ConvexOptimizationII-Lecture06

Instructor (Stephen Boyd):All right. We'll start. The first thing I should say is that we're about to assign Homework 4. How was Homework 2? That was due Tuesday, right? And you know we have a Homework 3 assigned. We're gonna assign Homework 4 so that we're fully pipelined. We have at least two active homeworks, typically, at once. What's wrong with that? That's how you learn. You're learning, right? Sure you are. And so the other thing coming up soon is this business of the project. And on the website, we have some date we made up before the class started. And I believe it's even tomorrow. It's tomorrow. So by tomorrow, you should produce some kind of two- or three-age proposal to us. And we'll iterate it. If we don't like it, we won't read past the first paragraph. We'll just give it back to you and you'll keep working on it until we like it. So the way we'll like it is it's got to be simple. There's got to be an opening paragraph. In fact, we were even thinking of defining a formal XML scheme or whatever for it. But it's got to be a paragraph that has nothing but English and says kinda what the context of the problem is. So for example, I'm doing air traffic control or I'm blending covariant matrices from disparate sources or something like that. And says a little bit about what the challenges are and what you'll consider. Then when you get into what the problem is, you describe kind of what it is and what you're gonna do. If it goes on for more than a page, if the setup is more than a page, it's gonna be too complicated for the project. Your real interest might be complicated, but then that's not gonna be a good project here. So what we want is we'll read these and we'll get feedback to you pretty quickly, we hope, on these things. So you should also be just talking to us, just to kinda refine the ideas and stuff like that. And things we'll look for is, it goes without saying, this will be done in LaTeX. Nothing else is even close to acceptable. I won't even name alternatives.

The other is that the style should look something like the style of the notes that we post online. That should be the style. There are lots of styles. That's one. It's a perfectly good style. So I don't see any reason why you shouldn't just attempt to match that style. That's statistically for random style checker. No statistical test should reveal that, for example, I didn't write it. That would be a perfectly good model. There are other styles that are very good and different, but that's a good style and you could go with that. So if you look at your thing and if you find that your notation for an optimization problem differs from mine, change yours. Because, although there are other consistent notations, like there's the Soviet one, and there's other ones, I don't need to see them. It's just easier. And if used consistently, that's perfectly okay. But just to make it simple, just use our notation. So when you describe an optimization problem, it should look like the way we do in notes, finals, lectures, all that stuff. So I guess they'll be due tomorrow at 5:00 or something like that. If you're still sort of in active discussions with us and haven't quite closed in on that, that's okay, but you should be – then come to us and let us know you're gonna do it over the weekend or something like that. And we'll consider this just the first cut. Are there any questions about that, about how that's gonna work? Anybody still looking for a project, because we can assign. Every lecture I say, "This would make an outstanding project." No one's ever gotten back to me – ever. We can make up project for you, too. Yeah?

Student: Can we changes to the initial proposal?

Instructor (Stephen Boyd):Yeah. Don't even think about handing us something that's 15 pages. We're not even – actually, ideal would be a half page. I mean, you have to have just the right project. By the way, you might be able to pull it off, half page. Okay. That's fine. That would be great. Shorter the better. And if there's any previous work or something like that, we haven't asked for it, but at some point, you're gonna have to do that. So I mean, I presume at this point, if you have an idea of a project, you've kind of done some Googling and all that. We assume that. That goes without saying. And you can add some references or something like that, if you want. Okay. Any other questions about this? Otherwise, we'll just continue. Oh – no, we'll wait. So last time we looked at localization methods. So the basic idea of a localization method – you can think of these as just generalizations of bisection. They're generalization to Rn. So you have bisection in multiple dimensions, and it's not a simple – there's no simple extension to Rn. Because in R, first of all, convex sets are kinda boring. They're basically just intervals. And there's an obvious center of an interval. In Rn, things are much more complicated. You have many different centers. I don't know if you have that point. Where is it? Here we go. Thanks. Okay. So the basic idea is you have some polyhedron, where you know the solution – if it exists. And you query a cutting-plane oracle at the point x, and what comes back – in this case, this is a neutral cut here. In other words, what it's basically telling you is that you need not even consider anything over here. So the solution is not here, if it exists. And so this is your posterior or updated ignorance set, or whatever you want to call it. It's the set of points or localization set. It's the set of points in which, if there is a solution, you know it must lie. I should mention one thing, there's more material on this, of course, in the notes, which you should be reading. And we're expecting everybody to be reading. Okay. So bisection on R is the most obvious and well-known example. Let's look at some other ones. We'll look at some of these in more detail today. The first is the CG method. I just got up to that at the end of last lecture. Here we have x(k+1) is the center of gravity. I'll review that quickly and say a few things about it that I didn't get to say last time. Other would be maximum volume ellipsoid, Chebyshev center, analytic center. These would be some of the other ones. And we'll talk about each of those in a little bit of detail. So the CG algorithm is the CG of the polyhedron of your query point. And there's an amazing fact.

