ConvexOptimizationI-Lecture19

Instructor (Stephen Boyd):We'll start in – we'll finish off interior point methods today. So if you go down to the Pad, we'll start, and I mentioned this last time, but I'll – I think I did it a little bit quickly, so we'll do – talk about feasibility in phase one methods.

In these methods, a feasibility method or a phase one method, the job is to figure out whether a set of convex inequalities and equalities is feasible or not. So that's – and this is sort of the very traditional way, where you divide it into a phase one and the phase two.

You've already had a glimpse at methods that don't need that. For example, the infeasible Start-on-Newton method does not require you to start with a feasible point. You only start with a point that is in the domain of the functions, but it doesn't have to satisfy the equality constraints.

Okay, so the feasibility method. The classic one is this, is you introduce a new variable, s, and you minimize s subject to f i of x is less than s. And this can easily be put in the same – in the standard format, except now you can – now you can easily come up with an initial point that is strictly feasible here, because, basically, you take any x here at all, and then you simply take s large enough, and you have a strictly feasible point for this problem.

Now, when you solve this problem, the optimal value is either positive or negative. Well, it could be zero. If it's negative, it means you've actually produced a strictly feasible point. If it's positive, it proves there is no strictly feasible point. If it – if the outcome value is zero, there's no strictly feasible point, but it's tricky at that point. You don't know whether or not there's a feasible point. That's right on the boundary.

Okay. Now this is – in some sense, you can interpret s here, if it's a positive number, you can interpret s as the maximum infeasibility, because that's what it is here. In fact, when you minimize this, and the number is positive, you're going to interpret that number as the minimum possible value of the maximum constraint violation. That's what this s – for example, if s star was .1.

If it's .1, it means that all of your constraints are satisfied within .1. Now, of course, that's sort of pushing the maximum infeasible – constraint violation down and, in that case, you would not be surprised, I would hope, to find out that a whole bunch of inequalities are violated by about .1 if s star is .1. So that's what you get here.

And sure enough, that's exactly the case. So here's an example where this is reversed. This is showing you the margin. So here you have a violation of, I don't know, something like .2, and you can see that 50 variables are slammed right up against whatever that minimum maximum – that minimum value of maximum margin is here, right here. So there – it's just slammed up against it here.

And the number of constraints that are satisfied is something like 39 out of 100 here, and that’s all of these. Everything to the right of zero is an inequality that is satisfied, not violated. So there’s 39 of them.

Now another method is this. If you minimize the sum of the infeasibility – so you introduce – here there was a global scalar infeasibility variable, s, which of course, if one of these is tight, is the maximum violation.

Here each constraint gets its own private violation parameter or – we’ve seen this many, many times before. This is the first line in the support vector machine, for example, and in fact, whenever you have a bunch of inequalities or something like that that you’d like satisfied, but you want it handled gracefully, when in fact they cannot all be satisfied, you would put in – you’d put in a little fudge factor here or something like that that would allow you to satisfy the inequality.

Then the hope would be to drive these to zero. Now when you drive these to zero – these are positive, so one transpose s you should even read as – well, I don’t know. You can write it this way. I mean, it’s true. It’s what it is. And this should fire up a whole section of your brain, not yet identified, which should be devoted to sparse solutions and things like that.

So what you’d expect in such a phase one problem, here, you should expect exactly the following. That a small number of constraints will be violated, assuming it’s not feasible. If it’s feasible, then of course, you ended up with a feasible point, obviously, because you just said s equals zero and you’re done.

But you should – would expect, in this case, to end up with a small number of constraint violations and, in fact, that’s exactly what happens, and here’s what you get. This is for the exact same problem instance over here. You’ve actually satisfied – a whole lot of them are right at zero. We expect that, but over to the right here is about 80 percent of – so actually only 20 are violated here.

Then use – is there no solution that satisfies more than 80? Actually, we don’t know. That’s a hard problem to tell. So – I mean is there – there probably is a – there probably is a point that violates, let’s say, instead of 21, it violates 17 or something like that, but the point is, as a heuristic, this works very, very well to do this.

By the way, you might ask here or you might say that, between these two, you might say, well now wait a minute. Here we introduced one new variable, here. Here we introduce m new variables, and so if you’re trained in old school thinking, which is, as I say, wrong thinking, you would say, wait a minute, that has more variables. This has n plus – this phase one has n plus one variables. This one has n plus m new variables. N plus m is bigger than n plus one, and that’s bad.

Okay, so – to which I'll just say this. I won't go through the details, but I'll say this. If you know what you're doing, that's the operative clause, this problem can be solved with exactly the same complexity as this one, okay?

By the way, it's the same reason that l one regularization, if you add that onto the end of a problem, it look – you introduce all these new variables and you say, my God, look at all those variables flying all around.

It turns out, if you know what you're doing, which is to say, you do smart linear algebra in computing the search direction, complexity is identical. There's no cost. No additional cost.

So here you can even guess why. Let's talk about sparsity in here, and then I'll quit on this topic, but let's talk about sparsity. S three. That's a variable. Please tell me where s three appears in the KKT system? Does it appear in the equalities? No.

Okay, let's see. S three appears where? It's coupled with just f three here, so that tells you something about the sparsity pattern here, and you'll find out, if you do this correctly, you will indeed get the KKT majors when you write it out here, it's going to look much bigger, but if you eliminate the s row, the s' first, you'll get the – I should say the ds', because you're actually calculating a step, you'll find out that the complexity is the same as this one.

Okay. Now another thing that's kind of obvious is, if you're solving a phase one problem, the amount of time it takes – it's not unfair to say that, basically, if someone asks you, how long does it take an interior point method to solve a problem? The basic answer is like 30 steps.

Someone says, really? All the time? And if they're looking nervous, you say, okay, fine, it can be 60. And if they look really nervous, then you say, fine, it can be 80, but don't go – you don't need to go about 80.

If it's an LP or a QP or something like that, you can say 25. These are just the numbers you say, and someone says, really, it doesn't matter like if it's image processing or finance or a machine? And you go, no, it's just 25, roughly.

Now, for feasibility, that's also the correct zerothor to answer. S if someone says, how long does it take to detect whether a set of matrix inequalities or whatever is feasible, the answer is 25 steps, right? And you're safe. So it could be 50 or something like that. But if you think carefully about it, it cannot be uniform – that cannot uniformly be the case, because, as you get problems where the feasible set is super tiny, there has to be something in the complexity that relates to that.

And, indeed, that is the case empirically. It's also the case theoretically, so if you look very, very carefully at sort of the complexity and analysis of interior point methods, hidden deep in the wonderful results and everything, there'll be a little constant, and you'll say, well what's that constant?

And then the person explaining it will say, don't worry about that. The constant turns out to be something related to how close to infeasible the problem is, right? Of course, if you think carefully about that, entirely violates the whole contract of complexity theory, because complexity theory says, how many steps does it take to solve this in the worst case? And you can't slip – you're not supposed to slip in a number that depends on the data, because once you slip in a number that depends on the data, then I can make anything I like. I could say, I can solve it in log n steps, and they'd say, really? And you go, yes, no problem.

