ConvexOptimizationI-Lecture07

Instructor (Stephen Boyd):Let me start with a couple of announcements. Not announcements. Actually, one was we're interpreting relative silence from the class as an indication that for example, CVX is actually installing correctly and that everything is working. Now, there is another interpretation and that is that no one has started homework three yet. So if you find anything, any inconsistencies in anything, let us know. We already had a couple of typos in the user guide pointed out, but if you actually find errors otherwise just feel free to let us know. Actually, it's a good way to get it all debugged very well. Now, I have a very strange question for you. The question is this. It sort of came up in office hours and then I talked with the TA's about it. Now, hear me out before you laugh or start throwing things or anything like that. The question is this. We think we know the answer. The question is this. Would you like to have a midterm?

Student: No.

Instructor (Stephen Boyd):Okay. Okay. Actually, it's an honest offer. And let me just say what it is so this is just if you want and then you can say no again, but then at least I'm saying something very specific so here it is. Today, we're gonna start basically the last theoretical, I mean, the last sort of theoretical concept in the class, so that's duality. There's a whole section on algorithms and things like that, we'll get to that later. But then there's just gonna be a lot of applications and things like that and just doing this stuff. So in a week or two we'd be in a natural place where sort of the core theory is basically that it's all done. That's it. It's done. It's more of the same, a lot of applications, things like that. And so the idea is we could have – I'm just saying we could, this is an offer, we could do something like have a boring conventional style midterm for an hour and half somewhere and it would be these boring problems like which of the following functions are convex and all that stuff. But then the final could be – which will naturally be a mixture of – it'll have a handful of these things, but then mostly it'll be on problem formulation and things like that. What do you think?

Student: No midterm.

Instructor (Stephen Boyd):All right. All right. That's fine. The TAs predicted that by the way. Okay. All right. All right. That's fine. All right. Well, let's continue. The next topic is generalizing the quality constraints. By the way, as I said, I know we're going quite fast over this material. We're going at about twice the speed I think we should be going, but that's fine because things like the generalizing the quality – we're going to be using these in applications basically from now on. So you're gonna weeks and weeks of seeing real problems that use these things so if it feels like we're going too fast right now we are, but you'll have four weeks in the context of applications to absorb all these things. So generalizing equality constraints. So here we're gonna generalize the inequalities, by the way, later, we're gonna generalize the objective. That's actually, in some ways, more interesting or it's a more significant generalization. But right now, we're gonna generalize the inequalities to be vector inequalities. So here, FI of X until now has returned to scaler and it was FI of X is less than or equal to zero, we're gonna

have FI of X return a vector, that's a vector inequality and it's defined by this cone KI. So in this case if FI is KI convex so it's a convex function with respect to this cone, this generalize and equality, then this is called convex problem with generalize and equality constraints. And you can generalize almost everything you've seen to this. For example, the set of optimal points for this problem is convex and all these kinds of things. Everything just goes through here with this. But there's a special case that's gonna be more important than the general one, and it turns out it's interesting. It's a very special case, but it turns out to be, in some sense, also the most general case and that's this. It's a so called conic form problem and you'll hear a lot about this. I don't think it's that interesting and I'll tell you why in a minute. Let me show you what is it. You'll hear a whole lot about it.

It's a very special case. Everything is affine so it's equivalent of a linear program basically. A normal optimization problem, if every function is affine you have a linear program because you minimize and affine function, subject to affine functions less than zero because you're conventionally called linear and inequalities. Affine equality, that's just conventionally called linear equality constraints. So here, everything is affine. You minimize a linear function. Now this is a very, very interesting inequality. This is affine and this just says that FX + G is less than or equal to zero with respect to this cone. That's all. So this is called a conic problem, a cone problem, something like that. You can type this into Google, a cone problem, and you'll get tons of stuff. I think it's interesting because it sort of generalizes the idea of linear programming to cone programming. If this inequality is with respect to the non-negative orphaned, so recall that if we erase this, that's the default. The default inequality between vectors with respect to R + and it means component wise and inequality. That's a general LP, that's a general linear program. So cone programming, I think it's caught on for several reasons. It is actually interesting to look at it, but it's caught on for a lot of reasons. First of all, if you're trained in traditional operations research you grow up – by the way, which is not me just so you know – but if you are trained or subjected to, maybe that's another verb one could use, a traditional training, everything you'll hear is about linear programming. The nice thing about cone programming is all you do is you drop this little K down there. By generalizing the cone from the non-negative orphaned you now have something that looks like linear programming so if that's your training and background it's deeply familiar, you've had three classes on it and now all you do is you generalize this and it means that everything is gonna be familiar and so on and that's what happens. A lot of things are familiar.

That's the first reason I think it's caught on. But the other is also interesting and you have to watch out for it, it's this, it's caught on because in fact the most highly developed solvers – we're gonna go in some detail in this in the last third of the class – the most highly developed solvers for convex optimization problems actually are conic solvers. So people who work on solvers naturally think that cone programming, conic programming and things like that is this huge important topic that everyone would be interested in. Actually, like linear programming, it's not. Okay. It's actually not. It's very important that if you worked on these things, fine, but for people who use these things, it's not that important. It's important in the following sense. There are a lot of interesting problems people really want to solve that transform to conic problems, and then, all the people who

worked on the cone solvers can use their super fancy primal dual homogenous methods to solve them, but in some sense, that should not be the business of the general public. That should be the business of this small group of people who worked on cone solvers. I'm saying this by the way because there's a lot of people who worked on cone solvers who will see this tape and I just wanted to insult them, but that's all right. I hope it worked. I'm sure it did actually. All right. So this is a conic form problem and in fact, the one you will see most – there's a question?

