ConvexOptimizationII-Lecture07

Instructor (Stephen Boyd):Well, this is – we're all supposed to pretend it's Tuesday. I'll be in Washington, unfortunately. So today I'll finish up and give a wrap up on the Analytic Center Cutting-Plane Method, and then we'll move on to, actually, one of the coolest topics that really kind of finishes this whole section of the class, and that's the Ellipsoid Method. So we'll look at this, and I'll try to make clear what is useful and what's not. The Analytic Center Cutting-Plane Method is useful. When you have a problem that you need to solve where you really do only have access to something like a cutting plane or sub-gradient oracle, and for whatever reason – you have to look at those reasons very carefully.

But if you have such a problem, this is an awfully good choice, and it's going to beat any sub-gradient type method very much. This is going to have much more computation, obviously, per step, more storage, all sorts of stuff like that, compared to sub-gradient method, which involves essentially zero computation and zero storage. They're way, way, way, way better than sub-gradient methods. Okay, so here's the Analytic Center Cutting-Plane Method. You're given an initial polyhedron known to contain some targets which might be feasible points, might be epsilon sub-optimal points.

Doesn't really matter what it is. You find the analytic center of the polyhedron, which is to say more precisely, you find the analytic center of the linear inequalities that represent the polyhedron. And you query the cutting-plane oracle at that point. If that point is in your target set, you quit. Otherwise, you add, you append this new cutting-plane to the set. Of course, what happens now is your point X (K+1) is not in P (K+1) by definition. Well, sorry. It might be in it if it's a neutral cut, but it's on the boundary.

What you need to do now is at the next step you'll need to calculate the analytic center of that new set, and I think – I won't go through this. There's a lot of different methods. Infeasible Sartinutin Method is the simplest one. Maybe not the best, but we'll go on and just go to a problem, and I'll show how this works. So this is a problem we've looked at already several times. It's piecewise linear minimization. It's a problem with 20 variables and 100 terms, and an optimal value around one. Let me add, just to make sure this is absolutely certain and clear. If this problem were – if you just had to solve a problem like that, it goes without saying you would not use something like an analytic center.

You'd just solve the LP. Let's bear in mind that every one of these iterations requires an effort that's at least on the order of magnitude of simply solving this problem to ten significant figures right off the bat by Barrier Method. So your code from last quarter, your homework code which shouldn't have been too many lines, will actually solve this problem very, very quickly. In fact, anyone want to take a cut at how fast it would be?

You have to solve – you have to minimize piecewise linear function 20 variables and 100 terms. It's gonna be an LP – when you add another variable, it's an LP with, I don't know, what is it, 21 variables and 100 constraints. Something like that. So let's say it's 20 variables, 100 constraints. Dominating cost is gonna be actually forming like A transpose

HA. So forming a 20 by 20 matrix, which is actually a matrix – 100 by 20 matrix multiply, or something like that. So that's 100 times 20 squared, and you're gonna do maybe 20 steps. That's an interior point method. How fast would that be, just an order of magnitude?

What do you think? Microsecond. Well, it's good. You're guessing sort of the right numbers now. You might be a little bit low. Actually, when you get down into the 10 and 20 variables, the N cubed, the various blast extrapolations start falling off. But let's – based on blast extrapolation – well, I can tell you how I'd do it. It's 20 squared times 100, and the way I do it is I think 1,000 cubed. Let's just say a second. That's way more 'cause a Cholesky factorization in this – I don't know. It's .18 seconds on my laptop, so it's not a big deal.

But something else matrix [inaudible] might be half a second. So let's call it a second. Let's round up and say 1,000 cubes is a second. You'd get a factor of ten right here because you're doing 100. You get ten, and then you get 50 squared, 2500. So it's 25,000 times faster than one second. Let's say that's 40 microseconds, and let's be generous and say you're gonna do 20 of those steps. I think I did that. So 40 microseconds [inaudible] 20s. So the answer is a millisecond, so if there wasn't a microsecond, but your answer was in the right spirit.

All this is just to point out that this particular problem could be solved easily through very high accuracy in under one millisecond. These are very good things to know, by the way. Very few people know this, I promise you. If you go around and ask people who – if you ask the authors of widely used packages, they have no idea how fast you can solve small problems.

Student: Is this leaving – every time you make a cut, is this leaving?

Instructor (Stephen Boyd):Yes. So in this example, this is gonna leap all constraint. So it must be understood here that when you start, we probably had 40 constraints because we probably put a box around it or something. I don't – I'm sure we did that. So we probably put a box around it. This has forty constraints, and in the classic ACCPM thing, how many constraints have you got over here? That's it. 240. So the number of – back to the size of the problem solved over here is six times as big as the problem here. It's 240. So we'll get to a bunch of this. We'll talk about it and just see how it works.

Nice thing you see here is you don't see that horrible slow-down you don't see in a sub-gradient method where you kinda go like this, and then kinda slow – get into this nice, one over K, very slow convergence. What you do is you see something that looks much more like an interior point method, except that would not be 200 steps; it'd be 20, or something like that. You'd see nice, geometric, linear conversions.

Okay, now the previous one shows – this is F of X, K – F*, and what this shows is the best of those. So this is probably what matters because when you terminate the algorithm at any point, you're actually – of course, what you're gonna return is the best thing you've seen so far. These long things here – of course, well actually, what do they mean? What does the long horizontal – what does a segment mean in this picture? This is iteration number versus sub-optimality. Your best one.

Student: Trying to satisfy the constraints.

Instructor (Stephen Boyd):No, this is – we keep all constraints. What does it mean?

Student: It's infeasible?

Instructor (Stephen Boyd):No, it can't be infeasible, right? Because we just – well, if it's infeasible, we stop. But that can't happen for this piecewise linear minimization problem. No, what this means here is that your F-K best has not changed. It means, in this case, that for ten steps you produced points that actually were worse than the best thing you've already seen. This is just underscoring the fact that this is a non-descent method. So a flat part means that some people might say that you failed to make progress, unless you failed to make progress in terms of finding a point with a better objective value. Actually, how have you made progress? What actually did happen in one of these little steps?

Student: You have a more constrained space.