You just put a little constant in front of it, and you'd say, what's that constant? And you go, it's just a constant. They go, what does it depend on? You'd say, the problem data. And the constant would be the following. It would be the actual number of steps required to solve it divided by log n. Everybody following this?

So the whole thing is it's a bit fishy. Now they would protest this vigorously and say that they can something about that constant, but the truth is, these complexity analyses are not as pure as you might imagine they are at first glance.

Okay. So there is something that involves how close to infeasible a problem is and how many steps it takes to solve it. I mean, both – and I'm talking both in theory and in practice. Now, in practice, though, it's quite tricky.

Here's a picture showing where you take a parameter, you take a set of inequalities, and you change this parameter, gamma, where, at some point, if you crank gamma down small enough – well, if delta b is positive, if I make gamma small enough here, then this will be infeasible. If gamma is large enough, this will be feasible. We can adjust the boundary to be zero. We'll make delta b positive, okay?

So delta b is positive. I adjust gamma so that this set of inequalities is feasible but not strictly feasible, okay? That set. This is strictly feasible for gamma positive and infeasible for not. Now what happens is this. As gamma varies, you get numbers of steps like this, and you can see that, basically, up to very close to this point where the set goes from infeasible – empty to – strictly feasible to infeasible or whatever, you can see that the number of steps, it grows slightly.

By the way, this is just a sloppy method. These numbers can actually be much smaller here. And you can see that, as you get very close, you could look at the numbers here. They're very, very close. You can see that it actually grows like this.

Couldn't even find a case where it grows faster than this, because then you get into very small numerical things. You have things where it's feasible, but with something like only in the sixth or eighth digit, or something like that, at which point it seems not to be a practical question anyway, at that point.

So this is just – this is this thing blown up on a different scale. By the way, I should ask you something, here, about this. What do you think the – an infeasible Start-on-Newton method looks like for this?

Well, it'll never tell you – it will never tell you infeasibility. It can be adjusted to, but let's skip that. So let's very gamma, and this is the strictly feasible range, and let's do infeasible Start-on-Newton, right?

Which simply applies Newton's method to the problem. Infeasible Start-on-Newton. And the minute is takes a full step, it's feasible, okay? And then we stop. Then we terminate.

So the question is, what do you imagine this looks like? This would be the number of steps required in an infeasible Start-on-Newton method.

Well let's think of it this way. Let's start with a problem that is like super duper – super feasible, over here. It's like the huge, huge, vast feasible set. How long do you think it'll take in this case? It's just a – it'll be just a couple of steps, right?

So it's going to look – it's going to be like this. Then this is gonna grow much more strongly, I think, than in this case. It's gonna look like that. So if you get down here, this is already things like – this could be 1,000 steps or something like that. So it will grow – this curve will look quite different from this phase one. The phase one will actually look like this, right? And just settle out at 20 steps. I guess this goes lower than 20 steps, like that. It'll look something like this. Okay.

Not that this matters, but just so you know. Okay. Now we'll look at the complexity analysis of these methods. And this is something like, I guess, some people would call this the crowning achievement of the '90s or something like – I don't know. There are probably people who feel that way. I mean I think it's very cool.

What effect it has on practice is not entirely clear. I'll say something about that when we – at various times, but first let's look at it. So let's go back to the whole idea of the barrier method, right? So the whole idea of the barrier method is something like this.

You minimize t – let's just take a linear objective. C transpose x plus this barrier – you minimize this for some – for one value of t, and then you crank up t higher and do it again. So this is what we're – I guess this is what we're doing.

Now, you can show all sorts of things. For example, you can show that the Hessian, here, of this thing, is actually going to be – it's actually becoming singular, okay? So if t gets big enough, the Hessian here is – the condition number is going to infinity.

By the way, that should set up all sorts of alarms right there, okay? And in fact, what you wouldn't be surprised of and you should not be surprised is, if when you solve this, it worked something like this. For some value of t, like t equals one, this took eight Newton steps. Then you said t equals t, and it takes like 12 Newton steps. Then t equals four, and it takes 20 Newton steps. And the number of Newton steps is growing. And someone will say, wow, your – the number of work per centering is growing. Why?

And you'd say, well, look, come on. It's getting a – you're getting a harder problem to solve. The higher t is – I guess maybe I should do it this way. The higher t is, the more this thing looks like – you remember this picture. Looks something like this, right? It starts looking like that, and it's got more – it's got a higher third derivative, which, indeed, you can check. As t gets small, this thing has a higher third derivative. Maximum third derivative. Higher Liepschischzt Gaussian on the Hessian.

So you could say, look, it's completely consistent with everything we know that, as you crank t up, the problem – these problems become harder to solve. It takes more steps.

Now, in fact – in theory – sorry, in practice, it is observed that this does not hold and, in fact, you should have observed it by now. Actually, how many people have finished their homework? Okay, so they wouldn't check out like numbers of Newton steps as you crank t up. Did it grow?

By the way, if you make t super-huge, it's going to grow, but for another reason. But what are the numbers of centering steps, typically? Five? 10? Did it grow as t got bigger?

Student: Yes.

Instructor (Stephen Boyd):How much.

Student: It went up to 10.

Instructor (Stephen Boyd):It went from five to 10. Okay. Well I'm not – I'm talking about catastrophic. It doesn't grow catastrophically, or you didn't push it that hard. Okay. Well, okay. So – all right, so it's observed, basically, here, that what – that, here, when you increase t by some factor, the number of Newton steps doesn't increase.

So – I mean that's good. It's an empirical – it's a great thing. It's sort of just an empirical observation and, in fact, that's the key to it. By the way, if you'd used another barrier, it could well have. So if you wanna know why log barrier and all that, and why – this is one of the reasons.

What we're gonna do now is actually see if the same thing is predicted by the theory. The numbers are totally different and, I think, fairly useless, the numbers, but nevertheless, it's always nice to have the theory agree with the practice, at least qualitatively if not in numbers.

So let's see how that works. We're going to assume the following. That f – the easiest assumption is f zero and phi are self-concordant, okay?

So – there's at least one case where you actually have to look at the whole thing, the composite. That's the entropy maximization. But, in this case, we'll just look at t. We'll assume f zero and phi are self-concordant.

Now, what that means is this. If t is bigger than or equal t one, t f zero is self-concordant and, therefore, so is t f zero plus phi. That's self-concordant. Okay. And then it's going to include a lot of problems.

Now we can actually say what the Newton iterations per centering step is, right? And remember what this is. This is five, which is my symbol for log log one over epsilon, okay? So that's what this is. It's six, whatever. Something like that.

And the number of steps is this. It's equal to your initial starting point, and we're minimizing, of course – well, let's see. Where are – yes, we're minimizing mu t f zero plus phi, this thing, starting from the solution of the problem for t f zero plus phi, because in one update we take t star equals mu, and that's what we do. So we multiply that by mu.

