first point here gives you exactly the optimal fit, or if you like, the geometric distance. This is the geometric distance from the point y to the line spanned by a1. That's what this is. And it drops here.

Hey, by the way, could that point be – could this point be here? No, not if your least-squares software is working. Could it be here? And when would it be there?

Student:[Inaudible]

Instructor (Stephen Boyd):x when the optimal x1 is 0, which would occur when, geometrically? It would occur when –

Student:[Inaudible]

Instructor (Stephen Boyd):– y and a1 are orthogonal. I'm getting a weird – did I say that right? Is it right? I'm getting some weird looks. I'm gonna blame you if I said something really stupid, and you failed to correct me. I think it's right. So that – it could be here.

So this drop, by the way, in statistics, they have a beautiful name for this. You'd say – you would say something like, "In this case, it expresses a percentage." Say something like this. If this dropped 15 percent, you'd say that a1 explains 15 percent of y. That's what you'd say. Not an impressive number. That's not something you'd brag about, probably.

This number is the distance from y to the span – to the plane spanned by a1 and a2. And you can see it dropped a healthy amount. And then this is – and of course, this has to go down, and so on. And that's it. So you get pictures like this. These pictures are extremely important. In many applications, you need to look at these because usually, this thing has something to do with the complexity of your model. And so you're gonna look at figures like – pictures like this.

Certainly, you would not want to fit a model with something more complicated than it needs to be. So we'll look at this in a very, very practical context, which is least-square system identification. This comes up in tons of fields. This is in economics. It's in controls. It's in signal processing. It is in finance. It goes on and – I mean, the applications of this just go on and on and on.

So actually, I once taught an entire class on, well, that, of which two-thirds of the class was based on that, so entire classes on this.

So here it is. Let's just say I have a scale or input and a scale or output. And I have an unknown system here – a dynamic system. And the system identification problem is the following: I give you a record – some observe data – input and output. So I give you – some people call that IO data – input/output data. And what I wanna do is I wanna fit a dynamic system model to this. So that's the picture.

Well, here's an example with a scale of u and y because vector is easy – vector case is easy to do. So one very famous model is called a moving average model. I think you may have seen this on the first homework or something like that. And it basically says this. It says, I will – my model is – and the model is with a hat on top – the model here is that the output is a linear combination of the current input – the input lagged one time instance and the input lagged up to n time instance.

So here, you have a set of coefficients in the model. That's h0 through hn. There's n plus one of them. And they're real, in this case, because you just scale it. So that's a moving average model. And it's a – the h's parameterize the model, they give you the coefficients in this moving average. It's a – I mean, if you wanna be fancy here, you could say it's a moving weighted average. But what everyone says is moving average.

Now, how do you set this up for some kind of – and your job is gonna be to pick h0, h1, and hn. Of course, you know by now that the – what you have to do is you have to wrestle with the matrices in the equations to make it look something like this, where the things you want to estimate or choose appear here. There's a matrix of known stuff, and then there's a result over here. And so I've done that for you. I've written it this way, and it looks like that.

By the way, this is kind of bizarre because normally, when you think of u as an input to a system, usually we think of inputs as appearing here. And you can write this equation a totally different way with the u's over here and the h – so lots of ways to write it. For what we're doing now, you write it this way.

Now, the model prediction error is this. If I commit [inaudible] – we have a coefficient y, y hat here is actually what I predict the output is gonna be. y is what I actually observed it to be. So the error is called – is – that's the model prediction error. It's just the difference like this.

And in least-squares identification, you choose the model, that is, the parameters that minimize the norm of the model prediction error. And the answer is the way to get these h's is this thing\that. That's how it's done.

So I won't even go into how that's done. You should know how that's done.

Here's an [inaudible]. Here's an input trace, like this, and here's an output trace, like that. And you wanna fit a model, some kind of moving average model to this data. So there it is. And [inaudible] kind of look at this, and you can say a couple things. You can say, "Well, it looks like the output is sort of a smooth version of the input."

Oh, by the way, one of the things you might also wanna do here is to – one possibility is in fact that these are completely unrelated.

By the way, how do you imagine that would come out, I mean, if you have enough data? How would that come out? What do you think?

Student:[Inaudible]

Instructor (Stephen Boyd):What's that?

Student:[Inaudible] model [inaudible]?

Instructor (Stephen Boyd):No, no, no, no, no, no, not – it's not the [inaudible]. Let's – no, that could be a problem. But I'm talking about even a more fundamental problem. The more fundamental problem is that this is the price of some asset here. And this is the temperature in Palo Alto. Now, by the way, they could well be related. But let's just imagine that, in fact, they are not. And I gave you the data. And I said, come on, you took 263. Let's – how – what would happen? What would suggest to you that in fact these are unrelated? What result? What would happen when you started fitting models to it?

Student: Equal weights.