Instructor (Stephen Boyd):Yeah, so your P-K got smaller. So, technically speaking, your ignorance was reduced over that step, but you failed to produce a point. You were lucky. You had a good point, but your ignorance – and that ignorance is going to translate into future good point values. So that's what those mean. Okay, now if you want to compare this to Newton's steps – and I think if we go back and forth, we can look at the two pictures. That's about 200 iterations, and here you can see that we did about, I don't know, about ten, eleven Newton steps. About eleven Newton steps per – so it took, on average, about eleven Newton steps to recalculate a new analytic center.

By the way, I should mention that we're – this is not tuned. We did have a fairly small tolerance for analytic centering. In other words we probably calculate the analytic center to land the squared less when you minus six or something like that. Needless to say, that means that the last three Newton steps we took in every one of our centering was a complete waste of time. Because remember, the goal here is not to calculate the analytic center. The goal is to get enough query point to – so these numbers, you could probably get it down to five Newton steps. I'm just saying if you actually wanted to do this.

So this is just sort of straight out of the box. No – well, all the code is online, so you can look at it as well as I can.

Student: For the infeasible Newton's, don't you have to wait until you get feasibility?

Instructor (Stephen Boyd):Yeah, and in fact, we don't have that here, but I might actually go back in the code and actually get some of the statistics. Like how many steps on average – the feasibility. Precisely right, that would be a really good thing to know. I may go back and do that. It'd be fun. You can, by the way, since the code is all there. Yeah, so that would be one question. And you can see, by the way, a little curvature means that as this thing progresses, the centering are getting a little bit harder. There's certain – you're on a pretty steep thing like this for a while, and then the slope here is half as – if you compare the beginning and the end, the centering have gotten harder.

By the way, what is the effort per Newton step? Is this a valid – does this track time well? Is that – I mean, that's Newton's step. Does it track time? Is that a valid – so, actually, how does time grow with these?

Student: Well, you're adding an extra constraint.

Instructor (Stephen Boyd):Absolutely.

Student: Every time, so yeah. I think it's –

Instructor (Stephen Boyd):So it's going linearly. The problem size. What's the competition complexity of the Newton step as a function of the number of inequalities? It's what? You guys have to get good at this. Here, you want to know the sloppy way to do it? Someone just walks up to you on the street and says, "Quick, quick, what's the complexity?" The big number times the small number squared because that's the answer for lease squares and any problem like that, right?

By the way, that's conditioned on the fact that you've done it right. So if in fact it's the big number squared times the small number then it's wrong. So that's my quick rule of thumb. Then later, when you can go catch your breath or whatever, you can go work out the Newton step, and there's 20 of these. But that's just a good working number. You know, if M is bigger than N, it's M-N squared. That's the answer for Lease squares, by the way. Lease squares, Lease norm, everything. So therefore, the cost per Newton step – what's happening is you have 20 variables.

That's a small dimension, and then you have the big dimension is the number of constraints. And it starts at 40 and ends at, I don't know, 240. What did we say? Goes up by factor of six, and roughly speaking, it's growing linearly. So as you go along here, that's a six times bigger problem in terms of big dimension. Therefore, if you do things right, it's a big dimension linear, quadratic in the small dimension. The cost of a Newton step from here to here actually goes up by a factor of six, linearly. So what you do is you'd do a quadratic distortion of this axis to give you the actual time. Make sense?

Because each step over here costs six times what a step over here costs. A step in the middle costs three times. So all I would do is take these numbers and spread them out quadratically. I'd have an affine expansion. Everybody cool on this? So that would be time. We didn't have it here, but it doesn't matter.

All right, so this just shows you again. This is the base line, nothing funny, no constraint dropping, no nothing, just basic ACCPM, right out of the box. So here what we have done is we have plotted two things. This is the true optimality. Now, of course, when you're running, you do not know this thing. We obtained this by using CVX to solve this in not one millisecond. Let's see what CVX had us interpret it overhead. It's gonna swamp everything. I guess by the time it gets down to STBT3, it's probably 30 milliseconds here.

Wait, super fast. We've got the actual solution, and then use that to judge our progress. This however – so this you do not know in real time. What you do know is that, this number. This is the best upper bound. By the way, the lower bound is also – does not go up monotonically, right? In an interior point method, right, what happens is that every step when you do a centering is you get a better point as your objective value goes down. And not only that, your dual value goes up. So you get the beautiful things where your best point and – you don't distinguish between your best point and your current point because the current point is always the best one. It's a descent method.

But you're lower bounds also go up, so you have no logic in an interior point barrier method, or whatever, that says keep a hold of the last point because it may be a while before you see one that's better. Or in terms of lower bounds, you don't keep track of the best lower bound you've seen because the next 15 you see are actually worse than the one you have. So here, though, you do know this one, and you can see it's off by some factor. I don't know, a factor of eight or something like that, or whatever. So that's all. That's all that this is.

So your stopping criterion might be based on the red dash thing, although I should probably say something. The sub-gradient method has, in general I think it's fair to say from a practical point of view, no stopping criterion. But usually when you're running sub-gradient methods, it's in some situation where you're so desperate to have anything that's all – that does anything, that you're just happy to have something. If things get better after 50 steps, or ten, you know. That's good enough, right? So stopping criterion is a luxury that you, generally speaking, don't actually have when you apply one of these.

The same might be true here in these. Maybe less so, but you still have the same thing. Okay, now we do constraint dropping. All right, so here what we did is – now by the way, keeping 3-N is actually quite interesting because we start with 40 constraints. We start with a box on X. X is in our 20. We put it in a box. I'm guessing that, but one of you with a laptop could actually look at the code and find out. You have a laptop. You don't have to, but I presume that's how we initialize it is with 40 constraints, which are BOTS constraints. So – and this is the progress if you do no constraint dropping the blue thing.

And remember, when you know dropping up here, your polyhedron has 240, or whatever it is. Wait a minute. Yeah, up here. It has 240 inequalities. So that's this thing at R-20, polyhedron at R-20. Kind of a small one, by the way, at this point. It's quite small, and it's got 240 inequalities. By the way, I have no idea how many of the 240 are redundant. I presume the original 40 are redundant. The original 40 were the boxes, and the idea is we just made a big old box. So it would kinda be bad if those were still active. So those 40 are probably redundant. Probably a lot of redundant, but I don't know. We didn't do that calculation.

But we keep 3-N. So we're going to keep 60 at any given time. So by the way, a polyhedron in R-20. What's the minimum number of inequalities that would give you a bounded polyhedron in R-20?

Student: Forty?