Okay, so this is the new f minus f star. F star is gonna be this thing here. Divided by gamma. Gamma is some absolute constant. I can't remember what it was, but I think it was one over 345 or something, if we do our analysis. I think, if you work very, very hard, you can make this one over .09 or something like that. Okay.

Now if you do this, this all very nice, but there's one minor problem, which is this. This is a problem, generally, in the self-concordant analysis complexity of Newton's method. If someone says, how many steps does it take? It's f minus f star divided by gamma plus c. And someone says, but what's f star? Or how would you find f star? You'd find f star by running Newton's method, but once you've run Newton's method, you don't need bound on the number of Newton steps, because now you know the exact number of Newton steps required. Everyone see what I'm saying?

So one could easy – you could easily imagine that this is not too useful. However, as usual, you wanna lower bound a convex function, what's going to come to the rescue is duality.

So what we'll do is this. This thing here, that's this top thing, that's equal to, and you can write it out this way, and that's less than or equal to, and I'm not going to go through the details or whatever, but this is what you get. This is mu t – actually, here you can argue it – it is duality, but you can argue it just completely directly.

I think we're using the fact that we're bounding this thing – there's a simple inequality, and actually I'm not gonna go through the details here. Actually, it just bounds log mu and one and mu squared over two and all that kind of stuff, but I'm not gonna go through it.

So what happens is you bound this thing by – I guess, when all the smoke clears, you get this, and this is an upper bound. So this is the – this is f of the starting part. That was the previous – that was f – it's t f of x plus mu t f of x plus phi of x minus phi of x plus – minus the same thing for x plus. You get this thing, and it's less than this. And then you stop here. That's the dual function. And you end up with a very, very simple thing, which is m times mu minus one minus log mu.

If you plug this in here, you find out that the number of Newton steps is less than this thing right here. So that's it. I guess you could do this if you wanted. Okay.

All right. It's a very, very simple thing, this thing. It just says that the number of Newton steps – and, actually, the interesting part is this. It depends only on mu. Mu is your multiplicative update factor in your parameter here. So if mu is 10 or whatever, it simple says there's a number of iterations, of Newton steps, that you can bound, that there's a bound on it, and it will never take more than that.

So this whole idea of the problem getting harder and harder as you crank t up, now the theory tells you that's false. The upper bound does not grow. It depends on nothing, notice here. Depends – has nothing to do with the dimension of the problem. It has nothing to do – it has to do with the number of inequalities, and mu, which is how aggressively you update. There's a question?

Student: What is the m again?

Instructor (Stephen Boyd):M is the number of inequalities. Inequality constraints. N is the number of variables. So it depends on absolutely nothing else. C and gamma are absolute constants. This – I think this one's one over 345 or – and this one's like five or six or something like that. They're just absolute constants, right?

So that's nice, and actually it's kind of cool. If you look at this, you will see that it is in fact – did we plot it? No, we didn't plot this, but this is an increasing function of mu, and the mu is small. And, in fact, this thing looks like the following. Near zero, it looks like mu and then plus one half of mu squared or something like that – sorry. In fact, it even cancels the log, because the expansion of this is log mu is about one minus – mu minus one plus – then the next term is a minus one-half mu – so this thing here looks like one half mu squared for mu small.

And that says – actually, that makes perfect sense. I wish I had plotted this. It looks like this, this function. It looks like that. The – so what it says is, if you take mu small, the bound is extremely small, because this is sort of like small squared. Of course, then there's the c, so the extremely small hardly matters, right?

So if mu is chosen small enough – this thing is small. As mu goes to infinity, this thing grows, and it grows kind of like mu. Linearly. That's the upper bound. Okay?

So this thing is growing with mu, and that makes perfect sense. That's the number of – bound on the number of Newton steps required per centering step. Mu is the – is how – is the parameter that sets how aggressively you update t, so if mu is large, this thing should be large, because you are aggressively updating your parameter and should expect to spend more Newton steps on it.

This thing, in contrast, is decreasing with mu. Here, if mu increases, this thing goes down. Why? Well this is simple. This just tells you it’s the number – this is, basically, mu is literally the duality gap reduction you get per centering step, and so the number of times you’ll have to do that is just this number. It’s very simple.

You multiply these two together, and you actually the total – a bound on the total number of Newton steps to solve the problem, and sure enough, here’s what it looks like. And this is for some typical values of gamma and c. M equals 100, so 100 inequalities, and we’re going to use something like a 10 to the five reduction in duality gap here. Actually, sorry, that’s the absolute gap is gonna be 10 to the minus five.

So what happens is it looks something like this. And, sure enough, it’s kind of cool. It shows you that, for mu small, what happens is this is really small, but the constant is – this is basically negligible, and that’s a constant, and then this thing gets big.

On the other hand, for mu large, this gets big even though this goes down, and this goes down like a log sort of a thing, and this goes up linearly, so that’s not a winning combination to make mu large. And, sure enough, you plot this and you get this. You get something like that.

What’s wrong here, of course, are the numbers. So what this says is that the – this would tell you that the optimal value of mu, for this problem, is 1.03 – I don’t know, two? It looks like 1.02 to me. And it says that the maximum number of steps is three – let’s see, I don’t know, 8,000. Everyone get approximately that?

So the numbers are wrong, but it says, if you’re a complexity theorist, it says, the best thing to do here, by far – I mean, for a complexity theorist, this is that mu equals 1.02. That’s a real homotopy method. That’s a 2 percent increase in t, and then, presumably, at that point, you’d do some number of iterations.

And the total number of iterations it will take is no more than 8,000, okay? So that’s great. Actually, if you put – you can put mu as a function of m and we’ll actually get a nice complexity bound. Of course, this is off. You know what good values of mu in practice range from like two to 50 or something like that, for all of these problems, and the number of steps here has a minimum value that’s around 25, 30, something like that, and it does – it’s not sharp either.

So, in practice, well it’s on a totally different scale, right? On a totally different scale. Well, you’ve seen what it looks like. We had it earlier, so you can see what it – so this is sort of the theory scale, and here’s the practice scale.

And, in fact, what’s not shown here is that this would actually go up eventually, like that. So this is the practice – this is, in practice, the number of steps required, as a function of how aggressively you update, looks like that, and the theory scale looks like this. And everything is different about them. I mean, they have the same qualitative factors. Roughly qualitative, right? This settles out at about 30 steps, but any mu, from whatever it is, five or two to a couple of hundred, it’s gonna work pretty well here.

And here, this makes you think there’s a sharp minimum. There’s not. I mean, obviously. Okay. Now if you plug in, here, mu is one plus one over a square root m, and work out what happens and do some bounding, it’s not too hard, but what you get is this. That the total number – this whole number, right here, if you plug that in, can be bounded by – it’s a constant times squared m log of this thing.

Now, by the way, this ratio is exactly – is basic – well, it can be interpreted this way. This is your ignorance reduction factor, is what it is, right? Because, once you’ve centered with t zero, your estimate of the duality gap is m over – your duality gap is m over t zero. That’s your initial duality gap. Your duality gap is a measure of how ignorant you are as to what the optimal value is.