Student:[Inaudible]

Instructor (Stephen Boyd):You can get equal weights? I sort of agree with that, but can you be more specific about the weights you might get, the h's?

Student:[Inaudible]

Instructor (Stephen Boyd):0. Basically, the optimal h's would kind of be extremely small. And your optimal error would be terrible. You would get a huge pile of data, go to this thing, find tiny coefficients, and say, "Well, I'm done. I just explained 1 percent of the data," meaning that basically, your guess is just 0 each time, or something like that.

Everybody see what I'm saying here? So that would be how you do it.

Now you had a – there's another problem. And that is that your data is not rich enough. You could throw in a few more regressors, and then you could actually get a better fit.

Okay, so [inaudible] how this works. So here's our example. And but when we work this out, we calculate the eight coefficients, and we get something like this. And a relative prediction error is .37. Actually, that sounds like that's not that good. It says, basically, that when each time – before a new sample comes, I can predict it through within plus/minus 37 percent. I mean, roughly, that's what this was saying. That doesn't sound like it's gonna be a good model. It's actually a pretty good fit. I'll show you in a second here. And then I'm gonna come back to the coefficient.

So here's what a 37 percent fit looks like. Here, what – the black is the actual output. And the y hat is predicted from the model. I think it's kinda fuzz your eyes a little. If you squint, it's really not bad. It's kinda getting the gross characteristics of the movements. There are places where the error's large.

Actually, in places like this, because the error is [inaudible] – the errors are actually large here. And they're large here for a while and so on and so forth. So that's what sort of explains this.

That – there's your picture. By the way, it's actually not too bad. I mean, if – for some applications, if you wanted to get a rough idea, this would – this is clearly giving it to you.

I have a question for you. Here, this last coefficient is pretty big. Actually, it's the largest coefficient. That should make you uncomfortable because it basically says that to calculate the current output, this model, with a lag up to 7, is heavily relying on what the input was seven samples ago.

It really strongly suggests you should increase 7 to 10 or 15. And let's see what happens if you get bigger. And that's exactly this business of growing regressors for this thing.

So the question here, then, is something like this: When you do a moving average fitting, how large should the model be? Now, the larger you make n, obviously, the smaller the predict here on the data used to form the model. That has to be because you just – your – you have a richer set of regressors. You're projecting on a bigger subspace. The distance can only go down. The fit can only get better. That's clear.

And if you – if what you wanted to do was actually just fit the data for the model, then you'd use the largest model you can possibly handle. I mean, there'd be no reason because the error just keeps going down.

And for the particular data example I just showed you, here's what happens. So this is it. This shows you sort of the fractional error, as you increase the order of the model. I think we were here, maybe. Is that right, 5, 6, 7. I think we were here.

Very interesting plot, very good plot. You should produce it, essentially, always whenever you do things like this. And there's even a concept of variable number of regressors. You should do this plot.

So there it is. By the way, this is carefully constructed to show you what it should look like in the cleanest, simplest, and best case. Now, this thing goes down all the time. We've already discussed that. But here, your eyes should be drawn to something, which is kind of obvious. That's why we made the data this way. And that is this guy right here.

And when you look at this, that's the traditional knee of the curve. And it basically says, somewhere at up till order 10, the models predict more and more. And above 10, well, technically, they predict more and more. And this is on the data used to model. But they start at – and what they start adding is very, very small. So there's some tradeoff here.

Oh, by the way, I should say that in statistics, there are very complicated – and signal processing and economics – very complex theories on how to choose the model order, very complicated. But basically, it's very important to understand it from a simple point of view, which is the one I'm talking about here now first. Later, you can go off and find out about the [inaudible] information criterion and blah, blah, blah. That's fine.

But for now, the first thing you do is just get it – how this works in the simplest case. So this would say something like n equals 10 is already coming out as a possibly good choice.

Now, the problem with this is the following: If in fact all you want to do in your model is reproduce the data you've already seen, then no one can argue against this. It's got a better fit, period. But in fact, we are creating that model probably to use it on data you've never seen.

For example, tomorrow, you wanna make a prediction about tomorrow. Or you wanna make a prediction five nanoseconds in the future. That's what maybe – this is the kinda thing you wanna do. What this means is you shouldn't – I mean of course, this is important, but you really should be checking that model on other data not used to fit the model. And that's a very famous method. It's called cross-validation.

In statistics, but in lots of other things, you may wanna zoom out a little bit here because it seems like [inaudible], but – or not, I guess. So cross-validation's very simple, totally obvious idea. It says that when you're fitting a model, you should actually judge the victive performance, not on the data that you use to develop the model. That's kind of obvious. Actually, if you're not doing well predicting on the data used to form the model, you don't need to cross-validate because your model is terrible. So you don't need to do – you don't need to go to this step.

