ConvexOptimizationI-Lecture17

Instructor (Stephen Boyd):Well today, we'll finish up Newton's method, probably won't take up much of the day though. But – then, we'll move on to our absolute last topic, which is Interior Point Methods for Inequality Constraint Problems. So we're studying methods, Newton's method, for solving the following problem. You want to minimize f of x, which is smooth, subject to a x = b. So we're assuming here that a x = b is feasible. I mean, otherwise, the problem is, the whole problem is infeasible; and that we have a starting point x zero that satisfies a x zero = b. So we assume we have a feasible point.

And we're actually gonna – the first method we're gonna look at, which is the basic Newton's method, is a feasible method. That means that every iterate will be feasible. They'll all satisfy a x = b. The reason for that is simple: The Newton's step, at a point x is the solution – it's actually the – this is the Newton step. It's found by solving this set of equations here and that's the Newton's step; we looked at what it means last time. Now the second block equation says a delta x Newton = zero. So it basically says that this search direction is in the Null space of a.

Now what that means is very simple. It says that, if you have a point that satisfies a x = b, and you add any multiple, that's the step-length times an element in the Null space of a, obviously you continue to satisfy a x = b. So feasibility is guaranteed, all iterates are going to be feasible no matter the step-lengths, and so on. This is the basic Newton method. Okay. So Newton's method looks like this, with equality constraints. I guess you'll be implementing one; some of you may already have. In fact, we know some of you already have because you've written with questions. So. But all of you will be shortly writing up a Newton's method with equality constraints.

So it works like this. You're given a point in the domain of f, of course, with a x = b and you compute this Newton's step, that's by solving this set of equations. This set of equations has got lots of names; it's – I guess, mostly this KKT system, is what this is because that's – nah, it's not the KKT system, this thing. I guess this is called the linearized KKT system; that's called the KKT matrix because that's where it comes up. A lot of people would just call it the Newton system. Do you solve? You compute the Newton step. If the Newton decrement is small, you quit. Otherwise, you'd do a line search by – you just do a standard backtracking line search.

So you try t = 1, if that gives you enough decrease, fine. Otherwise, you'd choose beta times t, beta squared and so on – beta squared times 1, otherwise known as beta squared. And then, you finally update. Now this method has the following attributes. The first is, of course, it's always that point x is always feasible because every step, everything direction, all of these things are in the Null space of a. And if the original a satisfies a x = b, that's preserved because of this. So it's a feasible method. And of course, it's a descent method, so f goes down with each step. And that's ensured by the line search.

Okay. Now this method is affine invariant, that's easy to show. If you change coordinates, it's – you generate exactly the same iterate, so you get a communicative diagram there. It has all the features you'd expect to see of Newton's method, which is the affine invariants. What's interesting about is, in fact, this is identical to applying unconstrained Newton's method to an eliminate – after you use elimination of variables, linear elimination of variables, so it's absolutely identical. That means – it means we don't have to have a new convergence theory or anything like that. You don't have to see a new proof because you've already seen it.

It also means that all the practical inform – stuff you've seen about Newton's method, which is like the fast convergence, the quadratic, all that sort of stuff, that's absolutely preserved here. Okay. So you might ask if it's the same, why do you have to go to all the trouble of formulating it this way, and not by elimination of variables. And actually, it's a good question and I'll answer it right now. The reason is this: There's no reason – there's a lot of people who would think eliminating variables is better, and there're actually cases where it is better. And I'll tell you, right now, when and why it is not.

If this matrix here has structure and then you eliminate variables, eliminating variables can destroy the structure. So actually, if there's one thing you should take – a couple of – you should take away, like I don't know – we'll make a little tri-fold card. That's too much, even. We'll make a little pocket – a wallet card of the things you should remember when you leave this class. But one of them should be it is not always better to solve a smaller set of equations. In fact, in many, many cases, it's far better to solve a much larger set of equations, provided A.) Those equations are structured and B.) You intelligently exploit the structure.

Okay. So I mean, these are kind of obvious, but you'd be shocked how many people don't know these things, or know them, but kind of don't apply them to their everyday life, or something like that. Okay. So all right. This is Newton's method and you'll be implementing one that works and so on and so forth, has all of the attributes you've seen, Selfkin coordinates all identical. Then you get an absolute complexity-bound on the number of steps required, and so on.

There's a variation on Newton's method that's very interesting. It won't play a big role for us, but it's actually important to point out. So here it is. There's a variation on Newton's method called an infeasible Newton method. And in an infeasible Newton method, it's actually very, very interesting how it works. You want to minimize f of f x subject to a x = b, something like that. In the feasible Newton method, x – the original x is feasible and then every iterate after that is feasible. So you will always satisfy a x = b.