This thing runs until that ignorance is pushed below the level epsilon, so this quantity is exactly interpreted as a – as an ignorance reduction ratio.

Log of that, of course, tells you the number of bits of information you have learned. Which is to say – yes, that’s what it means. If there were log two, this would literally be the number of bits of information you learned about it, because if someone, at the beginning, says, please tells me about the optimal value of the problem, you’d say, well, it’s between my current value, my current f zero of x, here, and, and then you’d have to draw an interval whose width was equal to – and whose width is this, that’s the current duality gap, there. And you’d say, where’s the optimal value in here? And you’d have to say, I don’t know. So that’s the width of your ignorance.

When you terminate – well, of course it’s gonna have to be somewhere in here. It’s gonna look like – you’re gonna terminate this way. That’s gonna be your suboptimal point, and the width of this will be less than epsilon, so the ratio of these is the reduction factor. And if this were log two, it would be something like the number of steps – the number of, like, by section steps it would take to get there. Okay. So that’s what this is.

And here’s the interesting part. It grows like square root m. So these are – this is, by the way, the best complexity estimate people have, to date, on these things. No one has done any better.

So if anyone – if someone asks you, again, what’s the – how many steps is an interior point method, you look at them, you look at their books, you kind of figure out what kind of a person they are. If they wanna know how well does it work in practice, what do you say? How many steps?

Student:30.

Instructor (Stephen Boyd):25, 30. And you look at them a little bit. If they're a little on the nervous side, you say 50, right? No, no. Because you want it to always be less than that, otherwise, you say 25, it's 40, and they go like, you lied to me. So you look at their personality. You could say 20 to 80 if you want.

If they're a complexity theorist, what do you say? You say square root m is the answer, right? And they say, doesn't it depend on the accuracy? And you go, yes, it does, but weakly, by a log here. Right?

By the way, this number here, if you want, you could just stick that in with the o, as far as I'm concerned, because basically you could just take – you could say, look, I'll reduce the ignorance by 10 to the eighth. 10 to the tenth. I mean, that's fine. Something like that is gonna be just fine, and this will be a very small number then. 25 or something. I don't know, whatever it is.

Okay. All right. So that's the final story. Let me ask you a couple of questions about it. N, the number of variables, doesn't come in. Why not? I mean that's the first thing you see here, is it has absolutely nothing to do with the dimension of the problem, in terms of the number of variables. Where does n comes in?

Student: It comes in in computing the Hessian.

Instructor (Stephen Boyd):Absolutely, right. So this is the number of Newton steps, and it's actually really cool that you could have this separation, where the number of Newton steps, your bound, is completely independent of n. It can be 10 variables, 10 million variables. Same complexity estimate. And so it doesn't depend on n at all.

N comes in, you multiply this to get the total number of flops or computational effort or whatever you wanna call it. You multiply this times the number, the number of flops or whatever that your computation measure is, per Newton step. Everybody got that? So that's the total effort. So no one is saying it takes the same amount of time to solve a 10 million variable problem as a 10 variable problem.

What this is saying – actually, where the complexity theorists and the people who do this in practice agree is that, in both cases, it takes a number of steps. Both of them agree that that number of steps has nothing – that you get the same number for 10 variables and 10 million. The practical person will just say 30 in both cases, and the complexity theorist will say square root m in both cases, and it will have nothing to do with n.

So let's do a couple of – we can do a – well, in the simplest case it's gonna grow something like n cubed. That's a very rough number, but n cubed is a good number, because if you just blindly form the Hessian or the KKT system and solve it or something like that.

If you wanted a really – an upper bound, I guess you could do something like this. Just solving the KKT system by your favorite method would be this.

So if you really wanted the full thing, you'd multiply this by – this is something – you'd have something like this, and then you'd write n plus m cubed. Okay? So this would be the full case. That would be the – and by the way, people would then say, if you look in the literature, if you go to Google and type complexity solving, SOCP or LP or something like that, you'll find lots of papers that have an m to the 3.5, or they may call it n, because they reverse the m and n, but anyway. You'll find a complexity – a 3.5 complexity algorithm or something like that. Okay?

Just a couple of comments here, just – I should mention. The people who – everyone's fooling around trying to make a method over here, where this thing goes below square root m or something like that, where the complexity goes below square root m, and people argue about methods where this is square root m and it's other thing – that kind of thing over here.

However, between these two terms, considering what we've been talking about the last three weeks, which do you think is the one where the greatest opportunities for making your method faster appear?

I'll give you a hint. It's not this term. Okay. It's here, obviously. So, in fact, if you wanted to work on something, don't sit down and try to figure out some new self-concordant, exotic barrier, blah, blah. This kind of thing, where you can reduce the theta factor. I mean, unless this is the kind of thing you like to do. That's fine. Just don't imagine it has anything to do with solving problems faster, anything, or large ones, or anything like that.

The real – all your efforts should go into this, and this is just a baseline, stupid KKT matrix backslash whatever. I guess, in the simplest case, minus g zero. That's what this is. So everything goes in here. This can be brought down hugely by doing intelligent, linear algebra, by exploiting the structure. All right. So that finishes that up.

Let's look at how this works with generalized inequalities. So for generalized inequalities, that's a problem where you have a conic inequality. You have vector inequalities, and these are with respect to a cone.

Now the ones that we're actually gonna really – the only ones that really matter to us, in fact, are semi-definite programming and second order cone program. So these are the only two – the only two cones that really matter. These are the – the Laurents are second order cones, and semi-definite programming.

By the way, there are ways to write semi-definite programming in a classical form, but they're not – they seem to just make things harder. It can be done, however. You want me to show you how? I'll just show you. It doesn't matter, but you'd find papers written on this.

If you have an inequality like, let's say, a – a of x is some affine function, and you have an inequality that, for example, says this, that that's positive semi-definite, right? So this would be a – this would be an STP. What you'd do is this, is you'd take LII of a of x. This is the ith element on the diagonal of the Cholesky factor of a. It can be shown that that's concave.

So I can write – actually, not only is it concave, but it's – I can write it this way. I can write this inequality in a classical way this way, and I can apply a barrier method to this, just like you've done. Of course, now you have to find out the Hessian and everything of the – the Hessian of the ith element of the Cholesky factor of a positive definite matrix. It could be done, but you will get a headache. And actually, it turns out these methods turn out to be what we're talking about now anyway. Okay.

But let me go back to this. Okay. So – and we'll assume everything holds like it's strictly feasible and Slater's condition holds is a zero duality gap and all that kind of stuff.

So, to generalize this, what we need to do is the following. We need, since we have a generalized notion of positivity, namely it's with respect to a cone, we need to have something like a log barrier.

To make a log barrier, you need to have a logarithm on this cone. Now, if the cone is, for example, the positive semi-definite cone or positive definite cone, you need something that acts like a logarithm, and so these are called generalized logarithms, and it's this.

It's gonna have – it's gonna generalize any of the logarithm, so it's gonna have a domain which is the interior of the cone, the same way the log has a domain which is the interior of r sub-plus, which is r sub-plus plus, which is the positive ray.