Student:I'm wondering why doesn't I [inaudible] on the K in the first – is it because you can do different cones in the same set of constraints?

Instructor (Stephen Boyd):Yes. So the question was why the I in the K and the conjecture was it's because you can have different cones reach constraint.

Student: You can generalize it to make it –

Instructor (Stephen Boyd):And my answer was yes.

Student: It seems like you could generalize it all to one cone, the highest order cone or something like that?

Instructor (Stephen Boyd):Yeah, you can do all sorts of things. You could lump these all together and make it one inequality with one cone, which is the direct product of all the cones. So you could do lots of things. Okay. Now, in conic program what you'll see the most is something called semi-definite programming. And this is actually interesting, well, it depends, what you'll actually see is the combination of two types of cone programming which you'll see in a minute, which I'll talk about in a minute. So semi-definite programming, this is relatively new. So this is maybe 15 years old, but like everything else can be traced back into the 60s although people had no idea how to solve SDPs. So as a specific example I worked on SDPs before, in fact, the name had been fixed in the context of control and things like that and we were very, very happy when we could solve tiny small ones in an hour. Boy were we happy. But it didn't occur to us that you could solve them a thousand times faster and things like that and that the current state now would've been just unimaginable 15 years ago. This has sort of made the transition in the last few years from this kind of esoteric thing that a few people in research worked on to just kind of it being all over the place. It's very fashionable. It's also used commercially I should add in areas of finance and stuff like that it's kind of widely used. So semi-definite programming or SDP is very, very simple. It looks like a linear program except for one generalization. So we minimize linear function, linear equality constraints – that's so far linear programming. Here's the difference. There's an inequality and it's a matrix inequality. These are symmetric matrix's here. So this thing here is called an LMI, or linear matrix inequality. So that's what this inequality is. It's quite sophisticated. A matrix inequality is very complicated. To say that a matrix is positive semi-definite or negative semi-definite here is actually a very sophisticated inequality so from that point of view, they're right, it is sophisticated but sophistication doesn't come because you have a name attached to it or anything.

So this is an LMI and that's it. There's an interesting thing. Here as drawn there's only one LMI, but of course, you could have multiple LMIs, but the fact is that if you have a bunch of LMIs like this, if you have two, you can lump it together into one and you lump it together into one by simply making a block diagonal matrix here with the LMIs because if I have a block matrix and I say that the block matrix is negative semi-definite, it just means basically each block is negative semi-definite. So that's why we can, without lose of generality, work with just one LMI in general. Okay. Now, semi-definite programming is a generalization. Actually, everything you've seen so far except geometric programming, so let's see how. So it's a proper generalization of linear programming, so how do you represent this linear program this way. This inequality here, that, this one here is a vector inequality so this means element wise. Okay. How do you write that? Well, it's kind of stupid. If you take diag, this is the diagonal matrix with this vector on the diagonal, a diagonal matrix is negative semi-definite if and only if all the entries are non-positive. These are interesting for lots of reasons. The first is just sort of theoretical. It's nice to know that you have a problem class that subsumes other common problem classes. It's nice to know that in fact you didn't really have to know about SOCP quadratic programming. Is there a question?

Student:[Inaudible] problem solver or with a [inaudible]

Instructor (Stephen Boyd):That's a great question. So it depends. If the SDP solver – we're gonna talk about solvers later in the class, it depends on how sophisticated the solver is. If any modern SDP solver – if it handles sparsity and it's a – let's look at AX minus B here, diag minus B and realizes it's diagonal, it will have no loss compared to solving it as a linear program. So that leads me to my second point. SDP embeddings actually also have substantial practical use because if you write an SDP embedding of a problem it means that that other problem class can be represented in an SDP and it means that one SDP solver can do everything for you. Now, very roughly speaking that's what CVX does.

Student:[Inaudible] use that in their problem solver or [inaudible]

Instructor (Stephen Boyd):Nope.

Student: Is it because the [inaudible]

Instructor (Stephen Boyd):Absolutely not. Because it's gonna be a mixture of LMIs, all these constraints and it cannot call an LP solver, does not call an LP solver, will not call. So it's absolutely not the case. For example, in CVX or any of these other tools if you write something that happens to be an LP, it doesn't not call a different solver. It calls exactly the same solver as if you'd written an SDP. Okay. So no difference. So the point there is an SDP embedding actually then has a practical use and it's so expressive. SDP is so expressive that it can be used to represent a huge number of constraints and functions and things like that. We'll talk more about that later in the class. I should say also that it's come up now in theoretical computer science. I think it's in lots of area. Actually, I think we just found out that SDP traces back with a different name into some calculations in