Instructor (Stephen Boyd):Forty, well that – in R-20. A 40 would do it, absolutely, if you have a box, but that's not the minimum number. What's the minimum number? Can you do it with eight?

Student: No.

Instructor (Stephen Boyd):What? Someone said no. Why not?

Student: It's 20 –

Instructor (Stephen Boyd):So you need at least 20. If you have eight, then basically there is a 12 dimensional sub-space of things orthogonal to those. And so you're unbounded this instantly in at least 12 dimensions, so you're not gonna be bounded. So the answer is 21. You need 21, and a simplex is the smallest number of – it's the simplest polyhedron that's bounded. Simplest in the number of inequalities, and it requires N plus one.

So the point is it takes 21 inequalities. To just have a bounded polyhedron, we're taking like 60, so it's not like it's a super – this is not an exquisitely detailed description of a polyhedron. And the wild thing is you see this? It's just totally insane. The progress is identical. By the way, what is the change in computation time? Again, not that we would care, but lets figure that out. What's the difference between solving – computing the analytic center for the red guy and the blue. What is it there?

Student: Post for each Newtons.

Instructor (Stephen Boyd):Yeah, I know that, but I want to know the number. You can figure it out. Everything is here. You know all the dimensions, so I want the number. Of course the cost per Newton step is different. That's what it is. Now I want the number.

Student: When you drop, or when you don't drop?

Instructor (Stephen Boyd):Both. The blue is no drop. The red is keep 60, so you're dropping. What is it?

Student:B minus 6 –

Instructor (Stephen Boyd):Do 40 over 60. So the – in this case, you analytic center 240 inequalities, 20 variables. In the red case, you analytic center 20 variables, 60 inequalities. So, again, big times small squared. It's linear and big. 240 versus 64 to one.

Student: What about the complexity of reducing the constraints?

Instructor (Stephen Boyd):Oh, that would solving like an LP per step. Oh, now we could actually do much more analysis in here that would sort of be fun. So you know an Analytic Center Cutting-Plane Method. At each step there is – you get for free – some constraints you can promise. You can actually guarantee are redundant with no further discussion. You don't have to avoid your complexity theorist friends or anything like that. Everything is cool because they're provably redundant. Actually, I would love to have figured out how many – at each step we drop things, obviously. Well, we actually didn't drop things until here, right?

Because we started with 40, and then 41, 42, 43. When we got to 60, we started dropping, and then it looked like that, right? It would be a plot of the number of inequalities. So we didn't have to look the other way for the first 20 steps. After that, when we start dropping, the question is what fraction of the things we dropped were provably or actually redundant. All of those could be calculated by just solving some LPs. Those would be really good numbers to have. We should have them in here. It'd be fun.

It is very unlikely that everything we dropped – oh, would that be right? Oh, hang on. Oh, that's not true, sorry. I was about to say something that was wrong. Good thing I didn't say it. After all this discussion, though, let me make sure that the big picture points are clear. So the big picture points are the following. It obviously works just as well. You don't need the number – there's no particular advantage in convergence you are getting by keeping all these constraints. That by limiting it to 3-N, you're getting the full power. You're getting perfectly good convergence.

The computational complexity like that means that the cost per set here is actually – in one case it's growing, and in another case it's just flat fixed. So that's the main point. By the way, proofs of convergence that handle constraint dropping are quite complicated, as you might imagine. They're quite complicated. So they exist. I'm very happy somebody has done it so that I can tell you somebody has done it. There's a gap in between, I might add. I'm pretty sure there's a gap in between the things people do, and the things people have proof converged.

So I do not know if a method that were – that keeping 3-N is actually convergence. I actually don't know that. Maybe it is, maybe it isn't. But I've seen ones where you drop things, and these are quite complicated, as you might imagine. Okay. Oh, I guess we figured this out on our own already, so there's nothing to say about this. The sort of numbers steps, and the number of inequalities, it's kind of a silly plot, right? But, anyway, it's – okay. So now we get to something interesting.

This is using a mega flop counter in Mad Lab which is notoriously bad and all that, and I think people don't even use it anymore, but it doesn't matter. Just to get a rough idea, we should be able to figure out – actually, we should be able to guess a lot of these things really, really well. So let's guess. Actually, I'll tell you how we can guess this plot. Let's go to the end, and ask what is the accumulated – we already decided that the red thing at this point is six times more efficient than the blue.

Student: Four.

Instructor (Stephen Boyd):Four, okay, sure, whatever. Yeah, four. Okay, so it's four times – and how much faster is it in the middle. Oh, I don't know. It's 60 versus 40 plus 100, 140? Close to two. This is a street fighting method. This is quick, dirty methods, fast. If it's four times more efficient and it's two here, what's sort of the average inefficient? And it kind of grows linearly or something. What's the average sort of advantage? Two, I don't know. What do you think? Well, it's not eight, okay? And it's not four, and it's not one.

Let's call it two-to-one, and let's see if our prediction is pretty good. Our prediction is, oh, hey. What do you know? Think about 150 mega flops, and this took about 70. So that's the picture. So this shows that constraint dropping, if you care about time or whatever, it works this way. By the way, a lot of this doesn't matter so much because you usually use analytic center cutting-plane in a situation where the oracle calls. The sub-gradient calls are very expensive. For example, the oracle calls – in many of the most interesting cases, this is when you're doing distributed optimization.

These are like other computers on the other side of the world. All sorts of stuff is happening. So the analytic center cutting-plane stuff is just like zero. Even for a problem with 10,000 variables, or something. That's how these things work. Okay, now if you do the epigraph analytic center cutting-plane method, now you have 20 variables and 100 terms. And what happens is you divide – so this actually goes way down here.

So you actually are dividing the number of steps required by just a factor of four, or something. And that's it. I don't think I'm showing F best here, am I?

Yeah, there we go. So this is the epigraph one, and here's what it looks like versus Newton's steps, and I don't know what happens here. But you – I mean, it looks like you're doing – there's points here where your doing better, or something. And the word on the streets as I know is that you should – is that the Epigraph Method is better. I don't understand it, to be honest. I'm not quite sure why, but anyway. For whatever it's worth. Okay, so this finishes up analytic center cutting-plane method.

By the way, I should say that we haven't really seen yet the really cool examples where you would want to use these methods. So – and I admit that. That's going to be the next chunk of the class, and we'll see some very, very cool applications where you really would want to use these methods. In fact, you will. So the next topic is just absolutely beautiful. You can guess where it's from.