Okay, so the domain here is gonna be the interior of the cone. It's got to be concave, like log is, and – so it's gotta be concave, so it's gotta be concave, and it has to have the following property. If you look at on a ray – so if you look at – if you examine it on a ray, so you multiply by y by s, it should look like this. It should look – the value of that point, it should grow like the log on any ray.

So if you go on a ray, it should actually go like the log on s, like this. Okay? This number, theta, by the way, is called the degree of the – is the degree of the generalized logarithm.

So an example would be something like this. If you have the non-negative orthant – the normal log is a generalized log with theta parameter one, for example. It's kind of obvious.

Non-negative orthant, if you take the sum of the log it says we had before, the degree is just n. Now, the interesting thing is what plays the role of the logarithm on positive – a positive semi-definite cone?

And the answer is log determinant. I mean it was – it could hardly have been otherwise, right? So log determinant plays the role. And for the second order cone, the logarithm is the log of the last entry minus the sum of the squares of the others.

Of course, this is on the point where the sum of these things is strictly less than or equal to y n plus one squared, okay? So this is the picture, and it's on this – this is defined on this set.

Actually, this is defined – this formula is defined – the formula, at least, makes sense when y n plus one squared is negative, but the domain of this is restricted to this, okay?

So that's – these are the logarithms, is log det. Okay. Now, you can show a bunch of things here. I'm not gonna do this, but we'll – I'll just say what they are. The first one is – let's see. The gradient of a generalized logarithm is positive with respect to the cone, so the – for the ordinary logarithm, the gradient is one over x or one over y, and that's obviously positive, but this is true in general.

So the gradient of log det – you should remember what that is. The gradient of log det is the inverse of the matrix, okay? So I think you've seen that somewhere, but the gradient of log det is the inverse of the matrix and, indeed, the inverse of a positive definite matrix is positive semi-definite. It's also positive definite.

The other one is this one, which is very interesting. It says that y transpose times the gradient should equal theta, and we should actually check that for log det. For log det, what is the gradient here? It's actually y inverse, right? And y is a matrix, and how do you form the inner product of two matrices?

I could put a big black dot in the middle, but that's kind of just a notational – what, in fact – how do you – what's the inner – what's the standard product?

Student:[Inaudible].

Instructor (Stephen Boyd):Yes. It's trace of this. You can put the transpose there or not, but these are symmetric, so I elect not to put the transpose. Anyway, it wouldn't make any difference. And what is it?

Student:[Inaudible].

Instructor (Stephen Boyd):What's that?

Student:[Inaudible].

Instructor (Stephen Boyd):What's trace y y inverse?

Student: Sigma y – sigma m [inaudible] so m.

Instructor (Stephen Boyd):M, thank you. Good, thank you. Okay. So that's m. Great. And which is, indeed, the order of the logarithm, okay? So I'm just saying, these look fancy, these look like fancy formulas, but in fact we're only interested in a handful of them, so it's kind of silly.

And for there, where it's completely trivial to verify these things. Okay. All right. So this is fine, and this all just works. By the way, the second order cone has a theta value of two. If I remember, I'll give you a – I can tell you what the second order cone – what – there's another meaning to theta, a rough meaning, geometric meaning. I'll get to it in a minute.

Okay. Now we're just gonna generalize everything from the interior point method. Just absolutely everything. So will log barrier? Same thing. It just looks like this. It's – I form five xs. Instead of sum, minus sum log of the margins – negative margin – well, margins, I form – I just replace ordinary log with this kind of vector log or whatever – however you wanna call this. A generalized logarithm.

I do this, and this function is gonna be – it's gonna be convex. It's gonna – that follows from the standard composition rules. This is in a more complicated case, because this is convex. This thing is concave, with respect to – minus f is concave with respect to this generalized cone k i, and that's k i convex, and so on.

Okay. And then the same – you can define the central path, which looks like this, and this looks very much like what we had before, where this was the sum of the – minus the – it's the sum of minus log minus f i.

Actually, it doesn't look very much like it. It's identical, this thing. So that's good. Okay. Now you start asking questions. You can define the central path, you could – then you look at the original one, and you say, look, on the central path you can actually find dual points. Let's see if we can – if that works here too.

If you write out the gradient of this, with respect to x, you have to write out – the gradient of that is easy. The gradient of that is a pain, but you can get it from this thing, and the gradient of that is equal to this, and that's equal to zero.

Now, the argument here is absolutely identical. What happens is you take lambda i is this thing. That's exactly the same as what we had before, except now it's in the form of a general logarithm whereas before it was a specific logarithm.

All of this will – is the same, and in fact, you find out, then, that if you have centered x – in other words, if you've computed a point on the central path then, whether you like it or not, you have actually constructed dual feasible points.

So you'll get a dual feasible point. There's a formula for the dual feasible point. It looks like that. And, sure enough, if you work out what the gap, what the duality gap is, associated with this dual feasible point, you will get that – remember what it is in the general – in the specific case, the scalar case. It's exactly m over t. It's the number of inequalities divided by this parameter t.

Here it's gonna be the same thing. It's sum theta i one over t. By the way, this generalizes perfectly. If each inequality is scalar, theta is one, and when you add these up, you get m over t, so you get the same thing. So everything just works beautifully.

By the way, this has lots of implications. This means that the code you just wrote for linear programming, or will have just written by the end – well, actually, I don't know that I announced that, but everyone got this email. If you want, you can turn in your homework like tomorrow or something. I guess that went out, didn't it? It went out. Yes.

So – but what it means is that it would take you about five minutes to change that code to be a semi-definite programming solver. Then it's kind of cool, because you'll actually have a solver for something that, 15 years ago, was considered highly nontrivial, and 20 years ago, would have been not recognized as an easily solved problem, and you'll have written one. It'll be 50 lines, with a lot of comments, and it'll work really, really well. So the same thing you wrote would work there.

Okay. So let's look at semi-definite programming. Let's see how this works. Here's the problem. You're gonna minimize c transpose x subject to this LMI, linear matrix inequality. It looks like that. Now, the log barrier is log det minus f inverse, here. The central path minimizes this, and if you take the derivative of this with respect to x i, you'd get the following.

Here you'd get t c i. That's the component here. And when you take the derivative of this, with respect to x i, you get the following. You get minus f of x inverse times, and then f – since f of x is this thing, you get f i, and you get a trace in there, like that, and I elect the – no, that's what you get. You just get just that.

So you get this thing, and then this has to vanish. If this vanishes, here, then it's the same at that. It gives you this. Then you'd wanna get a dual point, somehow, from this, and it turns out it's nothing more than minus one over t f of x x star of t inverse. That's gonna be feasible for the dual. This is the dual of this – the dual STP. And the duality gap on the central path is exactly p over t. P is the size of the matrices involved here, so f and z.

So that's it. The barrier method is very straightforward. In fact, let's look at it. Here's the barrier method. You start with a strictly feasible x, some given value of t, mu positive, some tolerance, some tolerance, and you'd do a centering step, and you update, and you do a stopping criterion, and then you t times equals mu if you want. Okay?