So this is sort of if you're getting good fits on the data you've seen, and I mean that in like the c sense, meaning that if that's not the case [inaudible] what comes next in the – in that.

So here's a model validation set. Oh, the first one could've been one week's of data. This is another week's data. And here's u and y bar. That's some other recent data. And then what you do is you take the data that you – that the coefficients you got from the other data set, and you apply it to this one. And you make a plot like this. And it's quite pretty, quite sensible, and you see something here. This shows you the error on the modeling data. This has to go down unless you made a mistake. Has to go down.

This one here does not have to go down. Obviously, it doesn't. And this is really interesting. You see this? This says that when you use a fifth-order model to make a prediction – to actually now try it on next week's data or something like that. Or I guess it wouldn't be next week's data, or you couldn't validate it. So some other historical week of data, you actually do worse than if you use the fourth-order model here. So this can go down.

And actually, now you – this is made – this is a thing that should be burned into your brain. This is sort of the beautiful – this is what you're hoping for is something like it because now it's totally obvious. It's right in your face that probably the best model here has order 10. So this is a fantasy. This is what it looks like. You might ask, "Will it ever look like this when you do it?" The answer's, "No, it'll never look like this."

Student:[Inaudible]

Instructor (Stephen Boyd):Well –

Student:[Inaudible]

Instructor (Stephen Boyd):Why here? Because in fact, the data here really is – the data here was generated [inaudible]. By the way, it's [inaudible] with order of 10. That would be unfair. So it's well fit by order of 10. So – but I'm not sure I understand what you're asking. Are you asking what – why does – why did it work here?

Student: Yes.

Instructor (Stephen Boyd):It worked because the –

Student:[Inaudible]

Instructor (Stephen Boyd):Oh, no, it could've – oh, no, absolutely. You – I mean, you could argue that 9 is fine here. So by no means – in this case, I mean, you could even argue that 3 or something is a good order if your noise level is high enough. And if the level of prediction you want is just what I showed on the graphs over there, that would be fine. So I just wanna show you the idea here.

Now, of course in practice, it never looks like this, although it can sometimes look like this. This is the fantasy.

Student:[Inaudible]

Instructor (Stephen Boyd):Oh, [inaudible]? They're the same size. They're the same size.

Student:[Inaudible]

Instructor (Stephen Boyd):Sorry?

Student: If I have a function [inaudible], [inaudible]?

Instructor (Stephen Boyd):Right. So there's lots of ways how you do cross-validation to [inaudible]. Then you get – it's a huge field, I might add. And there's lots of ways, so in fancy studies where things really matter, like medical studies, they'll actually do things where you're not – I mean, they will insure – you will not ever see the validation data if you're the one doing the modeling. You're not allowed to.

There's a nice – there's a protocol. You develop your coefficients. You send them to someone else who basically had signed something that he will not share the data with

you, and you check it. [Inaudible] he'll check it on the other side of this opaque wall. And you'll simply say, "Here's your error." So you'll never see it.

So in some cases, this stuff is taken very, very seriously, I mean, for obvious reasons. In other cases, you might do this. You might take your data, chop it in half.

Or some people do this. They take two-thirds of it or something and fit a model. And you can do insane things. Of course, if you have these two data sets, you can develop a model from this data set and try it on the other one. Then you can do weird things. You can chop them up into different pieces. And you can develop a model on all the chunks except one and check it on one. So cross-validation, then, gets very, very complicated and so on. And you can think of all sorts of things to do.

And then the theorists come along, and it gets even more complicated. Then they'll be the information criterion and blah, blah, blah, and the Kullback-Leibler divergence. And it'll go on and on and on.

So very big topic. This is a basic idea. This is – I mean, this is the fantasy. This is the pure thing. This is the way it should look. Of course, it'll only look like this in this class, but still, it's not that – it's not impossible that you will sometime do some fitting, and it will look like this. It's not impossible. Actually, it's likely in this class, but I mean, real – some real fitting.

Was there another question? You also asked about the size of the two sets, right? If you actually give the errors as RMS errors, then the size of the two sets now – at least the first order – does not scale your error. And you can actually talk about it sensibly. So here, they're equal, so norms are fine.

But if you – if you had – if your model data was 100 points, and your validation was 37 points, you would definitely need to look at something like an RMS error to be sure you're comparing the right things.

So – but I should add that when you do this thing – this phenomenon – and in fact, this example's a little bit off. I can tell you why in a minute. This phenomenon here where you get – as you get a higher and higher order model, and your modeling error residual goes down, but actually, your prediction ability's getting worse, the – this is called overfit. That's the name for this. It means you overfit your model. Another way to say it is you're fitting noise or something like, so that – and that's essentially what's going on.

By the way, sometimes it's not graceful like this. This can go like this. This goes – that would actually – in fact, I should probably redo the example to make the crime associated with overfit have a higher penalty and have this thing go like that.