Student: Moscow.

Instructor (Stephen Boyd):You're right, actually Moscow and Kiev. So it's from Uddin Nemerofsky Shore in Kiev, and so and so. Guess when. Close. Actually, it probably is 60s. They say definitely for sure early 70s, and yeah. I'm sure they knew it in Moscow in the 70s, or some 60s. And it's absolutely beautiful. By the way, it's quite famous, too; not very well known in the West. By the late 70s it was on the cover of the New York Times front page because it was one of the first papers used to show that linear programs could be solved in polynomial time. By the way, with a little star, meaning you have to go read the fine print.

But that was 1979, and they had a hilarious article on the front page. It was really funny, actually, and it said "Linear programming runs the entire world, everything. All airlines are scheduled for it. Nuclear weapons are built by it." It just went on and on and on, and said this is going to totally change your life because now, and by the way, I should add. At that point for 30 years, people had been solving linear programs unbelievably well using a simplex method, just so you know. Amazingly well. It was a non-issue.

They called some mathematician at Rutgers. I mean, actually, the poor guy wrote the article. All he said was there was a complexity theory result, and it was a pretty good one. By the way, up until that point, people didn't know what the complexity of solving linear program was, and they actually introduced a new class called LP, as in P. So you'd actually say this is class LP, and it was as easy to solve as an LP and vice versa. If you had a method for solving an LP, you could actually solve these things. You had to reduct the two-way reduction. So, yeah, they really did, and then it turned out P was LP or something.

What I'm saying is that if you really look into these things you'll see it's much more complicated and all that, and some of the things I'm saying, if interpreted literally, are just wrong. So this is on the front page of the New York Times saying it was gonna change the world, and it didn't. Linear programming was on the front page again in 1984 for barrier methods. This is just absolutely beautiful, this method. This is the one I just talked about. This is Gocheon, who by this time was at Rutgers, used this to show polynomial salability of LPs. So, actually, you're gonna see a code now.

What's funny about this is that the sub-gradient method is to solve an LP. I think that maybe you wrote a – I mean, it's not hard to write a six line code for it, and a sub-gradient method. You can just write a sub-gradient method that solves any LP. Even more, if someone says, "That's ridiculous. It's hard to write an LP solver. These are big, giant pieces of software you have to know a lot." The proof is like a paragraph because it's the paragraph from sub-gradient method. That's pretty cool. Slower than anything, right? But it is nevertheless cool that a six-line program with a two paragraph proof solves any LP.

This was gonna be eight lines, but even scarier according to the complexity theorists this is actually an efficient method. So – and the proof is quite short, which is even more shocking. Each step is going to require cutting-plane or sub-gradient evaluation, so it's the same as everything else. You're not gonna have mono-storage, which is to say it's gonna be Ovan squared. Which by the way is, if you do analytic center cutting-plane method with dropping, the storage is N squared because you have N variables and you store three N, five N constraints. That's all you need. The only state in that algorithm is the polyhedron. So your storage then is like, whatever. Six N squared. That's it.

And your update cost, by the way, is N cubed in that one, obviously because it's N squared. Whatever it is, it's the big times the small squared. You can actually make it faster with rank updates, but that's another story. Okay, so it's got mono-storage. In the Ellipsoid Method the update will involve solving no – well, you will solve a convex problem, but it has an analytic solution, so – and it's gonna be order N squared. And sufficient in theory, and it's slow in practice, but it's actually something like the sub-gradient method, actually, except that it's fed in the sub-gradient method. It's kind of slow in practice, but it is way robust.

You can blow off whole parts of it, and start it in the wrong place, initialize it wrong, and you can give it totally wrong sub-gradients many, many times. It'll just happily lumber along, and work really well. So that's – okay. So the motivation goes something like this. In a cutting-plane method, you need serious computation. I don't know. Do you really believe that? It's ten Newton steps. Each Newton step is a Lease squares problem. Okay, so it's ten lines, but this scares people, and people make a big deal about an analytical formula. You know, like a Kalman filter update versus if there is a conditional or align surge, they get all weird about it.

Of course, that's completely stupid and idiotic if you think about it right because a Newton step is nothing but a Lease squares. So I don't know why people – I mean, it's fine to have an analytical formula, and that's all great and everything, but the idea that these are sharply divided is really, really dumb. It's a distinction that makes absolutely no sense. What only matters is how fast can you carry out that calculation, and what is the variance in that calculation. And nothing else matters.

The variance can range from zero if you're doing a matrix multiply to a little tiny bit of – if you're doing a sparse matrix factorization, considerable. I mean, it depends on the – with a not given structure you have huge variance because it depends on who did the ordering for you, and how much filling you get and all that kind of stuff. Do an Eigen Value calculation for dense matrix. It has a very small – it's order N cubed, but it has an N squared component – or an order N component that is subject to some variation, which is the number of steps required to calculate the Eigen Value. Obviously, if it's N cubed plus a random number times N, the way to summarize that is it has no variance.

Oh, we just talked about this at great length. The other issue is this, is that the localization polyhedron grows in complexity as the algorithm progresses. We already talked about that. Now, I have to say, if you keep it proportional to N, like this – so this second bullet is not a problem in practice. If you do analytic center cutting-plane method, you are going to prune constraints. Okay? That's the way it's – that's how they're done in practice. That's how they work. They appear to work juts fine, in which case this is not an issue.

So this only irritates your aesthetic side, or your theoretical side or something like that. So that's all that is. The Ellipsoid Method is gonna address both. It's gonna have a constant data structure to summarize your ignorance at each step. Analytic center cutting-plane method has a variable size because it's a polyhedron with a variable number of constraints, and in the no cutting method it just grows.

So here's the Ellipsoid Method. The idea is that you're going to localize the point you're looking for in the ellipsoid instead of the polyhedron. By the way, that's actually kind of a very cool idea right there, and ellipsoid actually is – what's required to describe an ellipsoid? What's the size of a data structure in RN? What do you have to describe an ellipsoid? There's lots of different ways to do it? What do you need?

Student: Matrix and a vector?

Instructor (Stephen Boyd):Matrix and a vector. And a matrix is N squared over two, and a vector is like N, so – in fact the matrix is like N and plus one over two. It's N squared over two. That's the answer. So the number – to describe an ellipsoid, the data structure for an ellipsoid requires about N squared over two elements. Oh, let's see. Let me ask a couple more questions. Oh, and an ellipsoid is an interesting set. They're very widely used in RN, for example, if you do statistics or anything like that, it's sort of the first thing that gives you some direction. So let's just talk about localization sets just for a minute.