Now, if you look at this, you'll find that it's identical, absolutely identical, to the barrier method in the ordinary case, with one change. This had been m and is now sum theta i of t. Okay?

So that means, actually, if you wrote your solver nicely enough, it means it solve – you can write a general cone solver, and if you wanted to, you could write something that is like STP T3 or SeDuMi or whichever of the cone solvers you choose to use or whatever, for example, when you use CBX. So you could write one, and it wouldn't be long.

It wouldn't work as well as the real thing, which has thousands of person years – person hours, not years. Sorry. Person years of development effort into it, but it wouldn't be bad – it would be shockingly – you'd be shocked at how well it would work, actually, if you made your own cone solver.

Just from the LP solver you just wrote. It would work not badly at all. Okay. Everything else is the same, so I'm not gonna go through the detail. I mean, the complexity analysis is the same. You just – this just becomes m, and nothing changes.

Then you might ask, well how does it work? Well, here's the way it works. Here's a second order cone program here with 50 variables, 50 SOC constraints in R6, right? So who knows, whatever. Actually, I think this might be to make it parallel to – no, it's even bigger. It doesn't matter. The plots always look exactly the same.

And here's what happens. Here's the duality gap, and you can see it's the same thing. In fact, it looks shockingly – it almost looks like we're reusing figures here, but – because it looks too close to – like that, okay? So there's a GP, and here's an STP. Okay. And you can see things like this that, depending on what the mu update is, it can take on the order of what, 30 steps or something like that. That's the picture.

Here's what happens as you vary mu. And, indeed, we didn't do this long enough, but this would have gone up at some point, like that. Here's an STP, and it looks the same. So, again, it's like 30 steps if you optimally choose – if you choose mu well or something like that. It's the same story.

Actually, it's a good – this is a good time to stop and actually think about what the meaning of these two is. You have to stop and understand, these are problems that, 20 years ago, would not have been recognized as even possible to solve, in some sense. Right? These would be – these were very complicated problems. If you absolutely had to solve one, there were exotic methods. They would take a very long time, but it was considered difficult to do.

I promise you, although we won't force you to do it, I promise you that your LP code would be adapted with a very small number of line changes, and you would see things like this, and you would then be – you would have written, from scratch, a solver that solves something that, 20 years ago, was not even recognized as an easily solved problem.

It's actually pretty impressive. So these look very simple. I mean, it's not a big deal or anything like that. Now it looks simple. It was not considered simple 20 years ago.

And there’s, by the way, still plenty of fields where people don’t know about second order cones or matrix inequalities.

Student: So where’s the side on the mu?

Instructor (Stephen Boyd):On the generic ones? You mean like, for example, SeDuMi or STP T3? I’m gonna tell – I’ll say a little bit about how they do this. They don’t – in fact, it’s not exactly this algorithm, right? So I’ve shown you the simplest possible barrier method.

If you actually – what they’re actually doing something much more complicated, which is homogenous primal duals, but in fact, if you go through it, I promise, if you look at – if you read any of these papers or anything like that, or actually look at the code, it’s all open source, you can – there’s no mystery in there.

If you look at it, you will find all the things you’ve been seeing. You’ll see the KKT matrix, and things like that. It might even – some of it – it’ll be a little bit more complicated, and there is something effectively equivalent to their choice of mu.

For linear programming, it’s done by something called the Merotra Predictor Corrector, something or others. They actually calculate two things, and it’s this – then there’s a magic formula that’s proved to work unbelievably well, so – across a huge variety of different problems, right? So that’s how it’s done.

I think someone else asked about that at one point. They actually kept repeatedly asking what – how do the high-end things update mu. And I said adaptively. It’s using that method. So they’ll look – you will not find, actually, if you go and look for a barrier – if you’d like barrier method, something like that, you’ll only find things that, basically, are in my website or something like that.

So you won’t find a production – a fancy production code that uses a simple barrier method. In fact, if I go and tell people, this is what we use, or something, they’re very – they’re like, that’s so early ‘90s or I don’t know. I mean – so they’re snobby about it, because they’re moved on to the super-fancy homogenous embedding primal dual blah, blah, blah methods.

Which, by the way, guess how many steps they take? Kind of the same number, like 20 or whatever. So it’s funny. They’ll say, well yes, but our methods take 20 and yours take 35, but do you remember that second term? The second – that’s – the second term is related to how smart you are in linear algebra. Of course, that makes the huge difference, right?

Okay. Here’s a family of STPs, just generating STPs of different sizes. By the way, this thing, I think it’s added here just to show you that we won’t lie. That shouldn’t be there. I have no idea what that is. It should not be there. So – but you’re seeing it. It should kind of look like this. It should settle out at like 30 or something like this, and there should be – and the variant should not be high. Here.

So this is just a picture of what – how the STP complexity grows. And remember what a complexity theorist says. A complexity theorist says, this grows like the square root. Like that. That's the bound.

We will finish up. We'll answer your question. And this is – I'll just make some vague comments about it. I claim, if you fully understand the barrier method, which it seems to me you should, because you will have looked at it and you will have implemented one, at which point, it seems to me, you understand it, assuming it works. You're actually ready to look at – I mean, if you care to, you're ready to look at sort of the world of advanced methods.

So the real advanced methods, the ones that are currently in fashion are primal dual interior point methods. These are described in the book. You can look at them. They're actually even shorter. They don't have inner and outer – they don't have inner and outer – you don't have centering steps and then update t. It's just, literally, one iteration, and the idea is pretty simple. They just – it's basically the same as updating t at each iteration instead of what we're doing.

What we're doing, in a barrier method, is something like this. Is we're starting here. We set a target point, we don't know where it is, and now we Newton for a while and end up over here. Then, once we're there, we reset our target. We times equal – we t times equals 10, and we go on a new Newton adventure and land here, and that's what we're doing.

Now it's kind of obvious that those last four Newton – those last couple of Newton steps here are a complete waste of time, because we're now polishing off, we're getting accuracy in the sixth through twelfth digits of the centering point, which has nothing to do with what we really wanna do.

What we really wanna do is calculate this point. So it's completely obvious that this – that our method – what's shocking is that it still works, is that you could shave like a whole bunch of iterations off right away, and you could do this by incomplete centering, for example.

In other words, you stop when you get close here, or another way to do it is actually to increment t as you go. And, in fact, if you read fancy methods about it, which you are now, I claim, fully prepared to do if you're so inclined, you can read – you can read about all the fancy methods, and all the fancy methods will have a picture that looks like this. If they don't, they should. Actually, they don't, but some will. Okay?

So that's the central path, and the goal – this is – you'll have some measure of being off centrality, and they'll have all sorts of measures. You can look at these things. Again, if you want to, I claim you will see everything you've seen. All the log barriers, the gradient, the Hessians, everything, the KKT systems. It'll all be there, and they'll – this, they call a neighborhood of the central path, and the idea, now, is to just sort of make steps that go like that.