So we'll look at that. We'll – [inaudible]. And let me explain what I mean by the [inaudible]. First of all, let me say what we just did. We just looked at the [inaudible] sort of x minus y, like the sum [inaudible] like this. So it looks like – oh, there we go. So I did

this. And what we were doing is we were looking at controlled bits of a. So this comes up in things like that when you add one more [inaudible]. So that would be – that's where this is at in adding regressors. Come up in [inaudible] things and things like that.

Now, we can also look at another [inaudible], and that is where the a in the least-squares setup is actually growing not by adding columns but by adding rows. So that's the – that's instead of growing sets of regressors – growing sets of [inaudible] that actually comes up in a bunch of cases. So let's see what that is.

Now what we're gonna do is we're gonna write least squares in row form. In row form, you write norm ax – well, we write in norm – we write norm ax as a vector, which is whose ith entry is the interproduct of the ith row of a with x. That gives you this form here. Hopefully, it's identical to the other formula. It's just written it out to [inaudible] those [inaudible].

Now, the rows in a, if you're doing estimation, they correspond to measurements or data. That's what they correspond. [Inaudible] other applications. They have other meanings and things like that. But here, they generally correspond to measurements in a sensing situation. And so each row of a is a measurement of some parameter x that's unknown.

Now it's interesting because you start asking questions like this. You say, "Given a set of measurements," you say, "please guess what y is." By the way, this kind of has to start with a full rank and skinny.

And by the way, later, we're gonna ask questions, like if someone walks up to you and says, "Oh, I have one measurement. Please guess these ten numbers." For the moment, that's off – we can't do that.

So in this case, it's gonna work like this. We will only consider the case when you've already got enough measurements that that matrix is full rank. That means that if you're estimating n parameters, you have at least n. If you have n, it means they're also independent. So that's how that works.

So here, you imagine x is a vector to be estimated, and these are measurements. And the solution of this, that's a – this is assuming this thing is invertible. That's, of course, this is a transpose a here if you write it out row-wise. I wrote it out – the rows of a are the ai transpose. That's why it looks like ai transpose here.

So once a becomes – as you grow a row by row, once it becomes full rank, rank n, it will stay rank n, of course, as you keep increasing it. And that's when this becomes invertible. And this is the least-squares solution written out. This is a very interesting and very valuable way to think of the least-squares solution. Looks like that.

It's actually kinda cool, and you can ask all sorts of things here. By the way, the same story – well, you can actually do things like, say, well, let's look at cases where this would come up. This would come up this way. Suppose we're estimating a parameter x,

but we make measurements once every second. Oh, let's suppose your measurements actually are GPS pseudo ranges or something like that. So you're getting those once a second. You're getting a GPS pseudo range. But x is constant. x is, well, in that case, it's three numbers. Actually, it's four because it also includes the absolute time.

So x is four numbers, which is your exact position and location. And every second, you get a new measurement. Now, the satellites in GPS have moved slightly, so actually, the rows of the a are not the same. They're actually slightly different – more than slightly different, actually. They're a bit different. And so what this would do – the reason this would come up is because at each – at every time you get new information, you would wanna have a new estimate, in fact, a sharpened estimate, an improved estimate based on the new data.

You can't use the other 50 rows that are coming because that's in the [inaudible], and [inaudible] to estimate your position now. So that's the idea here.

Now recursive least squares is something for that. It's a simple version of something called a Kalman filter, which is for another class. Actually, once you understand this, that's not particularly complicated. So here it is.

Instead of calculating this, and then we're gonna calculate this for each m. And it's a beautiful thing. This is basically saying, "Here's my estimate of the parameter." And you can actually plot this in time and things like that. It'll shift. You'll say, "I think I'm here." Another measurement comes in, and it'll more a little bit more and more and more. It'll move around. And it'll just get a better and better estimate. Of course, except in the most extreme cases, you know now this can be calculated with very, very fast.

But these things are coming at you, let's say at a megahertz, so new examples are if m measures microseconds, that's maybe a different story. Actually, that's not even fast enough, but if it's suitably fast, you'd have to do this – well, that one should be enough to make this worthwhile.

Okay, so the way you do this recursively – actually, let me just explain what the idea is. This least squares now, I'm gonna be very, very crude here. This least-squares estimate is [inaudible] that's a vector. But I'm just gonna be crude and say it's one sum divided by another.

You're not allowed to do this, by the way, what I'm doing right now. Or you can do it, but just don't ever write it down or have it recorded. I'm allowed to do it because I said explicitly – end nonsense. And then I'll put the end xml tag and [inaudible] when I finish this discussion.

But for the moment, this is, again, nonsense. This is, roughly speaking, a ratio of two sums. Of course, that's a matrix inverse, so it's hardly a ratio. But it's a ratio of two sums. And the sum [inaudible].