In fact, we may even, if we can find a simple enough group, we'll stick it on Homework 4, which we're working on. And the amazing fact says this, that if you take any convex set in Rn, and take the center of gravity of that set, and then pass any plane through that point, the question is, how unevenly does it divide the volume? So anybody guess the answer's 50/50? For example, in R, it's 50/50. But it turns out immediately you draw a couple of pictures in R2 and you'll see that, in fact, there's no point – you can make a convex body for which there's no point for which all lines going through it divides the area 50/50. But it turns out you can't skew it too much. And it turns out than the actual ratio you can go to is 0.63/0.37. So it's roughly not even two to one. Because it's this divided by 0.37. Not even 2 to 1. It's more like 1.5 to 1. And that means, basically, if you find the CG of a set, and you put the worst hyperplane through that point, it will chop off at least 37 percent of the volume. So that's what it is. Okay. So that says in the CG algorithm, that's very nice, because it says the volume of the localized set goes down by the factor of 0.63 at least. It can go down more. If you have a deep cut, it goes down by more. And of course, the 0.63 is the factor for the absolute worst possible hyperplane at each step. And in fact, it obviously is typically much less, or something like that. So that's that. We'll look at the convergence and look through various things. And we came, at the last minute, to basically the problem with the algorithm. The problem with the algorithm is computing the CG is extremely difficult. So it's a conceptual problem is the way it's described. Now you can modify the CG method to work with approximately CG computations. I'm gonna mention some of those, because there's some very cool new work, not from Moscow, from MIT, and it's very, very cool stuff. I'll say some of that. Obviously, you don't have to compute the CG exactly. So first, there are methods even from the '60s, where a cheaply computable approximation of the CG is given, and then you have a chopping, a slicing inequity theorem, which is not as strong as the 0.63, but is enough to sorta give you convergence and so on. So those you already had in the '60s. So those are approximately CG methods. But let me just mention, actually, a really cool method developed four or five years ago, something like that, at MIT. Very, very cool. Goes like this. You want to calculate the CG of a convex set. And you can use a randomized method. And the randomized method is called hit and run. Has anyone heard of this? It's just very cool. It's an advanced class, so I can go off on a weird tangent and tell you about it. So the way it works is this, you have a point, and you generate a random direction. So you generate a random direction and you make a line. And you need the following: you need the ability to find out where that line pierces your set. So that's what you need. So here's your set. I'll make it a polyhedron, but it does not have to be a polyhedra. Just any old convex set. It doesn't really matter. Here's your set. And you take a point here, and you generate a random direction. Actually, how do you generate a random direction? Uniform on the unit sphere. How do you do that?

Student:[Inaudible].

Instructor (Stephen Boyd):Thank you for bringing that up. I should say, calculating the CG in 2D and 3D is no problem at all. 2D is trivial. All you do is you calculate, you find a point in the set, you calculate a triangulation of the polyhedron, done. In 3D, same thing. However, it's gonna scale sorta exponentially in the dimension. So if you're interested in 2D and 3D, CG is a non-problem. Okay. So how do you generate a random direction, uniform on the unit sphere? Just a random variable uniform on the unit sphere?

Student: Choose a random vector.

Instructor (Stephen Boyd):Yeah, fine, but how do you choose a random vector? What distribution?

Student:[Inaudible].

Instructor (Stephen Boyd):Why, because you can't think of any other? You happen to be right. You choose a random Gaussian vector. So it's n0i vector. That's circularly symmetric. And so if you then normalize that, you get a uniform distribution on the unit sphere. So you choose a random direction here, and so here’s the line you get. And the only thing you need to do is to calculate these two intersections, which is very easy to do. Now you do the following: you generate a random variable uniform on this interval, and that is your new point. So that’s gonna be right here. And you repast. So you go here, you calculate a random direction – the direction turns out to be this – you do a random point on there when you get there, and you go on. So I suppose these are the hitting points and this is the running, or who knows. I don’t know how that name came up, but that’s what it is. Everybody got this? So that’s a Markov chain. It’s a Markov chain propagating in the polyhedron. And it actually converges to a steady state distribution that’s uniform on the polyhedron. So that’s a method for generating a random variable that’s uniform. By the way, if you have a method that calculates a random variable that’s uniform on the set, how do you find the CG?

Student:[Inaudible].

Instructor (Stephen Boyd):Thank you. You just average. Let me ask you another question. Anybody got other ideas for how to generate a random point in a polyhedron uniform distribution?

Student: Run the distribution everywhere and reject the ones at the outside of the set.

Instructor (Stephen Boyd):It couldn’t be everywhere, because it has to be uniform. Okay. So what you do is you find a bounding box. Everybody knows how to find a bounding box. How do you find a bounding box for a polyhedron? This is the problem with scheduling class at 9:30. It’s before full outer cortical activity has warmed up. How do you find the bounding box for a polyhedron?

Student: Minimize x1 [inaudible]. Instructor.

So you minimize x1 and you maximize x1. You minimize x2 and you maximize x2. And these are like LPs or something. Of course, maybe we’re trying to solve an LP. I don’t know what we’re trying to do. But still, that’s how you do it, by solving two NLPs. So you make a big bounding box like this, and now you generate a uniform distribution on that. And you do that by taking a uniform distribution of each edge and sticking them together. And then you reject the ones that are outside this thing. And the ones inside, actually, then, are drawn from a uniform distribution on the polyhedron. Any comments on how you imagine that might scale with the dimension? I’m not saying you have the bounding box.

Student: Exponentially.

Instructor (Stephen Boyd):Which way, though? Exponentially what? What would happen exponentially? You’re right, something happens exponentially. What is it?

Student: To test if the point is in the polyhedron?

Instructor (Stephen Boyd):No, no. To test if a point is in the polyhedron, you form ax. You give me x, I calculate ax. I check if it's less than or equal to b. That's fast. What happens.

Student: Volumes.

Instructor (Stephen Boyd):There you go. Ratio volumes in the worst case goes down exponentially. And that means you'll generate exponentially many random points before you even get one in this set. So although that does, indeed, produce a uniform distribution on the set, it's not gonna scale gracefully to the high dimensions. Okay. So anyway, this is the hit and run method. There's lot of – you can certainly do a cutting-plane method with something like this. And a more sophisticated version. Okay, so that's the CG method. Yeah?

Student:[Inaudible] in the vertices, where it intersects?