physics and chemistry in the 80s or something like that and it goes back into control in the 60s and things like that, so it's got a long history, but that long history didn't really particularly come with good computational [inaudible] so it was just theory. That's changed. SDP is now not just theory as I guess you all well know by the time you finished homework three, you will have solved, whether you know it or not a lot of SDPs. We'll get to that later. Okay. Here's a more interesting embedding. SOCP, so second order cone programming, the question is how do you represent this as a linear matrix inequality? Now, this is highly non linear and it's non differentiable and the answer is this and we'll see more about this later and in fact we have something queued for you, not on homework four, but on the next one, I think. This is a matrix here, which has a block – this one one block of the matrix is a scaler times the identity. Now, actually I don't know how many people remember about sure compliments, but let's just say we the teaching staff will remind you about sure compliments soon enough, but a block matrix is positive semi-definite. Now, this is rough so this is not quite right. It's positive definite if and only if the diagonal entities are positive and if something called the sure compliment is positive and the sure compliment is this minus this times the inverse of that times that is bigger or equal to zero. Now, for this particular matrix, that says CI transposed X + DI like this – let me put these together. Maybe if you do it on the pad here and capture both the big LM on this one and that one. There we go. If you can just capture this and what I'm writing here. It's this must be bigger than or equal to this thing AI X + BI transposed times the inverse of this guy, well, that's I inverse so I'll just leave as I, sure, I'll just put I inverse here, why not, and then that's a scaler so I'll put that down here. I'm using a loose parson here.

And then times AX AIX + BI and that is that term. Okay. Everybody see this inequality? So this one has to be positive because it's the last diagonal entry in a positive semi-definite matrix. It can't be negative. That's for sure. So this one I could square and put it over here. Okay. Or I can multiply both sides of this inequality by that. I can get rid of all of that. I get this. Now, I take the square root. When I take the square root, I have to be very careful. I can only take the square root here if I know CI transposed X + DI is bigger or equal than zero, but it is and I'll get exactly this second order cone constrained so this is an example of how this works. That's sort of the idea here. Okay. Let's look at some other problems. Here's one. It's eigenvalue minimizations. So here, I have a matrix which is an affine function of a vector X, so an affine matrix valued function looks like this. It's a constant matrix, these are all symmetric, plus and then a linear combination of symmetric matrix's here and I want to minimize the maximum eigenvalue of that matrix. Now, let me just point something out. That's a really complicated problem, right, because you take this matrix – so far, it's fine, every entry of this matrix is an affine function of X, but now to get the maximum eigenvalue value you form the characteristic polynomial. That's it. It's huge so you don't want to see it. You can't see it for a 15 X 15 matrix. Okay. You don't want to see it. You get some huge polynomial of degree N if N is bigger than five, there's no formula for the roots at that polynomial then dispute the fact you have no formula for the roots, you now have to select the largest of them. Okay. Now, that can switch. As you mess with a matrix the second largest eigenvalue value and the largest one can kind of go like this and when they switch this function is non-differentiable. Okay. So I just want to impress upon how complicated this function is.

It's very complicated, this thing, however, we can write it's non differentiable, it's highly non linear and so on. Now, on the other hand, we can actually optimize the maximum eigenvalue very easily as an SDP. It's totally straight forward. It's just this. You minimize T subject to A of X is less than or equal to TI. Now, to write this in this full form, I guess, these things I'm not gonna do, but you write it this way. And this is indeed affine in the variables X and T. That's this thing. So people just write it this way. It's fine. So that's eigenvalue. So eigenvalue minimization is quite straight forward. This is something that 20 years ago no one knew that this was straight forward, no one, 10 years ago a few people knew and so on. So this is new. This is not LP. We're not talking LP anymore. We're not talking stuff from 1953 anymore. This is recent history. Let's look at another one. Matrix norm. Now, matrix norm, as you know is a very complicated function. It's the square root of the largest eigenvalue of A transposed A. Now, when you look at this you immediately know you got a problem here. Land of max, well, we know that's a convex function. We know that. Right. Not only that, we know it's sort of SDP representable because we just showed how to minimize the length. But the problem here is this essential non linearly. This is quadratic and in an SDP, which after all has the constraint of a linear matrix inequality, you got the word linear in it. You're not allowed to have products. This has products here of A with A. So you have XI XJ terms. That's not linear. So the question is how do you turn a non linear problem into a linear matrix inequality and again, it's an SDP embedded so you take a larger SDP, it's larger but it's actually gonna be now affine or linear in the variables. So let's see how that's done. It's done this way. And you have to study this matrix quite carefully. It's in the queue. We're getting it ready for you. You have this block matrix here and this is TI and you want to know when this matrix is positive semi-definite.

Well, it's basically you have to have TI positive, and the other condition is that TI is bigger than A of X transposed times the inverse of that. I'll just put divided by T because the inverse of I is I times that. Okay. And very important here – that is a matrix inequality. Okay. There. T is positive; it's non-negative so this T can go over here safely like so. Now, you take a look at that and you realize we're quite cool because T is non-negative I can write this as TI is bigger than – no, I'll leave it this way. This basically says that T squared times the identity is bigger than this matrix. In a matrix sense that means that all the eigenvalues of this matrix are less than T squared. Okay. That means the maximum eigenvalue of this matrix is less than T squared, but the maximum eigenvalue of A of X transposed A of X is the maximum singular value of A squared and you take square roots and you're back to this thing. Okay. So that's the embedding. You'll see more of this. I just wanted you to sort of see it now. Also, it's interesting to see you don't have to know the details yet, later you will, that if you're actually curious about what CVX is doing, after all you can look at all the source if you like and find out what's happening, but if you look deep enough in it and then ignore all the silly testing and cases around there, the essence is every function implemented in CVX is an LMI representation. That's not quite true. In fact, let me say a little bit about that. I don't mind lying, but let me just – I don't do it that often and when it's for a good cause it doesn't bother me at all, but I'll just make this a little more accurate. It's not a total lie. What cone solvers really do and it comes back to your question to be honest, so the question is if you go to Google and you type cone solver or something like that, you'll find all the