Here's one: bounding box. So you have a box. Someone says to you during a calculation, "Are you doing navigation?" and you say, "Okay." And someone says, "How accurate are you?" You'd say, "Well, give me plus minus two meters east west plus minus north south in elevation, such and such. Plus minus eight meters or something."

That's a bounding box. Widely used. How many – how big is the data structure required to describe a bounding box? An RN. It's two N. It's an L vector and a U vector. That's it. So that's two N. Okay, so that's – these are at the simple methods, right? Here's an even simpler one. How about just a ball? What do you need to describe a ball?

Student:N plus one.

Instructor (Stephen Boyd):N plus one. You have to give the center and a radius, so it's N plus one. So these are order N data structures that describe geometric sets, and they have uses. They have plenty of uses. Actually, for most people this is the simple – they shouldn't be made fun of. They're extremely useful when somebody says, "How are you estimating the velocity of that vehicle?" and you can say, "Plus minus .1 meters per second."

Everybody understands that. It makes total sense. An ellipsoid is already the first sophisticated thing. You see it in statistics because you talk about confidence ellipsoids, and what it captures is in statistics correlations. So it allows you to capture the idea that you could know X plus minus a meter, Y plus minus a meter, but for some weird reason, you know, X plus Y to ten centimeters. That's this kind of ellipsoid, right? So that's what an ellipsoid does. That's what an ellipsoid gives you. So you're into the N squared things.

How about a simplex just for fun? Just roughly what's the data structure? Think of the simplex. Not a unit simplex, a general simplex. Convex hold and plus one points. I just gave it away. It's N squared, roughly. Then it turns out you have to adjust for that because they're symmetric. It's order N squared. It's actually a big jump when you go from an order N to an order N squared data structure described geometric set. You actually go from not the inability to universally approximate to ability to universally approximate.

Universally approximate means you can – I can give you any old convex set, and you give the inner approximation of whatever data structure you choose that has some bound that says that if give an outer one, if you shrink it, it lies inside. Ellipsoids do – bounding boxes do not do that because you make a little razor thin thing that goes at a 45 degree angle, and you're in big trouble because that bounding box is huge, and you have to shrink it infinitely spar before it fits inside. But an ellipsoid, you have a bound. We did it last – or we saw it squared in, or whatever. N is the factor for an ellipsoid. Squared N of the symmetric. Okay, lets go on, it's just an aside.

So here's what's going to happen. You know – the idea is that you have an ellipsoid to summarize your ignorance. You know the optimum point is a current ellipsoid. Now the nice thing about an ellipsoid is there's no long discussion about centers. It's sort of like an interval, an ellipsoid. So you can say, "Well, let's calculate the analytic center of an ellipsoid. Let's calculate the CG. Let's calculate whatever you want."

They're all the same. In fact, if you have a center of an ellipsoid and it's not the center, then I'm deeply suspicious of your definition of center. So the point is there is no real discussion there. There's just the center of an ellipsoid, and you evaluate a sub-gradient at that point. And what you do – that tells you instantly of a cutting-plane. So what happens is you know that your optimum point – you're in an ellipsoid. You have sliced off a half-plane, and you have a half ellipsoid. And now you know – now, if you did this a standard cutting-plane method, you'd calculate now – by the way, you have a half ellipsoid, and you start about the center. Now the center is varied.

There's an analytic center. There's a CG. Then you do it two steps, and you have now a set, which is an ellipsoid sliced by two planes, and that's – so in fact, there's another step that goes like this, and this is the main point. Here's what you do. You have a half ellipsoid, and in order to preserve constancy of the data structure, you're now going to do something that's equivalent to constraint dropping or whatever. You're gonna set E-K plus one be the minimum volume ellipsoid that covers your half ellipsoid. Okay?

So step four – step three is where you get your knowledge, your ignorance decrease. So this is ignorance decrease, and in fact, by the way, how much does the volume go down between E-K and this localization set?

Student: Factor two N?

Instructor (Stephen Boyd):Yeah, a factor of two. If it's a neutral cut, if it's a deep cut, it's more than a factor of two. So in step three you get at least one bit. In other words, log two of the volume of the localization set, and step three goes down by at least one. The worst thing that could happen is your ellipsoid is cut in half. No matter how you cut an ellipsoid, every plane going through the center of an ellipsoid cuts the volume exactly in half. Your volume goes down exactly by half here. Now, in step four your volume goes up. So four is actually an ignorance increasing step, and then there's just a big drum roll to find out whether or not the ignorance increase in step four is more or less than the ignorance decrease in step three.

That's basically the method. Yeah?

Student: How are we finding that minimum volume ellipsoid? Are we keeping track of half-planes?

Instructor (Stephen Boyd):No, I'm gonna show you. There's gonna be just a formula for it, yeah, worked out by some Russians. The basic one is simple, but they did things like two cuts, and they actually had formulas for deep cuts, and parallel cuts, and crazy things. And then you get into some analytical formulas for those. Those are formulas that believe only a Russian person would work out, I believe. But there are formulas for it. But this one is gonna be simple, and I think you're gonna work it out because we're gonna stick it on the homework. Do we have an exercise on that? We just calculate the minimum volume ellipsoid that covers a half ellipsoid.

Student: We have one on that.

Instructor (Stephen Boyd):Oh, we do? Okay, but not yet assigned?

Student: No.

Instructor (Stephen Boyd):How convenient. What do you know? So here's the picture. Picture goes like this. Here is your ignorance at the K step. Now, by the way, some points in this thing could not be optimal because remember, you're cycling between steps that decrease ignorance and then increase it. So first you decrease ignorance. You call, you get a – this is a neutral cut, like this, and basically, this half ellipsoid, we do not have to look in ever again. So actually, at this half step, we can say that the solution lies in this half ellipsoid. Okay? Our ignorance just went down by exactly one bit.

Now the next step is we calculate the minimum volume ellipsoid that covers it. We're gonna get an analytic formula for that. Here it is. It's this thing. And what's cool about it is things like this and this are very interesting. That’s just slop. We have now – before this step, we know the solution cannot be in this little thing here. It cannot be there, and yet, we’re going to include it in our localization step as we move forward. Okay? So that’s the bad part. That’s the slop.