Instructor (Stephen Boyd):Oh, here? Oh, that's actually quite straightforward. If this is a polyhedron, let me show you how that calculation goes. So the calculation goes like this: here's your polyhedron. It's defined by that inequality. So ax less than b. You're given a base point x0 and a direction, v. And you want to find the maximum and minimum values for which this holds. That holds for t for an interval, right. So I'm assuming this is bounded polyhedron. I'm assuming x0 is in the polyhedron, although none of that matters. So that says that a(x0), that's if t equals zero, is less than b. And now you want to find how much can you increase t until this becomes false. And how much can you decrease t below zero until it becomes false. That's the question. So you just write it this way. It's actually kinda dumb. You write a(x0) plus t times av is less than or equal to b. That's a vector, that's a vector, now it's very easy to work out the maximum. Because this is basically – let's call this a plus t – no, no, that wasn't a good idea. How about a plus tc is less than b. Now it's gonna work. Now it's extremely easy. What you have to do is this, if I increase t, this thing – by the way, if a ci is negative, or zero, increasing t doesn't hurt – we're not going to get into trouble there. So if I want to know how high can I increase t, I only focus on the positive entries of c. So I first look at the positive entries of c, and I divide them by a, or something like that, and then I take them in, or something, and that's my – well, I do something. Anyway, it's a 300-level class. I can just say it's easy to do. This made sense, though? So in fact, when you actually do this, it's two little lines of something. Okay. Maximum volume ellipsoid method, this method is not a conceptual method; it's a real one. Actually works really well. And here, x(k+1) is the center of the maximum volume ellipsoid in Pk, which, I don't know if you remember or not, but that can actually be computed by solving a convex problem. By the way, how about the minimum volume ellipsoid method? In that case, you calculate the minimum volume ellipsoid that covers the polyhedron. Let's have a short discussion talking about the minimum volume ellipsoid method. I don't know if you remember this, but you can kinda guess. Any comments on the minimum volume ellipsoid method? By the way, it would work really well. You would get a guaranteed volume reduction and everything.

Student:

[Inaudible].

Instructor (Stephen Boyd):Yeah. In fact, finding the minimum volume ellipsoid that covers a polyhedron defined by linear inequalities is actually hard. The method is like the CG method. I gave you enough hints that you could figure it out. Did you actually remember that?

Student:[Inaudible].

Instructor (Stephen Boyd):Oh, and then your remembered. Good. Okay, that's a short discussion of the minimum volume ellipsoid method. But the maximum volume ellipsoid method is actually a quite – it's not a conceptual method. It actually works very, very well. And in this case, the volume reduction you get is actually very interesting. And the volume reduction is this: the guaranteed volume reduction is not a fixed number, as it is in the CG method. In the CG method, you will reduce by 37 percent, end of story. Any dimension. In this case, your volume reduction guarantee degrades with dimension. By the way, it degrades gracefully enough that this method can actually be used to construct a polynomial time convex optimization method. Because a good enough approximation of the maximum volume ellipsoid can be calculated in polynomial time and so on and so forth. And this is also called – it's got four initials, TK – Russian names go together. I won't attempt it. Actually, I think it's described in the notes. Now, once you know to get this volume reduction, you can work out a bound on the number of steps required, and in this case, it scales not like n, as it was before, but it's actually n2. So this one degrades. And this is actually quite a good practical method. Okay. Other ones? You can pretty much make up your own method here. Chebyshev center method, I thought I'd mention it. Now, the problem with the Chebyshev center, that's the largest Euclidean ball in Pk, and that's solved by an LP, by the way. That's a linear program to calculate the maximum volume ball that fits inside an ellipsoid is just an LP. Now this one, however, is not affine in variant. In other words, if I take my original problem, and I do a linear change – if I do a scale, actually, if I scale the variables, maximum volume ellipsoid is CG, you get a commutative diagram. In other words, if you take a polyhedron and you subject it to an affine mapping, you multiply it by t and add some s, or something – t's a non-singular square matrix – then the CG will also change coordinates exactly the same way. Because CG commutes with an affine change of coordinates. So that means that the CG method is actually affine coordinate change in variant. Okay. The same is true of maximum volume ellipsoid. If I give you a set and I ask you to find the maximum volume ellipsoid inside it, if I then change coordinates by an affine change of coordinates and ask you to do it again, it commutes. So the maximum volume ellipsoid that fits in the transform thing is the transform of the maximum volume ellipsoid that fit in it. So that was my verbal way of drawing a commutative diagram. Now, this has a number of implications. First of all, for our complexity theorist friends, this is, of course, extremely important.

But it actually has lots of practical implications, and they're quite serious. Basically, you have a method that's affine in variant, in practice what it means is scaling of variables is a second-order issue. Now in exact arithmetic, of course, it's a non-issue. But it means it's a second-order issue. It's a second-order issue because scaling can hurt you, but only through numerical round off and things like that. Therefore, being off by a factor of 1,000 in the scaling coordinates and no one gets hurt. Whereas, a method like Chebyshev center, here, I guarantee you, if you have your Chebyshev method, the center method works really well, and I scale, for example, the first and the eighth coordinate by a factor of 1,000 to 1, your method is not going to work, basically at all. So that's the basic point there. Now having said that, I should say this, if you're solving a specific problem where you know what x1 is, x1 is measured in meters per second, and it varies between plus/minus five. And x7 is a pitch rate in radiants per second, or something like that, or radiants per second squared. And you know how it's scaled. So in actual specific practical problems, the whole business of affine and variants may not be that relevant, because in a specific problem, you generally know what the numbers are, what the ranges are and things like that. Okay. Now we're gonna look at this one in considerable detail, actually, in the next lecture, because it's one that this one seems to have a very nice property. It actually works really well in practice. And it also has some nice theoretical properties. So in this case, you take x(k+1) to be the analytic center of Pk. I should mention that this English here, the analytic center of Pk, that's slang. That's informal and it's actually not correct. Because the analytic center is not the analytic center of a geometrical set. You have the analytic center of a set of inequalities. And obviously, you can have multiple sets of inequalities that describe the same geometric set. And in fact, take your favorite polyhedron, and add all sorts of redundant inequalities. The set hasn't changed; the analytic center has. And in fact, the following is true: take a polyhedron, and take any point in the interior of your polyhedron. So pick any point at all. It can't be on the boundary, but take any point in the interior of the polyhedron. By adding redundant inequalities, you can make any point you like the analytic center. Now, you may have to add a large number of inequalities and things like that, that's true. But the point is, you should remember that this is informal and it is slang. So meaning it's just not right. Okay. So the analytic center is this. It's the argmin, of course, of the sum of the logs of – you maximize the sum of the logs of the slacks. These are the slacks in your inequalities. By the way, this is the same as maximizing the product of the inequalities. So it's exactly the same thing. You maximize the product of the inequalities. By the way, if you – there's many other things you could do. You could do things like maximize the minimum slack.