basic ones, you'll find there's 20 or 30 by now, there's a commercial one actually that you can get in excel so it goes on and on. But they all kind of do the same thing. What they do is they solve a conic problem which is like [inaudible] note the period, C transposed X subject to – and there's many ways to write it, but I'll write it this way – I'm gonna make these lower case – something like that.

Okay. And you can put an inequality constraint here. That doesn't matter. So they solve problems like this. Now, here I have to interpret these – these are the Ks you can have. They're typically sort of R plus so that's linear inequalities, they can be second order cone or larents cones so they can be SOC of some size K or something like that and they can also be sort of linear matrix inequality of PSD cones of size K. So this is really what they look like. As to whether the linear inequalities and the second order of cone inequalities are then further dumped into matrix inequalities and solved, is actually a question of semantics and things like that. So we'll just leave that out. But this is in fact really what if someone says cone solver, this is what they mean. This is not that interesting to people who want to use these things. I mean, I don't know what this would be like. This would be like looking at a very low level LA pack code and looking at the full calling sequence. It's of great interest to people who write low level code. It's very important that it be done right. That smart people thought about it, implemented it. But not all of us need to see the 14 arguments that you call, you know, DGESL with or something like that. Most of us are perfectly okay with just kind of the back slash or something in between like in a python or something where you'd actually let somebody else deal with all the memory allocation and passing pointers and all that kind of stuff. So I think this sort of has that flavor of that cone solver. So now I've been honest about what an actual cone solver in practice is. Okay. All right. Let's move on. What we're gonna do now is we are gonna to generalize the objective to be a vector. So, so far, we left the objective as a scaler and we generalized the inequalities to be vectors and you get things like cone programming and all this kind of stuff. Now, we're into the objective and this is a different thing all together. This is a very different thing when you change the objective because now you actually have to redefine the semantics because first of all it doesn't even make any sense so the first thing you have to do is say what on earth does this even mean. What does it mean?

What is FO of X is a vector with two things? What does it mean to say minimize a vector? Okay. And you get all sorts of nonsense about this so it's actually important to get all the semantics right. So a convex vector optimization is just like this except FO of X is a vector and there's a cone associated with it and there's a cone associated with it. And let me just say what this might mean. Let's do the scaler case. Let me tell you the semantics of scaler objective optimization. It goes like this. If the semantics are extremely simple and it all comes down to the following. If you have three people come up with a proposed X – I'll tell you everything you need to know about the semantics of scaler optimization. I look at the first X and I check the constraints. If any constraint is violated in the slightest I dismiss that X and that person, right, it means it's completely unacceptable and they can't say things like it was almost feasible or something like that. They're dismissed. Okay. Now, the remaining two people both pass the feasibility test. They're Xs are both feasible. They need the constraints. They haven't been dismissed.

How do I judge between them? Well, I mean, it's extremely simple. I evaluate the objective function for those two points. Okay. If one is less than the other I say that's better and I dismiss the other one because here I have two potential points, each of which satisfies the hard constraints, this one is better on the objective. Now, if they're the same, then the semantics – actually, what is the semantics if they're the same?

Student:[Inaudible]

Instructor (Stephen Boyd):No, they're not both optimal. I didn't say they were optimal. The answer is as far as that problem is concerned, you actually don't even have a right to have a preference between one or the other. You don't. Okay. So if you look at that and you say, "Oh, no, I much prefer this solution," then your objective is not reflecting your interest and your utility function. If you like one more than the other, but the objectives are the same, you're modeling is not right. Everyone see what I'm saying here? So that's it. Now, the key to this – in the scaler case is easy because an objective comes out to a number. It comes out to, like, $17 a cost per unit, in which case, $16.38 is better than $17, period, if it's a cost. If it's a revenue it switches. Okay. So the point is that they're all comparable. Now, suppose they're vectors. Now they're vectors and as you know with vector inequalities they're not linear orders. Linear ordering is one in which any two elements can be compared. That's one of the nice things about linear. There are linear vector orderings. Did we assign that problem? I think maybe we didn't. Does anyone know a linear vector ordering? This is just for fun. What.

Student:[Inaudible]

Instructor (Stephen Boyd):Alphabetic or lexicographic. Someone else was gonna mention something. Same thing? Yeah. So lexicographic ordering on a vector says you look at the first entry, you get two vectors, you look at the first entry and if one of the entries is bigger than the other, done, it's higher in the ordering. If they're equal, you drop to the second and then you see who beats on the second and if they're equal, you go to the third. So that's lexicographic ordering and it's a total ordering or a linear ordering is the name. So if they're equal all the way through then the two vectors are the same and there wasn't really a question anyway, right? So that's lexicographic ordering and it's a total ordering or a linear ordering is the name for that. It means any two points can be compared. That's why dictionaries work and blah, blah. Okay. All right. Most orderings of interest are not total orderings, most vector orderings, of course, the ordinary ordering on R is a linear ordering, but most vector orderings are not. And in fact, that's exactly what makes it interesting. So for example, in matrixes for example when you're talking about confidence ellipsoids and you say which is better, this or that, this estimation method has this confidence ellipsoids and this one has that one, which is better?