Now, if I walked up to you, and I said, "I need to calculate something that's a ratio of two sums," that's – or if I have a sum, and I say, "Here's a new sum and," how do you calculate the new sum? You don't go back and re-add everything. You simply add the new thing to it. Everybody see what I'm saying?

So basically, you have something that looks like this. And when a new measurement becomes available, you take this thing, and you plus equals ym, am. That's what you do. You add – you accumulate it. That's a matrix.

So what? You plus equals, and a diad, which is am, am transpose. And what this means is your storage is now bounded. It said that as you do this, you don't need to store all of these things. You basically keep track of a running sum. Your storage is absolutely bounded.

So this is this thing. That's what I was just talking about here, is you simply add these up. And already, I mean, already have an algorithm, which [inaudible] interesting because it says, basically, you can do this – you can run this out, [inaudible] equals 100 million or something like that. It's just completely linear in m. The storage is absolutely bounded.

You have to store a symmetric matrix of size n by n. And you have to store an n vector. And that's it, period. And whenever anybody asks you, "Hey, what is [inaudible] so far?" simply calculate this. So that's how that works.

Now, calculating this, it depends on the timescale. It depends how frequently you calculate it. So you can actually [inaudible] match them, or you're simply accumulating these all the time. And then only occasionally do you get a query as to what the estimate is. When you get the query, you calculate this. Or you can imagine a system where you actually have to calculate xls of m for all m, for m equals 1, 2, 3, and so on.

But here it is, a famous formula. This one may be really is actually still useful – [inaudible] from the '60s, very useful. This one may actually still be – is still useful. And it's this. It's – and it's a good thing to know about. It's very famous. In this [inaudible], you have to calculate the inverse – actually, you don't calculate the inverse, but you have to solve an equation with a matrix, which is the previous matrix plus a rank 1 term. That's a diad. It's rank 1.

So by the way, there's a name for this. This is called a rank 1 update. If there's a minus sign here, it's actually called a rank 1 downdate. So – but that's – [inaudible] I – it kinda makes sense. So this is called a rank 1 update. If you go to Google and type "rank 1 update," you'll get tons and tons of things. Or you'll actually read papers, and people will say things like, "Here's our method, blah, blah, blah." And they'll say, "Well, of course. By exploiting rank 1 updates, we can do this, and blah, blah, blah."

If the numbers here are tiny, like 10 or 100, the whole thing is silly unless it's gonna be built-in hardware and run at a gigahertz. So there's no reason to have this. But if these

numbers are getting into the thousands and so on, and it's gonna run at kilohertz or something like that, these things make a huge difference.

So that – this is a rank 1 update. Well, there's actually a fast way to calculate the inverse of a matrix if you've already calculated the inverse of this matrix, and then you do a rank 1 update.

So this is the so-called rank 1 update formula. Actually, it's got a zillion other names, but this is a good enough one. And it's – the formula's this. It's p plus a, a transpose inverse is p inverse minus, and then it's 1 over 1 plus a transpose, p inverse a times this thing.

Now, notice that if you've already calculated p inverse – in fact, you don't even really have to calculate p inverse. Look, suppose you calculated p inverse, a is [inaudible] to calculate. And that's an outer product here. You've already calculated p inverse a, so in fact, you only have to calculate p inverse a once. And then you use it here, here, and here with the transpose there.

So this is very, very straightforward. Notice that, by the way, the update to the inverse is a rank 1 downdate. That's a scaler. That's a vector times a vector transpose. That's a rank 1 matrix. And there's a minus there. So that's – this is what people will call a rank 1 downdate in the inverse.

By the way, it kinda makes sense when you add something to a matrix, and you invert it – invert sorta when things get bigger, the inverse makes the inverse get smaller. So the idea of the rank 1 update should lead to a rank downdate in the inverse, so it sorta makes sense. You can include that in the nonsense, by the way.

Oh, sorry, I forgot to say this. I'll declare it now. Oops. It's the end of nonsense. There, okay, so it's over now. What I was just saying was not nonsense. The only last part was vague.

So what this does is this gives you an [inaudible] of [inaudible] n squared method for computing the inverse of the second one of from the first. And If you did – [inaudible] in fact, n cubed. Of course, this is – doesn't really make any difference unless n is 10 or 100 or 10,000. Then it starts to make serious difference. Equals 1,000, it makes a big difference.

Now, you might ask, [inaudible] or whatever. I'll show you the final artifact. To verify – we just have to verify that if I multiply this matrix by this matrix, we get i. And the way you do that is this – that's just a sum of two things. You multiply that, and then you get four terms. So the first one is p times p inverse. That's i. Then you get p times this thing over here. That's this term. Then you get a [inaudible] times p inverse. That should be here somewhere. There it is. It's right there. And we should have one more term, which is a, a transpose times this stuff over here. And I guess that's down here.