And if the ai's were all normalized to have the same length, for example one, that would correspond exactly to the Chebyshev center. Because bi minus ai transpose x is norm – whatever it is. It's the distance to that hyperplane divided by the 2 norm of ai. So this works quite well. And we're gonna go into that in great detail later today. Okay. For these cutting-plane methods, there are lots of extensions. I'll mention a bunch of them, and when we look at center cutting-plane method, we'll look at how these things work. So once you get the idea that the only interface is a cutting-place oracle, then you can imagine all sorts of cool things. So you could do all sorts of things. You could do things like this. Your oracle, when you call your oracle with x, instead of just giving a single – I mean, there's tons of stuff you could do. Instead of just returning a single cutting plane, you could easily imagine something that returns multiple cutting planes. And I'll give you an example. Suppose you're solving an inequality constraint problem. The simple method would be this: you take your constraint functions and you evaluate each one. You have to do that. So you evaluate f1, f2, f3, and the first time you find an f that's positive, you call that f, fk.getsubgradient. And you get a subgradient back. That's one method for generating a cutting place. But in fact, what you could do is if you sweep all the constraints, you could then get the most violated constraint. But another option is to give all the constraints, just give them all at once. And you just update your polyhedron and everything's fine. By the way, you can actually return not just – if you return multiple inequalities, you could actually return shallow cuts. Shallow cuts are fine, as long as one of the cuts you return is actually deep, or neutral. So it's still useful information. So let me just draw a picture to show what that would look like. So that would look something like this. And you might get some other kind of very colorful name for it. Here's your localization set. Here's a point – actually, it doesn't matter however you calculated it, CG, hit and run, analytic center – it doesn't matter. What would happen is you call it here, and you could do something like this. It could give you a deep cut, basically says, don't bother looking over here. But there'd be no reason that it couldn't also give you a shallow cut like that, that says, don't bother looking here. No problem. So you just incorporate both. So that's how that works. I mean, that's kinda obvious. Actually, what's interesting about it is that a lot of these things don't really make it work much better. Okay. Instead of appending one, you append a set of new inequalities. You can also do non-linear cuts. So you'll also find this talked about all over the place. And let's see, these would work – you could have quadratic cuts, where you return – that basically says, instead of returning a half space, where the solution, if it exists, must lie, you'd return an ellipsoid, basically. And there'd be plenty of cases where something like this would work, if your inequalities were quadratic or something like that. In this case, the localization set is no longer a polyhedron, but that doesn't really matter. I mean, in particular, things like ACCPM, they work just fine.

So these are kind of obvious and the ones, however, that are quite useful, and also quite strange and mysterious, is the idea of dropping constraints. So dropping constraints would go like this: after a while, you generate a big pile of constraints. So what happened was, you started with a whole bunch of inequalities. At each step, you have added at least one inequality to your set. So what's happening is the polyhedron, the data structure that describes your current ignorance, is growing in size as you go. And that means, for example, whatever work you're gonna do on it is also gonna grow in size. I mean, like grow linearly or polyhedrally – sorry, like polynomially. But at least it's gonna grow linearly. When you add these, there's the option of deleting dropping constraints. So then how do you drop constraints? The most obvious this is the following: here's your current set. And here are some redundant inequalities. So one obvious thing is to call a method on your current polyhedron, which is minimal. And what minimal does is it goes through every inequality, and drops the ones that are redundant. By the way, how do you find a redundant inequality? Suppose I give you a polyhedron and I point to the 15th constraint and I ask you, "Is it redundant?" How do you do that? How about that first one? How do I know if the first inequality is redundant in a polyhedron? Got any ideas?

Student: If it's linearly dependent on other columns?

Instructor (Stephen Boyd):No, that might be wrong, because the b it connects into. But that would be one. That's how you detect linear – that's linear equality equations being dependent. But that's – it beats relatedness, but there's actually really only one real way to do this.

Student: Consult an LP and see if it's strictly feasible?

Instructor (Stephen Boyd):You got it. Here's what you do. You want to know if a1 transpose x less than bi is redundant. Actually, there's a very sophisticated way to do it, but this is very simple, basic, everyone understands it. Here's what you do. You maximize a1 transpose x over the remaining inequalities. That's an LP. Now, the optimal value of that LP is either less than or bigger than b1. If the maximum of this over b1 is less than or equal to b1, what can you say about the inequality? Redundant. So that's one way to do it. Otherwise, actually, what that method will do is it will either produce a point that satisfies all the other inequalities, but not that one, or it will produce, basically, a certificate proving that inequality's redundant. And actually, if we get back to your suggestion about linear dependents and stuff like that, there's more sophisticated ways. In fact, is it relevant? I don't know. That's a great homework problem in 364a, but I guess this isn't 364a. Anyway, the question would be, how do you find out if the first five inequalities are redundant. You should think about that. Okay. You could implement a method that was called polyhedron.reduce, and it would just go through and solve one LP for each constraint. So it would check an LP for each constraint and drop that. Now, if, by the way, the subgradient calls are fantastically expensive, the subgradient calls are cutting-plane oracle calls. If the cutting-plane oracle calls require a network of computers to fire up and the wind tunnel at NASA aims to get fired up and all that kind of stuff, then no problem calling localization polyhedron.reduce. No problem. But in a lot of cases, that would actually be too much – to solve whatever k LPs would actually – anyway, okay. There's gonna be cheap ways to drop constraints, and there's safe and unsafe constraint dropping. Safe constraint dropping goes like this: you drop constraints that, without actually solving an LP, you know are going to be redundant. We'll see how that's done with analytics and cutting plane. Very cheaply. That's one method.