Well, of course, it can happen that one of the ellipsoids fits in the other and in which case you can say I like that one better, but of course, you can have this thing where they're not comparable. Were neither ellipsoid contains the other and then you have to say something like algorithm actually is better at estimating [inaudible] or whatever than elevation. This one on the other hand is better at doing this than that. So that's the idea. This is just to get

the rough idea here. Okay. So we actually have to figure out what is the semantics here. Well, the way this is done you have to look at the set of achievable objective values. Now, that is a subset of a vector space of dimension bigger than one. We're assuming that's the case here. You look at the set of achievable values. That's the set of objective values you can possibly get with a feasible X. And now you say the problem is very simple. You say the point is optimal if that achieved objective value is minimal in O. You say it's pareto optimal if it's a minimal value. Now, you probably don't remember these from long ago, but the ideas are this. A minimum point of a set is one where every other point in the set is comparable to it and larger. So this is a minimum point. If this is the set of achievable objectives, this is a minimum point. This is the only case, essentially, when you can say unambiguously this point is optimal. It's the only time you can actually say minimize something and say X solves this because no one could possible argument with it here in this case. Minimum means you're better than every other point. Minimal means no point is better than you. Now of course if in total ordering these are basically the same. They're the same. But the difference here is you can have points – so if you look at this point here, all of these things are honestly beat by this point.

They're honestly beat by this point because they're comparable to this and they beat the object period. That's this point. But these things over here are interesting. These over here are interesting so the things in the second and the fourth quadrants, they're not comparable to this point. They don't beat it. Any point over here would beat that point unambiguously and the whole point about being pareto optimal or minimal in this set O is that there are no such points. That's the definition of being pareto optimal. So you would say, in fact, if someone said, "Yeah, what about that point and that point," you would actually say that they're incomparable, that neither one is better than the other and you would say, in fact, that in discussing the two designs there's a tradeoff between the two. You can trade off one objective for the other. Okay. So this is the picture. Some of these ideas are kind of obvious and you've probably seen them before in other areas.

Student: That curve – a significant portion of that curve also is pareto optimal?

Instructor (Stephen Boyd):Yeah. You see this little thing in here?

Student: Yeah.

Instructor (Stephen Boyd):That's not pareto optimal. Okay. And actually, that point, although you couldn't possibly see it certainly at video resolution you can't see it, or can you? I don't know. It's somewhere right around one line. This point is open and this point is actually filled in so these are pareto optimal down here and all of these are. Right? Another name for pareto optimal is not stupid. That's another technical term because for example if someone says, "Yeah, well, that's my choice," okay, the technical name for that is stupid and the reason is this – if you go down here like this, you know, everything like this – anything in there is a point that unambiguously beats this. So it would be stupid to choose this point. Any of these other points in here would actually be comparable to that point and unambiguously better. So that's the other way to think of it is just not stupid. But the way to do it is very simple. It basically says minimal means you

look up in here and you want everyone else to be up there. Those are the places that are unambiguously worse and pareto optimal says you look down and left. I mean, this is in this very simple case. You look down and left and these are all the places which are unambiguously better than you and you want to make sure that no one is unambiguously better than you. So that's it. And you've seen this idea many times – my guess is you have. Or even you've thought it and just didn't have a name for it or something. All right. Vector optimization comes up in essentially only two practical contexts. One is where the ordering is with respect to matrixes and that's where you're comparing and that actually comes up all the time. It comes up in experiment design; it comes up in estimators and things like that where basically the objective is something like a covariance matrix like an arrow covariance matrix. Everybody wants a small arrow covariance matrix. It defines an ellipsoid. Everyone wants a small confidence ellipsoid. That's clear. The problem is those are not totally ordered, right? And we'll talk about what happens in that case. So that's the first case when these are matrixes. That's sort of the subtle case. The other case is actually much simpler although it comes up maybe more frequently. It's just multi-criterion optimization and here the cone is nothing but the [inaudible]. So here basically it's called multi criterion optimization. Your F0 has got a bunch of components and people call this a multi-objective problem, that's another name for it and you will hear a lot of nonsense put up around all of this.

You hear people say, and of course, it's true, you hear people say things like, "Oh, all optimization problems are multi-criterion," well, like, duh, so what? I mean, that's completely obvious and we'll see that it relates immediately if you know what you're doing to ordinary optimization problems. So we'll get to that in a minute. In a multi-criterion problem, you have multiple objectives and if you want to think about something specific here you might think of an engineering design, let's say a circuit and one of these would be something like power, one would be delay, one would be area, I mean, just for example. Okay. Or this could be some control system or something like that and one of them could be, again, something like power, one could be how close you're tracking and the other could be something like the roughness of the ride. The point is there are multiple objectives all of which you want small. Once you think about it you realize everything has this form. In an estimation you'd say, "Well, I want something that matches my observed measurements well," I mean, it would be foolish not to have something like that, but at the same time, I want an image that's smooth, or I want one that is simple in some method. In some sense, those are all gonna be multi-criterion problems. Okay. And I'll say a little bit about how it all fits together in a minute. Okay. Roughly speaking, you want all of these small. Now, the one thing you can say is this, if someone comes up with a point and you beat someone else on all objectives – or I should be more precise, if you meet or beat them on all objectives and beat them on one, you're better. Everybody got that? This is like a core in economics you go through all of this. You'd see all these things. That's what it means to be better. So pareto optimal means no one beats you. That's pareto optimal. It means basically that if you have two pareto optimal things, you have to have traded something off.