And now various things start canceling. There's a p and a p inverse that go away. Then you look at this, and you realize this is a times p inverse a. Let's see what – down here, let's see what's gonna – what is gonna cancel here. Well, this one goes there. That one goes down there. Ah, these are both a times p. These are both a. I transposed that a, a transpose, p inverse. That's this thing. And it's the same thing you're gonna get over here. You look at those two terms, and they actually cancel like this. Actually, all three of these go together because that's – well, the coefficient is 1 minus 1 over 1. I'm not gonna do it. I just leave it there. It works. And you get i.

So that tells you that this matrix is the inverse of that matrix, and of course, vice versa. Now, you might add, "I wonder if – did anyone come up with that?" Well, it wasn't this way. It – I mean, it's not this way. It's the way these things usually work. You work it backwards, and then you reverse it when you explain it to someone, which makes you look smarter, I guess. So that's standard procedure.

So we'll – there's a question.

Student:[Inaudible]

Instructor (Stephen Boyd):I'm sorry, does what play a role?

Student:[Inaudible]

Instructor (Stephen Boyd):I'm sorry, there was a noise. I didn't get it.

Student:[Inaudible]

Instructor (Stephen Boyd):Yeah.

Student:[Inaudible]

Instructor (Stephen Boyd):You're right, all of these. Oh, I'm sorry, no, no. It's okay. Let's be very careful about it. Thank you. m is irrelevant. m is just – when you're calculating the invert p, m is t, if you wanna think of this as time. So this is fine. Oh, no, no, it's all right, you just confused me. Yeah, I'm okay. You're not.

All right, let's – we'll both take a deep breath. Now we're gonna go over it. n here is a parameter – is a problem size. It's the number of things you're estimating. It's the size of x. m is something like a time index in most applications. It's a time index. So m is irrelevant. It doesn't have a value. It's – m could be 30 million. It could be minus 50 plus 200. It makes no difference whatsoever.

The question is how much work you do. Of course, it's interesting to say that it has nothing to do with m. We knew that because this method here – each increments, the amount of work is the same. You increment p. You [inaudible] q. So m being 10 to the eighth doesn't make any difference here.

Does this make any sense? No, I can tell I didn't convince you. We'll –

Student:[Inaudible]

Instructor (Stephen Boyd):Let me –

Student:[Inaudible]

Instructor (Stephen Boyd):Oh, oh, oh, here. You want here.

Student: Yes, [inaudible] p [inaudible].

Instructor (Stephen Boyd):If p is what?

Student: If p is 12 [inaudible].

Instructor (Stephen Boyd):Right. There's a similar formula that is valid when p is not symmetric. You can work it out if you like. No, you can, really. You can guess it. Did you have a question about that, or –

Student: No.

Instructor (Stephen Boyd):Okay, just – okay, yeah. So this is the formula when p is symmetric. Now we're gonna [inaudible] to [inaudible] and things like that. And the first topic involves a multiple objective [inaudible]. And let me just put – let me just tell you where we are and where we're going.

So least-squares methods, they're widely, widely used. They don't always have the name "least squares." It depends on the field. They might use some other name for it. They're widely used. But it's actually a couple of tricks. You learn just a couple of tricks, and you will be in very – you will be very, very effective at using least squares. Actually, just the basic least squares is an awfully good start. But there's a couple more things you need to learn about.

One is called regularization. We're gonna get to that today. Another is when you – if there was a special case of multi-objective least squares. So we're gonna cover just a couple of things, and you will be in very, very good shape for actually using least-squares methods. So let's look at multi-objective least squares.

So in a lot of problems, you have two – we'll focus on two, but in fact, you can very generally have more objectives. So it might be something like this, and in the case of two, you want one objective, let's say ax minus y, small. Now, so far, the only thing we've looked at is how to choose x to make j1 as small as it can possibly be. That's the only thing we've been looking at, just that.

But here we have a second objective, which is at the same time, we want another least-squares cost, norm fx minus g small. So we have two. Now, if they're the same thing, if a is f, and g is y, these are the same thing. And it's silly, by making one small, you make the other small. And there's really no competition between them. I mean, there's no issue. But in general, they're competing. And that means that basically, you – to make this smaller, you're gonna have to make this bigger, and vice versa. That's what competing means.

Now, very common example is this, and it's really common, and it just looks like that. So basically, in this case, what you're saying is, "I want ax minus y to be small." But at the same time, I want you to do it with a small x.

So basically, you're saying, "I wanna do polynomial fitting. I want the fit to be good." That's what this is. But you know what? I don't want insane polynomial coefficients with coefficients like 10 to the 8 in the coefficient. In fact, I don't even want 100 in the coefficients [inaudible] polynomials.