And so the idea is you stay in the – in the neighborhood of the central path. So that's what this looks like. Something like that. And so you'll see these things in the fancier methods, and the primal dual ones as well. They sound fancy and all that, but in fact, if you look at them, it – the effort, each step, is identical to the effort you're solving a KKT system. It's not a big deal.

If you profile a code that is running one of these methods, they take 20, 25 steps, all it's really doing at each step is solving a KKT system 20 times. That's it. Maybe 30. So that's it.

Okay. By the way, you've been using one or two all quarter, because you've been using CBX, which compiles your problem to a cone program with second order cone and STP constraints, and then it calls STP T3 or SeDuMi, one or the other. Each of those is just one of these fancy primal dual interior point methods. That's all it is.

And you could probably even look at some of the diagnostics coming out of it, which you probably haven't looked at before, but you can actually start – now you'll know what it means when it – you'll see all sorts of things. You'll see dual, residual dual. You'll have the primal feasibility residual. You'll also see things like when it – at least, when it reports the number of steps, it shouldn't be more than 60 or 80.

I don't – it can be up to 50 or something like that, and it's often 18 or something like that if it's a simple problem. But now you can start looking at the stuff that was spewed out.

Okay. So what we're gonna do now is – actually, I'm just gonna – we're gonna finish, actually, today, and I'm just gonna give some of the conclusions, and I'll just say, I don't know, wrap a lot of this up.

I think some of this was Arguis did in the section yesterday. So I mean the main thing here is – I mean the focus, my focus, has been on modeling, so that's, to me, the – I think the interesting part is basically what are the problems you can solve using these ideas.

Well so it should be completely clear by now if it wasn't before that there's lots of problems in engineering design, statistics, machine learning, economics. These – they can be expressed as mathematical optimization problems. I mean that's – that's clear.

A lot of people take that and say, no, no, no, there's – everything is multi – whatever, multi-objective. And well, so what, okay, no problem. So that's fine. So the main complaints people mostly make about this is something like, if you go to some field where they've had bad experiences with optimization, and by the way, there are a lot of them. A huge number of them. So it's extremely common to arrive in a practical field, talk to people, and they go, like, we tried optimization in the '60s. It really, really sucked.

And then they give you a long story, all of which is kind of misconceptions, but it wasn't helped by the people from optimization who went to try to help them, right?

So it would be – they'd say things like, well, we had – our constraints are not fixed, and you're like, well, we can handle that, and then they say things like, well, the other thing is there's uncertainty in our data, and when we optimize, then we plugged it back into the real thing, and when you change coefficients, the whole thing doesn't work anymore.

Well that's just called bad optimization. Okay. Now, the important part is actually tractability here, so you can write down any – anybody can write an optimization problem down. It is a normal part of intellectual development to realize this and to go into kind of a – to go into a state where, for the next two weeks, you look up, and you go like, my God, everything is an optimization problem. Absolutely everything. It's amazing. Every class I'm taking, every lesson I've ever taken, everything I might ever wanna do is an optimization problem.

At some point, you calm down, or not, because some people are arrested in that cognitive state. But if you – you calm down when you realize, yes, but so what? It turns out, most of those problems you can't solve.

So this is very rough, but I would say tractability kind of requires convexity. I mean there's – this is very rough. There's people who would argue with this. It's extremely easy to come up with – come up with counterexamples either way, okay? So non-convex problems that you can solve with low complexity. I mean, one would be this, you know.

I mean, if I asked you to maximize x transposed p x subject to x transpose x equals one where p is a symmetric matrix, I mean, that's a perfect example where it's not convex by a long shot because I didn't say p was positive – negative definite, if I'm maximizing, and this constraint is, for sure, not convex, and yet you can solve this very easily. The solution is whatever the maximum eigenvalue. The maximum eigenvector of p, okay?

So there's an example of a non-convex problem that you can – that's tractable. The point is that there's very – there's just an isolate – a very isolated, small number of these things, okay?

And there are other ones that are combinatorial optimization problems that are solvable and so on. Okay. So – but it's not too bad to say something – to make a statement like that. As a zero authoritor statement, I would stand by this.

Now, if you have a method for non-convex problems, these find local or sub-optimal solutions, or are very expensive. This is called – this is global optimizations, if you guarantee to find the one – they're very – they can be very expensive.

By the way, these absolutely have their uses. I mean so these are extremely useful in lots of areas. If you're doing circuit design or something like that, obviously, if you can have a convex formulation and assert that what you just found is not just the best circuit design you could find, but the best anyone could find, that's a better thing – that's better than not being able to make that assertion, but it's very useful to be able to say, I just reduced the power of your circuit 30 percent.

And if someone says, is that the global solution, you'd go, I don't know, but that's – 30 percent is 30 percent. It's great. So it's to be understood that these are very, very valuable.

The other real thing is this. Until, I don't know, 15, 20 years ago, it was kind of presumed that very few problems were convex. That's because no one was looking for them, but it turns out this is just not the case.

So let's say a little bit about the theoretical consequences of convexity. Sort of the one you learn on the street would be this, right? Local optima are global, okay? That's the thing you'd learn on the street.

The interesting thing is duality theory for – in convexity actually is now useful. You have a very extensive duality theory, and this is just a systematic way to drive lower bounds on optimal value, so -- by the way, that's even for non-convex problems. That – there's a lot of stuff going on on that right now.

You get beautiful necessary and sufficient optimality conditions that extend the usual ones, the Lagrange multiplier things you see in your – whatever your second class on Calculus or whatever.

And you get interesting ideas, like when a problem is infeasible, you actually get a certificate proving infeasibility, and then these are related to sensitivity analysis, all the duality.

Okay. More interesting, perhaps, are the practical consequences. So I would even put this – I would actually put that back in the theory, and since this is practical consequences, I'm gonna go like this. I'm gonna go like that, and I'm gonna even be stronger and say that. 30 to 50 steps. That's the number of steps it takes, period. I mean that's fine.

Now the other thing you should know is that things like basic Newton, they're very easy to implement, and they work for small and larger problems, and they work for larger problems if you exploit structure. In fact, the whole point, obviously, if you're doing this implementing an LP solver, is just to demystify it.

Would you ever use your – the one you just developed? No, because you put two hours into yours. Is that fair? Three? Two? Two. I think less. I mean, I think if you didn't make any mistakes, it's less than two. So you have one per – well between one and two person hours into it, and trust me, STP T3 has a whole lot more.

On the other hand, actually, there are cases where your little LP solver, you might wanna use. You might wanna use your little LP solver, for example, if you're doing a real time implementation, and you wanted to solve an LP, let's say, in a millisecond. A little small LP, right?

The problem is that there's so much junk in all these big ones made for solving huge, large-scale problems, that by the time all the libraries and things are loaded in, a millisecond is gone. That's an exaggeration, but it's close. By the time all the data structures are set up and all the memory has been allocated, a millisecond is gone. Not quite, but anyway. You actually might use your little baby thing, because in fact it would – if you wrote that in C, and all that, and maybe LA PAC calls and all that, you would have something that would run super-fast.