They have to beat you on something; you have to beat them on something else. Okay. Now, pareto optimal, well it says exactly that. It says that if one thing is less than or equal

to the other the same objective that's in a vector ordering then they're equal. And if you have different pareto optimal values there's a tradeoff between the objectives and the classic thing from lee squares or anything else would be something like that so you'd write it this way, you'd say minimize with respect to R + squared. I mean, this is a bit of a pedantic way to write it, but it is actually exactly what you mean. You say please minimize with respect to the second order [inaudible] the following: two objects, both by the way convex here. Norm X minus B squared, that's like a matching criterion. That's how well do you map something. But at the same time, please do it with a small x. Okay. How do you judge such a problem? Well, what you do is this. X by the way could be a vector in 10,000 dimensional space, but there's only two objectives, so in principal, what you do is you check every vector in our 10,000 so you plug in every possible X and you get a score, which is a pair of numbers, non-negative numbers and you put a dot at every single point. This is conceptual. You now do this for all 10,000 points and you get this shaded plot that looks like that. In fact, it's some kind of a hyperbole or something, it doesn't matter what it is, right? Now, also I should add that O has some interesting structure out here, but it's not interesting to us. What's interesting for us is down and to the left. That's the only thing interesting to us. And therefore, whatever O looks like up here is actually of no interest unless it kind of does some weird thing and wiggles around and comes out down here, but it doesn't, okay? So we're only interested in what's down and left. So in actuality the only thing that's interesting here is what's shaded here is the pareto optimal curve. Now, you saw this if you took 263. This is called the regulization and so on, so you've seen all this.

I'd say the technical term for this is quite stupid. Let me justify that. It's quite stupid because, first of all, there's better points. That's number one. Number two, it's not even achievable, so that's why it seems to me it's even worse than stupid, but anyway, this should be clear. Here's a generic example. It looks like this. I have two objectives and let me explain the problem here. I have a vector that represents an investment portfolio so XI, for example, might be the fraction – people set this is up differently, but a classical one is this – XI represents the fraction invested in asset I. So you have N assets, negative – if you want to have negative, I mean, here we've ruled it out, but the point is if you want to have negative it means you have a sure position. If that asset is cash, it means you're margining. You're borrowing money to finance the purpose of other assets or something like that. But here to make it simple we just made X a positive and you'd say that there are no short positions and if one of those assets is cash you'd say no borrowing. No margining. Okay? So that's the idea. That's the fraction. And of course as a fraction, it's got to add up to one. Okay. Now, what happens here is you invest in these assets as one period and what happens is there's a return or the prices change, so you can call P – it's a vector of relative asset price changes. So if you form P transpose X that gives you the total return on your portfolio. Okay. Now, obviously, if the returns were known – in fact, let's discuss that because it's a very short discussion. If I told you the returns, what's the best possible portfolio?

Student:[Inaudible]

Instructor (Stephen Boyd):What is it?

Student: You put it all into the best one.

Instructor (Stephen Boyd):Thank you. You put it all into the best one. So you find and someone tells you, reveals to you what the returns are, it's extremely simply, you simply put EK, where K is an index corresponding to whichever one had the highest return. Okay. That was the discussion of deterministic of finance. It was a short one. So the interesting part is this. It's a random variable. So the return is actually a random variable so it's a random variable here and you can get much more sophisticated, but the basic one goes like this. So P is a random variable and therefore P transposed X, that's the total investment return, is a random variable. Okay.

And now obviously you want it large. What does it mean to say a random variable is large? By the way, there's tons of ways to say a random variable is large. Okay. Lots. You could say it's 90th quintile is large, it's 10th quintile is large, you could say all sorts of things. There's many, many ways of doing this. The simplest thing that makes any sense at all is this, you evaluate the expected return and the variance, or you could take the square to the variance and get standard deviation. That's sort of the minimum that makes any sense at all would be this pair. And now you have a bi-objective problem because everyone wants high-expected return and everyone will take a lower risk over a higher one. Let's just say that people are risk adverse. So what that means now is you plot these things by two things; you'd look at the risk as measured by standard deviation of return – of course you can get much more sophisticated measures of risk like you can have downside risk, value at risk, you know, it goes on and on. But the simplest measure is just variance. Then you have all sorts of long discussions about this like somebody would make the stunning – are you ready for an amazing observation – how about this, somebody points out, "Yeah, we judge portfolios by risk, but it turns out – well, imagine what would happen if your return was much higher than you thought, would you be unhappy or happy?" And you'd say, "Well, let me think about that for a minute. I guess I'd be happy and they go see variance is the wrong measure of risk, you care about the downside variance." So these are all totally obvious things, but let's just – and by the way, if you make any assumption about the distribution, like, if it's galsium, then all these go away. So let's just assume variance is a measure of risk or standard deviation and what you'd do is you'd solve this problem – sorry, I haven't said yet how to solve a multi-criterion problem, but this would actually be the pareto optimal boundary. You'll do plenty of these, just for fun.