Now compared to a cutting-plane method, you can state various things like localization set doesn’t grow more complicated, it’s easy to compute a query point – well, yeah because it’s actually part of the data structure of the ellipsoid, right? An ellipsoid is given by a point, the center, and some positive, definite matrix. Okay, fine. It’s an access into a field of a data structure. The bad part is, of course, we’re adding unnecessary points in step four. Anyway, I wouldn’t be here telling you about this if it didn’t actually work. We’ll get to that.

So the first thing is ellipsoid method in R reduces it by section. Good, very good. So R, what’s an ellipsoid in R? It’s an interval. And then it says go to the center of that interval, and get a sub-gradient. It basically tells you the left or the right is your new localization center, and indeed you have gone down by one bit exactly. And now you have an interval, and you call a method that says please find for me the smallest length interval that covers this interval. So I can write that method, that’s easy. In that case there is no slop. But this example back I just showed the visual example shows immediately. In R-2, they’re slop. You are adding points into your localization set that you know for sure cannot be optimal.

Okay, so we’ll get to the formula – the update formula. That’s easy. Now it turns out that this – the new localization set can actually be larger than this in diameter, but it’s always smaller in volume. This is how the drum roll ends, and it’s in fact more than that. It turns out that the volume of the new ellipsoid is less or equal to E to the minus one over two N time the volume of this one. Now, the good news about this number is it’s less than one. That’s the good news. The bad news is that it’s very close to one. Shockingly, that is enough to give you a polynomial time result, which we’re just gonna do in a few minutes.

So does that make sense? So compare this volume reduction factor. In CG, what number goes here? Oh, I already gave it away, it’s a number. Sorry. So what expression do you put in CG here?

Student:163?

Instructor (Stephen Boyd):163, right, which does not degrade, which first of all is a very respectable number. It is not .999. That’s number one. Number two: It does not degrade. The CG method does have a few problems. For example, the sub-routine that you have to call to calculate the CG is far harder than solving the original problem. But still, you can see what happened. So this is a very slim amount. MVE you get something that is much better. I forget. It’s something like one minus one over N. Is that right? It’s in the notes. You just have to look back on a lecture, but let’s say it’s something like that.

It degrades with MVE, but nothing like an exponential. So let's look at an example and see how it works. So you can't see it, but there's some sub-level sets of convex function here. It's not exactly polyhedral, so I don't know what it is. I probably logged some X for something like that. So you start here, and sub-gradient – probably gradient in this case points that direction. And that says that this half circle cannot contain the optimum. I think the optimum is somewhere around here. Okay?

That's your localization set, intermediate. This is half circle, and you take that half circle, and you cover it with the smallest volume ellipsoid that covers it, and that's this thing. Then you move over here. That's this one. You go to the center. You call sub-gradient. You get this. This half ellipsoid has now been ruled out, and you have this half ellipsoid, and then you jump over here. And I think just looking at these, since this is obviously – I reject the idea that there is, I mean – there is no problem minimizing convex function in R-2 because you just grid things. That's a non-problem.

But still, the fact where you have a non-problem, you can sort of visually see that it's going to be slow. It's sort of a hint on these things, and I won't trace through these, but that's the next three steps. You kind of get the idea. So now let's talk about how you update an ellipsoid to cover half an ellipsoid. By the way, I'm doing – this is the formula for a neutral cut. So how do you cover a half ellipsoid with a minimum volume ellipsoid?

We also do deep cuts, and in deep cuts you have to ask how do you cover a cap? Not a half ellipsoid, but something where you cut a fraction off by a minimum volume ellipsoid. There's a formula for that. It's not too bad. But let's look at this one where you update the ellipsoid this way, and I can – let me just tell you why. This is going to be on your homework anyway, so I'll just say a little bit about why it's not that big a deal. Let me ask you this. I can change affine change of coordinates, and this problem is the same.

If I create an affine change of coordinates for an ellipsoid – when I do an affine change of coordinates, all volumes get multiplied by the determinant – the absolute value of the determinant of the transformation matrix, right? Therefore minimum volume – I can transform coordinates, minimize the volume, transform back, and I got the answer because everything just got multiplied. So basically, I don't have to consider this a general thing. I can transform any half ellipsoid I like to half of the unit's sphere, half of the unit ball just pointing upwards, or whatever X one bigger than, or X-N bigger than zero.

So that makes it simple. Now let's just do this visually, and you have the homework problem. If you have half an ellipsoid, it's completely symmetric in the remaining and minus one dimensions. In one dimension it's got to be positive, but the other one. So that says that the ellipsoid you cover it with has got to actually be symmetric in those dimensions as well. That's a standard thing we've seen in where convex authorization. If the problem has symmetry, the solution must in convex authorization. It has to have the same symmetry.

By the way, that is absolutely false for non-convex problems. So, for convex problems, any symmetry in the problem, you can immediately assume that the solution will also be symmetric under that group of symmetries. Okay? So in this case, since you have a half ball that is sticking a cap like this, sticks like that. Then it's symmetric in all of these other dimensions, and therefore that means that the P matrix is down to one variable or something. I think you have two variables to mess with. By the time you're down to two variables, it's calculus time. It's not a big deal. It's what you learn calculus for. It's the only thing you learn calculus for.

So here is the update. Oh, for N equals one, we do know the minimum volume ellipsoid that covers a half ellipsoid. That's easy. That's basically itself because an ellipsoid is an interval. A half ellipsoid is another interval, and I know the minimum length that covers an interval. So, okay, the minimum volume ellipsoid looks like this. Here's the update. Let me just say a little bit about what the data structure is that we're gonna use to describe. Lot's of ways to describe ellipsoids, right? Lot's of ways, or at least I know three or four. Image of uniball, inverse image of a uniball, quadratic function, and lots of things.

But we'll do it this way. We're going to parameterize it by the center, and a matrix P that's positive definite. In this formulation here, things like the volume is proportional to something like [inaudible] to the, I guess, square root, or something. So the bigger P is – I guess the point is that roughly speaking, the bigger the P is, the bigger the ellipsoid. Did I get that right? Yeah because if P is big, it basically says P-N versus small, and you can go Z minus X can get very big before this thing runs up against a limit of one.

Student: What part of P is vague?