And the other method is the unsafe method, is you just drop constraints, whether or not they are redundant. That method works, by the way, really well. Capital N here. It seems sort of like the word on the streets is that if capital N is five little n, everything's cool. There are some proofs of this and all that, but they're very complicated, as you might imagine. When you are dropping constraints that are not redundant, you are increasing the volume of that set. That's our merit function. So now the proofs get very tricky. And you have to show that your constraint dropping method is increasing the volume of the localization set more slowly or roughly than your cutting planes are reducing it. So that would be it. But the fact is that we'll see something like N equals 3 to 5 n works, apparently, very well. Okay. In fact, there's a shock. We'll see it next lecture, which is gonna come up very soon. What's weird is you can actually drop constraints, and the method will do better. I know it's a bit weird, but it's also true. It can happen. Okay.

Another variation on this is so-called epigraph cutting-plane method. If you have – it works like this. You put the problem into epigraph form like this, and what you're really going to do is get a cutting-plane oracle for this problem here. So this is the problem. And what you want to do is you want to separate x and t. So x should be considered as your current proposal for x, or something like that. And t is to be considered here and upper bound on the objective. And you're gonna separate this from x*, p* where p* is the optimal value. Now the way to do that is this, if the current point is infeasible for the original problem, and violated the jth constraint, you add a cutting plane that is associated with that violated constraint. And this is a deep cut here. I guess this is positive by definition. This is a deep cut here, this one, where you get a subgradient of the violated constraint here. And this is a deep cut here. And you can also just put this with zero here and that would be a neutral cut for the same problem.

Now, if x(k) if feasible for the original problem, you can actually add two cutting planes. And the two cutting planes you can add are this. This is a deep cut here. This is a deep cut that you add in x and t here. And this here is actually something that says that the current point, that's the objective value because feasible, t, which is supposed to be an upper bound on the optimal value, is clearly less than that, so you can add that one, as well. And this requires getting a subgradient of the objective. So that's how you do this in a – you can run a cutting-plane method in the epigraph. Okay. Now you can also get a lower bound. We'll get to some of these in a minute. This is related to something called Kelley's method. But suppose you evaluated a function, f, and a subgradient of f at q points. So it doesn't need to be convex, but of course, it has to have subgradient. What that means is that f(z), by definition, means f(z) is bigger than this, all z. So it says that this is an affine global lower estimate on f. Well obviously if f(z) is bigger than each of these, it's bigger than the max. So you can write it this way. And this piecewise linear function here, we're going to call that f hat. So it's a very useful way to think of what it means if you get a subgradient at a bunch of different points, a single subgradient will give you an affine lower bound. If you call multiple subgradients, if you get multiple subgradients at different points, for example, if its an algorithm and your k steps in, and at each step you've evaluated subgradient, you actually have a piecewise linear lower bound. And that's this thing. And I'll draw that. It's kinda obvious what it is. But let me just draw it just to give you the picture of this. So here's the picture of that, other than R, which is kind of a stupid example. Here's your function. And let's suppose I've evaluated the – radiant, sort of here, you know, and here, and here. And what you do is you simply draw these first-order approximations, like this. There you go. Something like that. And then this one looks like that. And this function right here that goes like this, that's a piecewise linear lower bound on your function. By the way, if you minimize that, you get this point here, and that's clearly a lower bound on your function. The point is that once you evaluated a couple of – not a couple, but in fact, n + 1 subgradients of a function in Rn – if you evaluated n + 1 subgradients in Rn, then generically, you can now produce a lower bound on that function. The cost is solving an LP. So what happens is this, if you replace f0 and fi by their polyhedral under-approximators, then solve this problem, which is an LP. Well, it's not an LP, but it can be converted to an LP. You know how to do that.

So if you convert this to an LP and solve it, I guess this one's trivial, and this one you introduce one epigraph variable t, you get an LP. You solve that and the optimal value is a lower bound. In fact, it's this value here. By the way, we didn't mention this. I'll mention it here, because it's an interesting method. If you solve that LP here, then you get a lower bound on the optimal value, and if this is your new point at which to query, you get a nice and you get a famous algorithm. So if your next query point is here, and it goes like that, that's called Kelley's cutting-plane method. And it's actually quite a good method from the '50s, I think. Either '50s or '60s, one or the other. That's Kelley's cutting-plane method. And what would happen is when you add this, your lower bound goes up there. And your next point in here. And you can actually see in this case it's a smooth problem, that it's gonna do pretty well. It's not hard to show that it converges and so on. A small modification of Kelley's cutting plane method, actually, makes it work really well in practice, and it's not clear from this picture why it wouldn't. But of course, pictures are generally in R2 and R3. And that's not where – there's no interesting problems. These aren't real optimization problems. Real optimization problems go down in like R10 minimum. They get mildly interesting in R100. And actually are real honest problems in R100K, or something like that, where your intuition in drawing pictures isn't going to cut it even close. Okay. Let's see. So we'll go on and do the analytic ACCPM. Okay. So we'll look at ACCPM. I think this is sort of – I'm not sure, but I guess I'd probably put this at the if someone needed to run one of these non-differentiable methods or something like that, a cutting-plane oracle, or something, and you actually really needed to do this, I think this is probably – depends on the situation – but this is probably the algorithm of choice. We'll look at that. And there's a couple of things we have to talk about. So the first is, is just to remind you what the analytic center is.