Why? Well, because you're using this thing, for example, to interpolate or extrapolate or something like that, and the larger these coefficients, the more trouble you're possibly gonna get into when you use it for that. So that's the basic idea. And that's the most classic case is where you wanna trade off these two.

Well, conceptually, the right way to think about this is as follows. You think of the plot of achievable objective pairs. And so what this is – and by the way, x here is r100, say. So x is an r100. y is an r200, and g is an r300. It – none of this matters. I'm just saying these are big dimension things. What this is a picture of is the following: Basically, for every x and r100, you simply evaluate j1 and j2, and you get two numbers. And you plot them in r2. So it's not a plot – that's [inaudible]. That is not x1. It corresponds to x1. So in fact, to really write that label correctly, I should write it this way.

That's j2 of x1, j1 of x1. That's really what that blob point is. This is – this just means that's the point that corresponds to x1, and that's x2. So that's the picture. Now, the picture looks like this. And actually, it's a real interesting picture, and it's one [inaudible] in a lot of context, it won't – the stuff I'm about to say is kinda obvious. Everybody knows it, but it's actually very worthwhile to have seen it once or twice. It works like this. You have two objectives, j1 and j2. And now we can actually talk about various things here. Look, in fact, let's talk [inaudible] all now. So I'll clean it up.

[Inaudible]. You might ask, "Why is j1 vertical and j2 horizontal?" And the answer is, I don't know because [inaudible] for some reason. So let's look at some points here. So suppose somebody proposes an x and gets this point. And someone else proposes another point, gets that. So let's call this x1, and this is x2. What can you say about x2 compared to x1 in the context of this problem? x2 is better, period, even though – and why is it? Because it beats – x2 beats x1 in both objectives. So if one is – if j1 is fit, and j2 is fit, then x2 beats x1 unambiguously. It gets a better fit, and it's smaller. Everybody got it?

And in fact, if I draw this region down here, so anything down and to the left of where a point comes up here is actually the set of points that are better, unambiguously. And it doesn't matter how much you care about j1 relative to j2. Every point in here is better than that one because basically, you beat it on both. Or if you're up here, you meet it, and let's be more precise. You meet the same objectives as the other one in both, and you beat it in at least one. So that's better.

So all these points are better. How about these points up here? What can you say about [inaudible] compared to x1? [Inaudible], so up in this quadrant compared to a point is unambiguously worse. And over here is unambiguously better. And now what about this guy? How would you compare that point to x1? What would you say? Say it's better? No. Can you say it's worse? No. They're not – it's not comparable.

So what you would say about x4 is you'd say, "Well, it's better in j2, but it's worse in j1." And someone says, "Yeah, but is it better?" And you'd say, "I don't know. Depends on how much you care about j1 and j2." The point is, you can't say. And the same is true here.

So the first and third – is that what they call it? Are those the third quadrants. Yeah, so the first and third quadrants are the unambiguous [inaudible] the other period. The second and fourth quadrants on a plot like this tell you that's ambiguous.

So – all right, so let's go back to this thing. So these are all points. This point here [inaudible] r3. x3 is a choice, but look at this. If I look – let's look at the things that are better. Unambiguously, but in x3, I'm seeing there's a lot of other points. In particular, x2 beats x3 on most things, and therefore, x3 would be an exceedingly poor choice if you cared about j1 and j2.

The thing that kind of should become obvious here is what you really want is a point. Oh, by the way, there's a name for this in economics. They say that x2 dominates x3. So it's just better. And you'll find other words for it in other fields, too, but it dominates.

So what you want is a nondominated point. So you want a point that's achievable so you can say there – the fact that it is clear says that there's no x that achieves that pair of j1 and j2. However, if you look at x2, and you say, "What's better than that?" It's everything over here, but there is no – there are no points over there. x2 is not dominated. And there's a name – it's got all sorts of names. One is it's called a pareto opal [inaudible] – it's pareto optimal. That's [inaudible] years old. [Inaudible].

This curve here is the set of points that are pareto optimal. They're nondominated. These are the ones where if you were only told that you should minimize j1 and j2, you cannot be fired for. The nonpareto points you can be, unless there's another j that you're not writing in here, like for example, a time crunch or something like that. But that's the way it works.

So I mean, these are sort of obvious things. Actually, it'll get not obvious pretty soon. So this is, by the way, called – some people call it the optimal – this is called the optimal tradeoff curve of j1 versus j2. That's one name for it. It's the optimal tradeoff curve. It's also the pareto optimal points, all sorts of other things.

Let's see. So I think we've talked about all of this. By the way, if you have three objectives, you have a pareto optimal surface, and now we're talking about octants. So you have pareto optimal surface, and you have three objectives. And it's – or people call it the optimal tradeoff surface. By the way, people do plot these. You plot it by slices, very typically.