So, there actually – there'd be no reason not – in that case, you might actually use it. However, the point was just to demystify it. To say that – to point out to you it's – these are not very hard – very simple methods actually work shockingly well.

Okay. So let me – let's talk a little bit about how to use convex optimization. So let's see. So what you would do is something like this. If you're in some applied – I guess, in this class, we've focused on problems that actually were posed more or less exactly as convex problems, because you start there, right? That's where you start.

Then, later, you branch off. Once you're kind of comfortable with that – I mean, we don't give you problems that are not convex or whatever. I mean, with a couple of – well, we don't. So, after a while though, don't get the idea that it only works if someone throws in some non-convex constraint that you're – that there's nothing you can do and all that kind of stuff, and throw everything out, because there's all sorts of cool stuff you can do.

You can ignore – you can comment out constraints you don't like. That's got the fancy name of relaxation. That's one option. So if there's a non-convex constraint you don't like, you could relax it, comment it out. And the other thing is, of course, you can just do approximate modeling, right?

If something looks like – if something looks like that and has this small section where it's non-convex, you can fit a convex thing to it. And of course, what should you do is start with simple models and small problem instants, and basically an efficient solution. That is, don't attempt to exploit any structure.

So if you're doing image processing, you start with 30 by 30 images, obviously. Because if you can't make what you want happen in that 30 by 30 image or something like that, then there's no point worrying about how to make it run for a one k by one k image. I mean just for example.

Now it's always useful, when you work on one of these, is to work out, simplify and interpret the optimality conditions in dual. Actually, what very often happens is you get a beautiful, commutative diagram, and it won't be the one you think of, or you'll get a really interesting thing.

You’re working on some problem. You’ll work out the dual, just mathematically, just you turn the crank. Form the Lagrangian, minimize it from the dual function, write the dual down. You stare at the dual for a long time. It is very often the case that the dual has a practical – has some kind of other practical interpretation.

So, for example, this is universally true in areas like finance. It’s true in mechanical engineering, electrical engineering, almost any application I can think of, and some of it is something really cool. Like you’re doing experiment design, and the dual turns out to be one geometric problem, like a minimum volume ellipsoid thing, and it’s just kind of – so whether or not that helps you with your original one doesn’t matter. It’s just kind of cool to know that this is the case.

So, for example, if you’re doing some maximum likelihood problem with an exponential family, you work out the dual, and you’ll find out it’s a maximum entropy problem or something involving Collac Liveler diversions, and you’ll just say, well that – the last you could say is just, that’s cool, because you’ve just connected one practical problem to another.

Whether or not it has any practical advantage for you is another story. Okay. Now the other important thing I wanna mention is the following. Even if the problem is quite non-convex, you can use convex optimization. You can use this in sub-problems, and you can actually repeatedly form and solve a convex approximation turn point.

And I’ll just say one little bit about that, because these things work shockingly well. I would have snuck one of these into a homework exercise, but the class ended, and I was told I couldn’t assign anymore homework, so – but I will – I’ll show you what I mean by that.

Let’s just take – let’s just suppose – yes, I mean, it’s something like Newton’s method, but it’s kind of cool. So that’s it. You have a set of – a huge set of non-linear equality constraints, okay? But we also have all the other things. I have a polyhedron like this. I wanna minimize, let’s say, f zero of x. Let’s say that’s convex. These are convex, and the only problem is I have some set of non-linear equality constraints.

Now, you know this. This takes you outside of convex optimization instantly, because the only equality constraints allowed are linear, okay? So this is not convex.

You encounter this and you say, well I can’t solve that, and you go, yes, but you took that class. It was 10 weeks, and it was like 10 hours a week of work, and you can’t solve that? You know what I’m saying.

All right. So the ways to do it – I mean this is – what I’m talking about now, these are street fighting methods, so no – you’re not gonna get a theorist stamp of approval on this, but I’ll tell you how this – what you would do. It’s very, very simple.

You start with an initial point x zero, okay? You go over to f, and you get an approximation of f near x zero, okay? And you could do this many ways. You could take the Jacobian. That would be – that'd be kind of the 17th century approach, and indeed, all the way through the 20th century approach, 21st even in some departments.

Okay? So you could – Calculus has been used for 400 years. Why not? Calculate the Jacobian and make an affine model of f. What you – you replace this with the affine model, okay? And you add one constraint very important to this, which is this. So I replace this with f of x zero plus d f of x zero, that's a matrix, times x minus x zero, and I say that's equal to zero.

Now what's cool about that is that's a linear constraint, and I add one more thing, which is this. I write x minus x zero in some norm is small, like something like that. Okay? Everybody see what I've done here?

This is a trust region constraint. This says, don't consider x very far from where you are, and the reason is, although such xs could be quite good, they – this model will no longer be good. So you'd do this. Now, what's gonna happen here is this might be infeasible, and so you might add a residual term or something like that. I won't get into that, but you'd be shocked at the number of highly non-linear problems that will yield to 15, 20 steps of this.

Now what's nice about this is the following. All the convex stuff, like all these inequality constraints and the f, that's being handled for you by convex optimization, so you don't even have to worry about that, okay?

So these things work – can work really well. Have you computed the global solution? Does it always work? No. Have you computed the global solution? No. And so on, and so forth, but they work really well. This is called sequential convex programming. That's one way you've heard of it. You may hear of it from that – with that title.

Okay. So I'll just say a little bit about some of the last topics, and then we'll quit – or topics that we didn't cover. So we didn't talk about methods for very large-scale problems, so there's a whole world of that. Some if it's covered in 364 b, so sparse stuff will take you up to 10,000 variables if it's – if the Gods of sparse matrix orderings are smiling on you and your problem, you can take that up to 100,000.

Now, of course, if a problem is banded, for example a signal processing problem or something like that, you can take that to a million variables. You might even sometime soon.

Anyway, so banded – banded problems, you would need to – I mean that's this class. That's not an advanced topic, okay? However, there's really – there's big ones for iterative things, and they'll take you to the million variable, 10 million variable realm. That's for – that's 364 b.

We didn’t talk about sub-gradient calculus and convex analysis too much. There’s other huge families of methods that are quite pretty. They’re very nice, but they don’t seem to me to be in a course where the focus is on modeling and actually using things. They do absolutely have their uses, so these are sub-gradient methods and things like that.

And we didn’t talk about distributive convex optimizations, so everything we’ve been talking about is localized. So, in other words, they run on – of course, you can distribute your linear algebra, and that would be fine, but in fact, there’s beautiful classes of methods that work, for example, for network flow and things like that, and you just get distribute – they distribute perfectly. These are in 364 b.

So I think, at this point, we’ll quit. I’ll thank the crew of TAs, some of – several of whom are sleeping, I presume, so – at the moment, but I guess they’ll get the message later. So, boy, the put a lot of effort in. So just remember, when you’re doing the final, if you think you’re putting a lot of time into it, trust me. They put a lot more time into it. Anyway, so I have to thank them, and thank you for putting up with 10 weeks of this, and we’ll quit here.

[End of Audio]

Duration: 75 minutes