Again, this is not correct. Oh, unless – well, this is wrong. This notation universally defines p to be this set, and this is not a function of p. It's really a function of the inequalities. But we'll leave this and just understand that that's slang. So what you have to do is you have to maximize the product of the slacks in a polyhedron. And that can be done by Newton's method. This is completely smooth. It's actually self-important. So that means if you're gonna give a complexity theory result for it, you're already well on your way. And here's the ACCPM algorithm. You start with a polyhedron. By the way, it doesn't have to be even a polyhedron, but it doesn't matter. You start with a polyhedron known to contain the target set. Then you go this, you calculate the analytic center, you query the cutting-plane oracle. The cutting-plane oracle can tell you that you're done, in which case you quit. Otherwise, it returns a cutting-plane inequality, that's this, and that is appended to p, the description of p. Now if this polyhedron is empty, then you can announce that capital X, the target set, is empty and quit. Otherwise, you increment k and repeat. So that's ACCPM. Let's see, for constructing cutting planes, I think we've actually looked at a bunch of this before, but we'll go over it anyway. If you have an inequality constrain problem, then you do the following: if x is not feasible, you choose any violated constraint and do an added deep cut. If it's feasible, you can do a deep objective cut. And the deep objective cut is given by you maintain the best value you've found so far, and you add this. So that's a deep objective cut. You get the analytic center. You have to solve this, and there's lots of ways to do that. But there is one issue, and that's this. You're not given a point in the domain of this function. So that's actually the challenge here. And there's lots of ways to get around this. One, of course, is to use a phase one method to find a point in the domain. And that actually will have the advantage here of if you do that, the advantage here is going to be that that will surely, in a graceful way, determine if this thing is empty. If it's not empty, you can use a standard newton Method to get the analytic center. You don't have to use an infeasible start Newton method. And there's one more method, which is you can actually solve dual. You do the dual of this analytic centering problem. And if you think about why that will work very nicely, you have a set of inequalities, the dual has a single variable for each inequality. If you add an inequality, you're actually adding a new variable to the dual. And so, there the initialization is easier.

So that's yet another method. Infeasible start Newton method, actually, the truth is I didn't cover it, really, at all in 364a. It's in the book and stuff like that. But here it is. I covered it maybe too fast. You want to minimize this separable convex function subject to y equal b minus ax, so we introduce this new variable, y. And the way it's gonna work is this: the infeasible start Newton method does not require the points to be feasible. The traditional Newton method you must start with a feasible point and every direction you generate is in the null space of a, and so the points remain feasible. So it's a feasible method. Infeasible start Newton method, you don't have to be feasible. So here's the way you would solve this problem. You initialize x as anything you like, zero, the last x you found, some guess, or something like that. And a good – the radical method is to initialize all these y's to be, for example, one. Why not? That's a point nice and squarely in the domain of the objective, which is the sum of logs. When you do that, obviously, this equality constraint is not satisfied, unless you're really, really lucky. Unless it turns out that your x. So this is the initialization. So here would be a common choice. Suppose you had to guess x previous. You could choose y to be the following: you would evaluate each bi minus ai transpose x, that's the ith row here, and if that's positive, no harm. Leave yi to be equal to that. If it's positive, it's in the domain of log y. And there might be a little threshold value, but small enough you don't. Otherwise, you just set yi to be one. So what happens, then, is these equality constraints here are some are already satisfied. But anyone where this is not the case are not satisfied. So that's how that works. By the way, this infeasible start Newton method is very cool. It has the property that if any constraint, literally coordinate by coordinate, is satisfied at any step, it will be satisfied forever afterward. So for example, if there's 100 inequalities here and 80 of these are satisfied immediately, but 20 are violated – or let's make it even simpler.

Let's do cutting-plane method. So in a cutting-plane method, you just solve the problem with 99 inequalities. You added one inequality, and now you have 100. You satisfied all the 99, because you were actually at the analytic center of those 99. However, the new constraint you add, you for sure violated. Well, you might just have equal to zero. It might be just – if it's a neutral cut. If it's a deep cut, you're guaranteed to violate that point. But then what's cool about that is only that one – you have 99 equality constraints here are satisfied and only one is not, which is the new one. Everybody understand? Okay. So the method works like this, you define the primal dual, the primal residual is y plus ax minus b. Now of course, in standard Newton method, you are always primal feasible. So you don't even have the concept of Rp. Rp is just zero always, initially, every step Rp is zero. And it stays Rp because your Newton step is in the null space of a. So no matter what step you take in the null space of a, you continue to satisfy ax equals b. In the infeasible start Newton method, that's false. Rp starts off – in fact, it's guaranteed of the original point was not feasible, starts off non-zero. And the dual residual is a transpose nu ng plus b plus nu. So these are the components associated with both x and also with y. Okay. And here the gradient is very simple to calculate. It's simply the vector, which is basically one dot with a y minus sign. That's the gradient. Now the Newton step is defined as follows: you actually linearize these equations at the current point. And you'd linearize the equations Rp equals zero and Rd equals zero. Now, this equation is already linear, so there's hardly any linearization going on there. This one is not linear, because g is a non-linear function of y. And you'll get something that looks like this. Now, h here is the heshen of the objective, so that's diagonal. You get a and you get a transpose and so on. Now that's a large system. The system is size, I guess, let's see, I can't remember the dimensions of n and m, but y is going to be m, so that m by m – this is not a small system here. It's quite big. However, this should be screaming structure at you. Screaming at you with structure. There's tons of zero's, there's i's. This is diagonal. There's a and there's a transpose. So you should have an overwhelming and uncontrollable urge to apply a smart method to solve these equations. And in fact, you can do this by eliminating various blocks and things like that. Instead of having a complexity, which is cubic and m plus n, you'll actually do much better.