I should – these ideas just – they go – I mean, you should actually incorporate these ideas. They're so obvious that you probably already had all these ideas. But it's nice to know that there's a language for it and that people have been thinking about this and talking about it for a couple hundred years.

So for example, if you do circuit design, you, generally speaking, have a tradeoff between, say, power and speed. And you think – you should think in these terms. Basically, a point up here is – the technical name for a point up there is a bad design. That's the technical name for it. These are good designs. This would be called – if this was – if this is power, and this is delay, then these – this would be the optimal tradeoff curve – optimal power delay tradeoff curve, for example.

And it's very interesting. I mean, it's very interesting. It's very useful. And in fact, you should probably do these – any time someone tells you to do one thing, you should probably do is do tradeoff analysis anyway. So we'll get to that.

Now, this brings us a question. So as we stand right now, by the way, you cannot argue that x1 is better or worse than x2 or vice versa. They are absolutely not comparable. All you can [inaudible] x1 is better on j2, and it's worse on j1. And if someone says, "Yes, but is this better?" You say, "Depends. Depends how much you care about j1 and j2."

So the most obvious thing to do is to form a weighted sum objective. So that's done other ways, but here's the most obvious one. You form a composite objective, which is j1 plus mu times j2. mu is a parameter we're gonna use. It's positive, and it gives the relative weight between j1 and j2. So that's what we're gonna use. It's a composite objective of this.

By the way, mu has a lot of meaning here. For example, it is often the case that these two measures have different units. They can have different units. This – well, obviously, in a circuit – I mean, circuit's not gonna be least squares. But anyway, let's imagine that it was. You'd have power and delay or something like that. These are in watts and seconds.

So you can – it's actually very useful, and maybe in this case, watts squared and second squared. It's very useful to think of mu here as actually containing the units that translates both of these into units of irritation because that is, after all, what an objective that you

minimize is supposed to be. It's a unit of irritation. It could, for example, be units of dollars or euros.

That didn't come out right. It's the payment you're gonna make. Now it came out right. So it's a payment. Then it's units of irritation. And so here, mu maps the units of the second objective to – makes it comparable with the units here. I'm gonna give you an example.

Suppose we're gonna do a function fit with – of some data. So you have some basis. Everyone agrees that's the – we should use the prolayed spheroidal harmomics for estimate a [inaudible]. I have a ton of data acquired at considerable expense and trouble, and I wanna fit a small model to it. Now, of course, one thing is gonna be the fit. You wanna fit the data. But the other one might be the smoothness of the data, so – I mean, the smoothness of the model. Sorry.

So I don't wanna get – I could get an excellent fit with something that wiggles like crazy. It would wiggle to fit your data, which might have no noise in it. But no, I want it – so I wanna trade off smoothness here with fit. This is like classic example of multi-objective fitting.

Now, what – these have actually kind of – they actually have different dimensions because smoothness is related to a difference or a double difference divided by a spatial difference or something like that. So this might be – let's suppose we're actually – what we're trying to fit is a temperature distribution. This will be – if you take this value, that's gonna be in degrees centigrade squared. This is gonna be in degrees centigrade per meter quantity squared.

Therefore, mu has units of meters squared. Thank you. I was gonna say it, but thank you. [Inaudible]

So that's just – so you think of mu as translating. Now, there's one more thing I wanna do before we quit today, and that is to look at what j1 plus mu j2 does on a j1, j2 plot. Well, if I look at the points where this weighted objective is equal to [inaudible] alpha, these are actually just lines with a slope minus mu. That's what I'm doing.

And so in fact, if I ask you to minimize j1 plus j mu 2, the visual – it's very simple what you're doing. Again, you don't have this, but you conceptually, if someone gave you [inaudible] achieve, [inaudible] what it means, and they said, "Please minimize j1 plus j mu 2," what you do is you take things of [inaudible], and you move them down [inaudible].

This is lower – this is when you move in this direction, the line – actually, there's a name for this. It's – I just forgot the name. It's a beautiful name in – it's the equivocal set or something like this. It's the things you care about equally for in economics. What is it?

Student: Level set?

Instructor (Stephen Boyd):Level set. It is a level – it's a level set of j1 plus mu, j2. But there's an economics – there's a beautiful name. It means you're just – basically, it means –

Student: Indifference curve.

Instructor (Stephen Boyd):What is it?

Student: Indifference curve.

Instructor (Stephen Boyd):Thank you. Oh, thank you. [Inaudible], this is an indifference curve. It basically says if you [inaudible] that, that, or that, [inaudible], they look – they're different, but they're cool for cool by me. So this is an indifference curve, meaning you don't care which it is on there. But you do care the lower you go.

So the point here is, you take this thing, and you move it down like this until you just touch here. And that is the minimizer of j1 plus mu, j2. That's the picture. And we're out of time, so we'll continue on Thursday.

[End of Audio]

Duration: 77 minutes