Instructor (Stephen Boyd):Well, yeah. That's vague. What does it mean so – I was being vague. When you say a matrix, a positive of a matrix – P is big. When you pin somebody down, it can mean lots of things. It can mean big in determinant, big in diameter, big in the smallest Eigen Value, so I just mean hand waving. What I'm doing. You can talk about which directions it's big in, right? You can talk about the Eigen. So the Eigen values of P tell you – for example, the condition number P tells you how non-isotropic or an-isotropic that ellipsoid is. It tells you if the condition number is like two. It says in one condition it's twice as big as it is in some other. If it's 10,000, it says something else and so on. All right.

So let's look at it. Here's your – the new ellipsoid looks like this, and it's really cool. I'll give you another interpretation of this in a minute. So the new ellipsoid looks like this. The center – of course, it's very interesting. It's a step in the direction P-G. I'm gonna talk about that. You step not in the direction G – negative G. But in the direction P-G, so in a sort of change of coordinates you step in a different direction. That's this. Step length is one over N plus one, and the new ellipsoid looks like this. And I want to dissect the different parts. Remember, P big in matrix sense roughly corresponds to E big.

So lets talk about the different parts. Let’s forget the scaling, and let’s focus on this. This – by the way, what is this matrix here? What’s the rank of that guy?

Student: One?

Instructor (Stephen Boyd):One, rank one. It’s an outer product. It’s like P-G times the transpose, okay, with a constant in front. This is actually – that’s a rank one down date. In other words, you take a positive definite matrix, and you subtract a rank one thing. I mean, you can’t subtract – this thing is carefully – well, by definition this cannot be positive. This is, again, positive definite. You know what that means? If I ask you about – suppose let’s take the ellipsoid defined by P, and then the ellipsoid defined by that. What can you say about the second ellipsoid? What can you say about all the Eigen values of that matrix compared to the Eigen values of that matrix? You can guess.

Student: They must have gotten smaller.

Instructor (Stephen Boyd):That’s true. They all got smaller. This matrix is less than or equal to P in the matrix sense. It got – it shrunk. Actually, it only shrunk in one direction. It actually shrunk in the direction P-G, or whatever direction this is. It only shrunk in one direction, okay? But it got smaller, therefore this is – that’s the good part. This is the good part where P gets smaller. Then you multiply it by this, and what happens to the ellipsoid? Now what happens to the ellipsoid when this happens? You increase it. So basically, I don’t know if people have seen this in Kalman filtering, but basically what happens is – again, this is only if you’ve had these classes. If you haven’t, then don’t worry about this.

You’d have two steps, right? What happens is first you take a measurement. So you take a measurement, and based on the measurement your estimate of the current state gets better. So your co-variance matrix goes down. It had to. You just got some information, and if you know how to process the information correctly, how could your estimate get worse? So your area co-variance matrix goes down, okay? Then the next that happens in sort of like a Kalman filter is you multiply. You propagate it forward by the dynamics, and add some unknown process noise, and what happens then is that in that step – that’s the dynamics update. Your ignorance increases.

And the hope is when you’re running this whole thing after many steps is that the amount of increases in ignorance – in that case measured by a co-variance matrix, is overwhelmed by the amount of decreases in ignorance that come every time you take a measurement. So that’s what’s happening. You have a vehicle or something you’re tracking. It’s being disturbed by something you don’t know. That’s a process that’s leading to increased ignorance at each step, but you get noisy measurements of it, and that’s a process which if you process those measurements correctly can lead to decreasing ignorance.

This is the same thing. That’s the decrease, and the decrease came from kind of a measurement. If you want to call this a measurement, well you can. That’s a measurement, and then that's your slop, so that's how this works. I can actually – let me give another interpretation of this. Anyway, you're gonna prove this formula, so here's what I'm gonna do. I'm gonna write down an interpretation of that, and once you see this interpretation, you don't need to know anything else. So it's this. Let me see if I can – I just want to interpret the step.

So this is how it goes. You have an ellipsoid like this, and here's your point. And you get a sub-gradient, and let's say the sub-gradient points this direction, okay? That's the sub-gradient. Now suppose you were doing a sub-gradient method. Where would you step in the sub-gradient method? You'd step somewhere along here. Now where depends on your step length on that step, and your step length can be as stupid as one over K, right? So you'd step in this direction. Let's actually see where you step here. Where do you step here? That's the question. I'll just draw it geometrically.

It turns out what you do is you go in the direction – you solve, actually, a problem. This direction that you step in solves this problem. Here's how you get the direction. You minimize G transpose V, where V is a direction subject to V in your ellipsoid. Okay? So you go as far as you can in this direction in this ellipsoid, and in that case – I'm going to try to draw it, and I'll probably get it all wrong, but anyway. It looks something like that, and so it might be here.

So in fact this is the direction – if that's G, that's minus P-G. You know, roughly. I mean, forget the scale factor. This guy down here is minus G, and so what you see is –you can think of it as a distorted – it's like Newton's method. It's exactly like Newton's method. Instead of going in a negative sub-gradient, you're twisting it by a positive definite matrix. Okay? This make sense? And you can check that this is – that P-G is sort of this direction here. And there's a beautiful way to interpret it, and it's this. And it's interpreted in change of co-ordinance.

So change of co-ordinance interpretation goes like this: Here is your ellipsoid at step N, or K, or whatever they call it. It doesn't matter. Here is your ellipsoid at some step, and what this says is – you know, the ellipsoid tells you your ignorance. If it's round, it says you're equally ignorant in all directions, right? If it's little pencil like things then it means that you have very good knowledge of the solution in some directions, and very poor in others, right? So what you do is you say no, and you change co-ordinance. And by the way, you change co-ordinance in the original thing by multiplying by P to the half.

So you use co-ordinate Z equals P to the half times X. You change co-ordinates to make your ignorance round. To make it isotropic so you're equally ignorant in all directions. You change co-ordinates. When you change co-ordinates, you get a transformed G. Let's call that G, I don't know, G – G had been pointing some direction. Well, it doesn't matter. This is your new – I 'm gonna call it G bar. That's G, but transformed by this P to the one-half, or whatever. Okay? Now in this case, the question would be which way to step?

And so the correct step in a situation where your ignorance is isotropic – so you're equally ignorant in all directions. The correct step is to go minus G bar – and it's even funnier than that. It's basically divided by N plus one. This constant step like whatever N plus one. That's it. That's what the ellipsoid method is. The original discussion of ellipsoid method, which was, I guess, by Shore, had this name. In fact, it had a very long name I'm trying to remember because it's really funny when translated into English. It's sub-gradient method with space dilation in the direction of the gradient.