What'll happen is, you can actually solve them by calculating delta x this way and delta y that way, and so on. Now when you calculate delta x, you have a couple of different methods for actually computing delta x. One is you actually form this matrix. And in fact, here you can exploit any structure in a to form this matrix. But this matrix, you can – even if the matrix is dense, you can exploit structure informing it. That would be one method. If this multiplied out matrix here, if this heshen is actually has structure, you could use a Sparse Cholesky Factorization or dense or something like that. And the other method for solving this, and this would be probably the method that would be not frowned upon by people who do numerical things, if you just simply worked out what this is, this delta x is the solution of this least squares problem. If you actually form this matrix, a transpose ha, I guess that's a – most people who do numerical things would consider that a misdemeanor. It's called squaring up. So whereas, over here, you never actually form this matrix, which has, whatever, the square of the condition number of this thing here. So this is another way to do it. And by the way, the h halves don't have to scare you, because they're actually diagonal. And this is nothing but a diagonal least squares problem. So here's the infeasible start Newton method. You start with a point, x, arbitrary. And y has to be positive. It could be all ones or that initialization we talked about before. You have your usual parameters. And what you do is you calculate the Newton step in delta x, delta y, and delta nu. And you do a backtracking line search. In infeasible start Newton method, you cannot use the function value to do a line search. That's completely obvious because you're infeasible, so your objective, actually, can be very, very good. Very good. It's just totally irrelevant because you're not even feasible.

And you really can't – anyway, it would make no sense. And in fact, you could easily see the method would fail, because you initialize it at any point that's infeasible, but has an objective value that is better than the optimal objective value, which of course, is higher because it requires feasibility. Then if you did a line search or just a numeric function, the function value you'd never get to the optimal point. That would pay off. Instead, what you do is you do a line search on the norm of the residual. And that's easy enough to do. The simple way is something like y plus t delta y is not positive, you could do this. By the way, related to the question you asked earlier about this, it's actually must more common. In fact – it doesn't matter. You can either t time equal beta in a while loop here, and simply find out if you're positive. But the fact is, you'd probably immediately, if you're writing this in a real language, what you'd really do is you'd go here, you'd get y and you'd get delta y, and you'd quickly calculate the largest t for which this is still the case. And then you'd take the minimum of that t multiplied by 0.99 and 1. So that'd be your minimum. And then you'd check. You'd drop down to this one. And then you'd t times equal beta over here until this is satisfied. And then you update. Now there's actually some pretty cool things about this method. This method has several properties. I'll just mention them and you will see them at some point soon. Maybe on Homework 5. I don't know. You'll see them. One interesting point about it is this, if you ever take a step size of one, then all equality constraints become satisfied on the next iterate. And they will always be satisfied from then on. That's one property. Another one is if any of the individual equality constraints is satisfied, ever, at some step, it will be satisfied forever on.

Actually, that's unlikely to occur during the algorithm, so probably the better way to say that is something like this, any equality constraints initially satisfied will remain satisfied during the algorithm. Then there will be a point at which you take a step size of one, after which all equality constraints will be satisfied. That's how that works. Okay. I think I already said all these things. There's a couple things to mention. This method does not work gracefully if there is no feasible point. Feasible means there's no point in the domain. So there's no positive y for which y equals ax minus b, or something like that – b minus ax, y is b minus ax. There's no positive y that satisfies that. And then, of course, obviously, what happens is the Newton method churns along. It obviously cannot take a step length of one, because if it took a step length of one, the next would be feasible, but that's impossible, so it doesn't. So there are methods you can do to recognize infeasibility and stuff like that, but I think the best way to say it is that infeasible Newton method is designed for things, which are going to be feasible. And they just don't gracefully handle infeasible things. Let's go back to this pruning and I can say something about this. This is also material from 364a and in the book, but it's a big book and 364a we went way fast, so I didn't expect anybody to get it. So we're just doing backfill now. These are actually very, very good things to know. So if you have the analytic center of a polyhedron – again, that's slang – then if you calculate the heshen at the analytic center here, and that simply looks like this. It's nothing but this thing here. The gradient, of course, of the analytic center is zero, obviously. That's the definition of it. Then it turns out the following is true: that defines an ellipsoid. In fact, let me say something about the other one. Because there's actually some very good things to know about these. These are in the book, I think, but I can't really remember. Here it is. Here's a polyhedron. Here's a point, which is not the analytic center. Calculate the heshen, that's this thing up here. The heshen of the log barrier like this, like that. That's not the analytic center. And this ellipsoid here is the set of x such that x minus z transpose h, this is less than one. So you'd simply calculate that heshen right there, that formula, without the stars. You calculate that heshen. This ellipsoid is guaranteed to be inside that set, always, whether or not you're at the analytic center. That's gonna come up later, because we're gonna look at another interesting method, very interesting method called the Deacon method.

So at any point, if you calculate the heshen of the log barrier function, then the ellipsoid with a one, defined by that heshen, is inside this set. Period. Let's go to the analytic center. If you go to the analytic center, let's say that's here. Then this result holds, because it actually holds for any point. So if I draw the ellipsoid like that – I made it too small, by the way. You'll see why in a second. If I draw it with a one, it's inside the set. And what this says is if I draw it with an m2 – let's see, one, two, three – how many of these? Six? So if I draw this ellipsoid with a one, I'm inside it. This result says if I puff that up by a factor of m, which is six in this case, you will cover the set. No, I'm gonna miss it. I didn't draw it big enough. But theoretically, if I drew this right, and I puffed up that set by a factor of m, I would cover the set. So if you calculate the analytic center, you get, actually, an ellipsoidal approximation of your polyhedron. And it's an ellipsoidal approximation that is within sort of a factor of m in radius. In other words, you have an inner ellipsoid, you have an ellipsoid guaranteed to be inside. Puff it up by a factor of m, guaranteed to be outside. By the way, the fact that this is guaranteed to cover the set is not – that's actually interesting, because if I just give you a polyhedron, and I give you an ellipsoid, and I say, "Would you mind checking if your polyhedron is inside my ellipsoid," that's NP hard, because you have to maximize a convex quadratic function over a polyhedron. So you can't even check if a general one was. So in this case, there's a fast way to verify that this is true. I think that this is easy to show. It might even be – you may even do it yourself. Who knows? So what this allows you to do, now – and you're gonna calculate this if you use analytic center cutting-plane method, you had to calculate this anyway, because that's that ah a transpose. That's all that is.