Now let me say something about what does it mean to solve a multi-criterion problem? Actually, that's an interesting question. We all know what it means to solve a scaler problem; what does it mean to solve a multi-criterion problem? Well, there's many definitions. One is this, if that multi-criterion problem has an actual optimal point then for sure solving means you better find it. So if there was actually – and it could happen. Here in a finance problem you could have something where basically you have one asset that beats all of the others in both risk and return, for example, in which case this would be a stupid problem. The pareto optimal curve would be a single point. Everything else would be down and to the right meaning that you can choose this portfolio or if you wish, there are many portfolios to choose from, but all of the have both lower return, higher risk and

one of those inequalities is strict. So that's what it would mean in this case. It could happen. It obviously doesn't happen in practice, but that could happen. Okay. Now the other meaning is something like this. You could define the semantics of solving a multi-criterion problem as producing a pareto optimal point. That's another meaning for what it means to solve it or an interesting one would be to produce one or more – or actually you want a method for exploring the pareto optimal points. That's actually probably what you want when you say how do you solve it. So most methods for solving multi-objective problems will have some parameters or a joy stick in them that would allow you to move around on the pareto optimal surface, which would be called exploring the pareto optimal surface and that kind of thing. So classic methods, probably hundreds of years old, scalarization. How does that work? In the general case, actually it's quite elegant. It works like this. You choose a positive vector, positive in the dual cone, okay, in the dual cone. Now, you might remember this.

When we looked at it, it was completely context free and had no actual practical meaning so you probably ignored it, but one way to say that F0 of X is K convex – one condition for that is the following; it's that lambda transposed F0 of X should be convex. That's a scaler function. It's a convex function for all lambda incased [inaudible] in the dual cone. So that's one way to say it. In particular, this is a scaler convex optimization problem. Okay. By the way, this is already said something. In the case of multiple criterion we'll see it's something very simple. It's a positive way to sum up the objectives. I mean, everybody knows that. You want to minimize a bunch of things, you put positive weighted sums and you wiggle the weights around to change the tradeoff. That's obvious. It's much less obvious when we get to matrixes. You want to minimize, for example, estimation error covariance, it says take a positive definite matrix, you know, lambda and minimize trace lambda sigma. That's what it says. This is not obvious just to let you know. Okay. So here's what happens. This is a general problem, non-convex and here's what is really happening. So here in this problem if you take a lambda like this and you minimize this, it basically says you go as far as you can in this direction and you might find that point right there. That's pareto optimal. Okay. Now, by wiggling lambda you're changing the slope and it says for example you can get this point. Now, what's very interesting here is that this point right here is pareto optimal, but you cannot get it by any choice of lambda. It's shadowed. There's no way of getting it. If you want to wiggle around on the pareto optimal design, you change the weights and it's kind of obvious how you change the weights. Actually, the choice of weights relates to our very next topic which is duality, but for now, you can think of the weights as simply a joy stick that moves you along the pareto optimal surface so that's the right way to do it.

So if you go back to the risk return tradeoff now I can tell you how those plots were obtained. They were obtained as follows; we maximized this minus that and I varied lambda from very small to very large and I solved this QP for each value and that's how I got the curve. Every one of those is risk return pareto optimal and this is in fact gets you everyone in the convex case. By the way, this has a name; this is called the risk adjusted return. That's the average return and now you've penalized the return by some adjustment for risk and then you would say something about gamma is your risk aversion parameter because it tells you how much mean return are you willing to give up for

variance in your total return so this would be a risk aversion parameter. For us, it's a weight, it's in the dual cone of R2, it's a weight that simply weights variance versus return. That's all it is. So that's that. Okay. So this finishes our whirlwind tour through optimization. You should have read chapter four by now and you should read of course the CVX users guide for that and they should be coherent in fact. Well, I hope they're coherent. If they're not coherent, let us know. We're by no means done, but the rest of the class, every homework, no exception, you will see problems where you will formulate them as convex problems, you'll do numerical solutions, you'll work out more theory as we do it. So in some sense, we've kind of finished our first coverage of problems – we do problems from now on period in applications and things like that. Now, we're going to enter a new topic, which is actually sort of our last sort of theoretically oriented topic. There will be some others later, but this is a very important one. It's duality. It relates exactly to these weights. And I'll start on it. This is in chapter five. You should start reading chapter five. Okay. Duality is a very interesting topic. It's interesting theoretically, it turns out it has extremely interesting computational implications. It also turns out it's extremely important in all practical cases so whenever you use optimization, ever, you have to be aware of duality. And by the way, it's not just a theoretical thing; however, if you randomly search in the literature on duality you'll imagine it's just something that people who do theory worry about. It's immensely practical and we'll see that into it. There's something I forgot. I pushed it on a stack and didn't come back to it. I want to come back and say something about multi-criterion optimization versus scaler. And I just wanted to put them all in the common parent so the common parent of scaler and multi-criterion optimization is this. If you want to put them on a common footing, you'd introduce a concept called the hardness of a constraint. Okay. So that would be the word used. A constraint that is infinitely hard or let's say an objective that's infinitely hard. The problem is constraint already means hard and objective already means soft so let's call them objectives, but let's call them functions.