So this is space dilation because when you're changing co-ordinates – and then, by the way, the dilation is only in one direction in each step because when you do just a rank one update, you're actually just scrunching space. Actually, you're expanding it. Well, you shrink the ellipsoid, and that has equivalent in the change of co-ordinates of accentuate of dilating space in the direction of the sub-gradient.

So when you look, if you were to go to Google or to look at some of these early Russian books from the '60s and '70s – some of which are actually in English and are superb books, you'd have a – instead of ellipsoid method, you'd hear ellipsoid method with space dilation in the direction of the – and various other things translated like that. This make sense? The thing you don't do in the sub-gradient method that you do do in ellipsoid is once you step this way, this ellipsoid gets shrunk.

You learn about stuff in this direction so you shrink a little bit, and you re-pull it apart. You do a change of co-ordinates to re-balance or isotropize your ignorance. That's definitely not a word in any language. Make it isotropic. This make sense? So this is the picture. So in some senses what's funny about this is it's a little bit like Newton's method. Because if someone says, "What's Newton's method?" and you can write down some formulas and stuff, and they, "Well, why would anyone want to do that?" and you can blabber on and on.

But if someone says, "Yeah, but what's the idea of Newton's method?" you can say that idea is that the gradient method, although it's the first thing you could think of – it's greedy, and it might look like a good idea to go downhill the fastest way that way, but it might turn out that you should really be going about 45 degrees over that way, or more commonly, 89 degrees that way. And someone saying, "Why?" and that's because you draw a valley and all that. And then you say that Newton's method is basically the gradient method applied after a change of co-ordinates makes the curvature locally isotropic. That's what it is.

So the one case when greedy is good is when your ignorance or function has kind of an equal curvature in all directions, and this is a sub-gradient method with this. In fact, people have a beautiful term for these things. It's called variable metric. The metric tells you about the topology of the space, so you would talk about how Newton's method is a variable metric, gradient method. And in fact, you'd talk about [inaudible] as a variable, metric, sub-gradient method. It has this other interpretation, and so on.

The stopping criterion is pretty simple, and the way that the stopping criterion – let me just draw that over here, and then let me tell you something embarrassing about the stopping criterion. So here is your current ignorance level, and suppose I give you a sub-gradient like that. Well, if you were gonna do another step, you'd cover this with an ellipsoid, and that would be your next point. And actually what you'd do – now you know geometrically what you'd do. You'd solve the LP of going as far as you can in that direction here, and in fact that would give you to this point, right?

So in fact your next step direction, now you know how to construct it, is here. And then the step length is somewhere along here, given by some formula. But you're not gonna do that. You're gonna terminate, and what you do is you stop, you send a signal to this process, and you say, "Done. Time is over. Give me your current point." And they'd say, "Please give me a lower bound on how far you are from optimal."

Well, what you do is this. You – of course you have this. This is just your basic formula for – everything is based on this formula. It's the definition of a sub-gradient. Everything is based on that. So what you do is this. X star is in this ellipsoid. The ellipsoid is a localization region. Therefore, this thing couldn't be – has to be bigger than or equal to the infimum of this linear affine function here over the ellipsoid, like that. It just has to be. You can just analytically work this out. I mean, what the worst thing is. In fact, the worst thing is – in fact, literally, what you're doing is you're solving this formula here.

You're calculating this point, and then when you put that back in – this just has an analytical formula. It's this. So it's absolutely beautiful. Square root of G transpose P-G is actually a guaranteed sub-optimality bound. Couldn't – and it's absolutely beautiful. Do you know what it is in this transformed space? It's this. It's the norm of the sub-gradient in the current co-ordinate system if you change co-ordinates. It kind of just makes beautiful sense, and it's a guaranteed sub-optimality. It makes sense because if someone walks up to you on the street, and you say, "Listen, consider a unit ball." That's kind of your standard ignorance set.

You say, "I have a unit ball. I evaluated the sub-gradient at the origin of a function, and I got this. I know the minimum is somewhere in the unit ball. How far could I be off?" You get this. It's the norm of G. That's how far you are off. Okay, so it's a beautiful stopping criterion, and especially because I think you calculate that as a side calculation anywhere. So you have to calculate this anyway. You calculate it right here. So basically, you calculate this. If it's less than a threshold, you're out. Otherwise, you update.

Okay, and I do have to tell you one thing about this. Well, here's the ellipsoid algorithm. I'll show you the whole algorithm. I have to admit something about it. So you start with an ellipsoid containing X star, you know, typically in the same way an ACCPM might be some box or something. You know, this might typically be a ball of radius capital R. That'd be very common. Okay, here's what you do. You evaluate a gradient. If the gradient in the scaled norm is less than epsilon, you return X – and that's actually certified. There's nothing fuzzy about that at all.

Otherwise you update the ellipsoid like this, and you repeat. That's the ellipsoid method, and now I have to tell you the truth. No, it's not the truth. We're not lying. It is not know, as far as I know, and I spent a week last summer with a bunch of people from Moscow and Kiev in the '60s. They were all there. Enough of them spoke English, so everything was cool. There was a lot of vodka, so that was on the other side, but it's not clear. Anyway, they claim the following: There is no proof that the exit criterion to will actually be satisfied.

So there's no reason to believe it. So the only actual stopping criterion is the one that comes from the complexity theory. Actual means that you can absolutely prove that it will be reached. However, this works invariably in practice. I think we already talked about this, so we already – okay. So here's an example of ellipsoid method. This is our now famous ellipsoid method, and you can kind of look and see how it's working. Each step is ordered in N squared, or something like that. Let's go back and see what the ellipsoid step is. Yeah, it looks to me like it's ordered N squared. Everybody else seeing N squared here for the ellipsoid?

The numerical update, I think, is N squared. The reason it's N squared is – I don't know. Here, you have to write all the entries of P out, so there's N squared right there. And other than that, I don't see anything fancy except I'm seeing [inaudible] vector multiplies. So it's order N squared. This just shows kind of how it works slowly, and so on. This is the lower bound, which is indeed getting better. So I think what we'll do is we'll quit here, and we'll sort of finish this up next time, which will be next Thursday.

[End of Audio]

Duration: 74 minutes