So you've already calculated all this stuff, anyway. And it turns out that you can now define something, which is this irrelevance measure. And it's this: it's bi minus ai transpose x* divided by this thing. Here's what you can guarantee now. It's actually the normalized distance from the hyperplane to the outer ellipsoid. And I guess I'm not ever sure I need the m here. Is that right? No, I guess you do. Maybe we should make eta go between zero and one. Anyway, I don't know. So here's what you're guaranteed. If eta is bigger than m, then the constraint is redundant, and that you know without solving an LP. And by the way, it's super cheap, because the fact is, I think you've actually already calculated at the last step of Newton's method, I think you've actually already calculated all these numbers. So I think you have, but we can check. But I'm pretty sure you've already calculated all these numbers anyway. Or you're very, very close to getting them. Well, we'll leave it. You have, because you've actually factored – it's cheap, anyway. You've factored this to calculate the last Newton step. So again, if you're doing direct methods, you've factored that, so once you've factored that, computing this is actually the norm, you have a Cholesky factor of that, this is the norm of L minus 1 ai and that's one back substitution.

Student:

[Inaudible].

Instructor (Stephen Boyd):Hang on here. If you have a whole bunch of constraints in describing P, the analytic center cutting-plane method says you've gotta calculate that heshen. You have to calculate a Newton step to calculate the analytic center. So that's done. You're gonna already be doing that. Now what is true is that there's no reason to believe that any of – this is safe constraint dropping. In other words, if you drop a constraint based on eta being bigger than m, it's safe. You don't have to avoid your complexity theorist friends and things like that. The proof is still very, very simple, because the pruning does not change the set. It changes the description of the set, because you dropped irrelevant inequalities, but it does not change the set. So you can hold your head high if someone grills you and says, "Are you absolutely certain this method will converge?" Then you can say, "Yes." So that's safe. There's no reason to believe, actually that this will work. So in fact, what's generally done is you keep 3n to 5n constraints, and you just keep them in order by this. So you drop the ones with the – you keep the 3 to 5 n constraints with the smallest values of etai. Now, I can describe it another way. Let me describe it another way. Here's another way to describe this method. The method goes like this, and in fact, it's a perfectly good way to describe it. It goes like this: here's your polyhedron like that. And you calculate the analytic center. Now, you change coordinates so that the heshen of the log barrier at that point is i. So in other words, what that does it you make your set – you're rounding your set. You're making it more isotropic. Now, what you know now is that a ball of radius one fits inside your set. But a ball of radium m is outside your set, covers your set. So that's what's happened. And so now, the irrelevance measure is merely the length in those transformed coordinates. So you can say all sorts of interesting things about it.

Actually, it says that no value of eta could possible be less than one. It's impossible. Because at that center, the ball of radius one fits inside your set, and eta is the distance to the first hyperplane. So all the eta are bigger than one. Any of the eta's bigger than m are guaranteed to be redundant. That's the picture in the transformed coordinates. Okay. And you get a piecewise linear lower bound. We can go through that. In ACCPM you get that trivially, because you get the subgradient. You can form the piecewise linear approximations. We know this. And you can actually get this piecewise linear lower bound in analytic center cutting-plane method. That's not surprising because the analytic center cutting-plane method calculates an analytic center, but an analytic center is a step and a barrier method. When you finish a barrier method, you get a dual feasible point, which we know. That's the basic idea of a barrier method. And so it's not surprising that you're gonna get a lower bound. The details don't matter, I think. But the point is that this x(k+1), that's the analytic center of the big thing that's gonna satisfy something like this. And this is maybe better done in the notes, rather than in slides. Anyway, you end up with a lower bound from all the stuff you just calculated. And it's based on the idea that you calculated an analytic center, analytic centers are associated with center path, on a central path you've actually calculated dual feasible points, whether you know it or not. Dual feasible points means you have lower bounds on the problem. And so when you put all that together, that’s the lower bound right there that you’ve actually calculated. Okay. And ACCPM isn’t a decent method. So you keep track of the best point found and so on. Let’s just take a look and see how this works. So here’s a problem instance with 20 variables, 100 terms. So I guess it’s a piecewise linear function. I think it’s the one we’ve been using all along. So this is what it is. And I guess the optimal value’s around one. So that means 1 percent accuracy. That means 0.1 percent and that means like coordinates of accuracy. And this is f minus f*. Obviously, you don’t know this at the time. In this case, it’s an LP, and you solve the LP to find f*. And then this shows you the progress. And you and see that it’s not a – we can say a lot of things about this plot if you compare it to the subgradient plots. This appears to be, and that’s actually correct, so they’re proofs of polynomial time converge and everything like that. This appears to be what we’ll call linear convergence, because that’s a log scale and there’s like a line that looks like that.

And that means something like this. That roughly each iteration gives you a constant factor of improvement. You can work it out by calculating the slope there. Now, subgradient is very different. Subgradient you’re talking like one over square root of k or slower convergence. And that doesn’t look like this. It looks like that, and then it gets very, very, very slow. Okay. This is actually, if you look at just the best point you found so far. We’ll come back and look at some of these later. And this shows you the real effort. The real effort is the number of Newton steps required. And in fact, you can see here that the infeasible start Newton method at some point jams. So each of these is like a constant here, and the width of a tread is the number of Newton steps required to find a feasible point and then calculate the analytic center. By the way, things like this, these are just appalling. That’s 200 Newton steps or something like that. So this is sort of the amateur implementation. The correct method of dealing with this, which would have made this thing go like that, would have used something like a phase one there. So phase one would be like far superior. This is just to show. And then this shows the bound and everything like that. We’re out of time, so we’ll come back and get to this later, although I will just show one more thing, which is this, and we’ll talk about it next time. If you drop constraints, you do very well. Okay. We’ll just continue here next time.

[End of Audio]

Duration: 78 minutes