There we go. Functions. F0, F1 and so on, we're gonna call them functions. Infinitely hard constraint – so you can do the concept of hardness by plotting your irritation versus the value of FI of X. So here's 0, that means you're not irritated at all and the semantics of the objective, your plot looks like that. That's your irritation; I guess this means you're pleased. Okay. And this says basically the following; it says that you can make F0 – F0 can be large, if it has to be large, it has to be large. It irritates you, but if it has to be large, it has to be large, so this would be a soft irritation function, right? So that's the picture there. You can also be pleased and if you have a choice between two points, one of which was sort of here and one was here, you'd prefer that because you're lower and it means you're less irritated. Everyone see this? Suppose you wanted to change this irritation function; what irritation functions work right away with convex optimization, just instantly? Let me draw an irritation function and you see if you like it. How about this? I'm down there. That's my irritation. And then someone says, "Why," and you go, "Not a problem. If this is the area, yes, I always like things to be smaller, always, but you know what, when things start getting bigger than some threshold, it starts irritating me super linearly," that's what this is. Can I express this as convex optimization problem? I can because I would simply take this function of F0 – this function is convex and it is increasing and therefore I can do that. Okay. Then the objective value would match

yours. For a constraint, the canonical irritation function looks like this; it's along the axis so I kind of have to draw it that way. Okay. That's zero. That's your canonical irritation function. It basically says if – let's say in a classical problem, if F3 of X is less than or equal to zero, it doesn't irritate you at all. By the way, the same is also true in the semantics of the basic problem that a smaller value of F3 means nothing to you. Absolutely nothing. By the way, you might find someone in practice say something like "Oh, no, I prefer a smaller – I prefer," but the actual semantics of the base optimization problem says that F3 of X equals 10 to the minus nine, right, and equal minus 100 are equal for you. This is by no means any better than that. They are both feasible. End of story. That is the same as that. This and this are also equally and infinitely irritating. They are totally unacceptable, and here you can't say come on, I just barely, you know – now, as a practical matter, you might argue and if you look carefully, what numerical solvers will return, I mean, obviously, they'll return things that, for round off reasons, might violate things slightly.

Anyone who asserts that there's any difference between these means that you haven't formulated the problem. The common parent is the following is that you look – this is sort of a very general thing where every constraint has a certain softness and hardness and if you're willing to label all the merit functions except for one as absolutely hard and one as soft, then you have a classical problem. That never really happens. If you label them all as soft, you'd have a multi-criterion one. What really happens is that some constraints are non-negotiable and others maybe are and they might be negotiable for different ways. And you might even have weird things like you could say, "Well, you see this power constraint or whatever," but actually if the power goes up to two watts everything is fine because I can use this – I mean, actually, I like less power, but I can go up to two watts with this package. If you go above two watts, boy that's not good. Let's get back to duality. Maybe I'll just say a little bit about it. It's Lagrange duality, which we'll see it in a bit. I'll just say a little bit about what it's used for and all that kind of stuff. Hopefully, this is will de-mystify the often mystical Lagrange multipliers, which is generally taught as a behavior in an advanced calculus class. So this will completely de-mystify it. That'd be the first thing. Actually, it's got a lot of interesting things. In fact, what it's gonna do is it's gonna end up producing – from any optimization problem it will produce a convex problem, which is called the Lagrange Dual. Okay. If you start with a non-convex problem, you'll get a Lagrange dual, which is convex. We can solve convex problems, I mean, roughly speaking. We can solve them. Okay. That Lagrange dual problem is actually gonna give a bound on the optimal value of that original problem. Okay. If that original problem is convex, it's gonna be a sharp bound.

There's some conditions, but it's gonna be a sharp bound and it's gonna be very interesting because it's gonna give you alternative methods for solving that problem. It's gonna give you all sorts of interesting things. It's also gonna be extremely interesting when the original problem is non-convex. Well, for the Lagrange dual, it's gonna be a convex problem, we can solve it. And you're gonna get bounds on that original problem. We'll see a few cases where that actually happens. So that's a case where a problem that looked hard by forming a Lagrange dual and then doing some extra math to show the bound is sharp, actually is solved by convex optimization. By the way, there's gonna be

lots of those. Okay. Any time someone says a problem is easy and it's not posed as a convex problem, it's likely that in fact it's because the Lagrange dual is sharp and that's true for many [inaudible] problems and all sorts of things. The other thing that could happen is you could get a bound on a problem not known to be hard, but suspected to be hard and there's amazing things that people have recently proved. There will be a semi-definite programming Lagrange dual of a hard [inaudible] problem and they prove things like specific ones, like, Maxcap for example, when you solve this Lagrange Dual the bound is never more than 13 percent off. Crazy things like this. That gets the complexity theorists all excited and indeed that result got them very, very excited. As you can imagine, right, but also as a practical method for generating sub optimal solutions of that problem, these things can also be fantastically useful. So that's our next topic. It's gonna be essentially the last sort of core theoretical topic in the class and I guess we'll start in on that next time.

[End of Audio]

Duration: 77 minutes