There's another version of this, which works like this. You're actually allowed, for intermediate steps, to actually not be feasible. So I can start with any x I like. So here's an example. You're solving this one, or you will solve shortly, on your homework, this. You'll minimize minus sum log x i – I think. I can't remember; is this what you're solving? I think it is. Yeah, something like subject a = b. Okay? So actually, finding a – I think this is what your solving, right? Yeah, it is.

So here, in an unfeasible method, you're allowed the following. By the way, in an infeasible method, you cannot be outside the domain of f. So you can have – you must have x positive, actually, with a feasible method as well, obviously. You must have x positive at each step. However, you are absolutely allowed to have a x = b violated. So in other words, you can start with x zero it's just like 1. Okay? Then you're nicely in the domain here, but for example, unless you're very lucky, you don't satisfy a x = b.

Now if you have a method like that, there's a couple of things that get trickier. The convergence gets a little bit trickier, not much. But the first thing is you cannot have a descent method. In other words, in this problem, you cannot expect the objective to go down because the objective actually is meaningless until you're feasible. So some kind of combination – what should go down is gonna be some combination of the objective or some measure of sub optimality and the residual. By the way, once x is feasible, then it makes sense to talk about the objective.

But until then, it hardly matters. And indeed, it's clearly not a feasible, sorry, a descent method because if you start with an x for which f is very small, smaller than the optimal value, it's gonna have to increase as iterations go. Okay? So watching this algorithm, you're gonna want to look at two things. You're gonna want to look at things like a x minus b. That's the so-called primal residual, and the other thing you look at is the dual residual, which will give you the optimality conditions. So here, the way it works is this.

In this case, it's no different, you simply linearize the optimality conditions and you get a set of equations that look like this. By the way, it's identical, if you're feasible, this is identical to the Newton direction for a feasible point because in that case, the primal residual is zero. Okay? So whatever this is, this Newton step at infeasible point, it is a generalization of the Newton step when you're feasible. Okay? However, it allows a non-zero point here and this – so you can actually – let's see; we can actually work out a couple of interesting things.

We can see the following: If it always the case that x plus delta x Newton is feasible and you can see that directly by this bottom row. This bottom row says x delta x = minus a x minus b. So in other words, if you multiply – well, like well, multiply it by this, I get a x plus a delta x Newton. And from this bottom equation here, a delta x Newton is minus a x plus b, and I'm done. Okay? So a x is b. So if you take a full Newton step, you will be feasible. Okay? By the way, in this method, once – if you ever take a full Newton step, the following iterate will be feasible. Once it's feasible, the Newton step you're calculating by the infeasible start is identical to the Newton step for the feasible Newton method.

Okay? So all right, that's actually a good thing to know. I'll say a bunch more about that. So in other words, what happens is – we haven't even gotten to the infeasible-start Newton method – but what'll happen is, if you ever take a step of size 1, a full-undamped step people call that, then the next iterate will be feasible and all future iterates after that will be feasible. So you'll achieve feasibility. Okay. And let's see what – and people, you can interpret this many, many ways.

You could say, simply, that you're linearizing the non-linear optimality conditions. You can also – maybe that's the best one; that's maybe the simplest way to do it. Right. Some people would also call this a primal-dual Newton method because you're updating both – you're updating both the primal variable and this dual variable at the same time. Of course, that's true of Newton's method, as well. But still, I guess that's the idea. So this infeasible start Newton method looks like this. You start with a point; it has to be in the domain of the function, but it does not have to satisfy a x = b. That's – it does not.

And you start with some initial Legrange multiplier; actually, it turns out the initial Legrange multiplier is totally irrelevant, but that doesn't matter. You compute the primal and dual Newton steps; you compute these. Then, here's the important part. You cannot do a line search on the function value; makes absolutely no sense. In fact, in the feasible – infeasible start Newton method, the objective can, and will and, in fact indeed in some cases, it must increase. So it can't go – and the reason is very simple. Well here, I'll just give you a – let's just do a really, really simple example.

Here. Here's your equality constraint. Gives you – that's the feasible set. And here's your – here are the sub-level sets of your objective. So here, I'll just draw the optimal point; there's the optimal point. Okay? What if you start right there? Well f is now at its unconstrained minimum; that's the optimal point. So f has to go up, going from here to here. So obviously, a descent method is not gonna do the trick, I mean, at all. Right. By the way, in a problem like this, if the function were quadratic, what would the Newton step be here? The infeasible start Newton method – step, what would it be from here?

Student:[Inaudible]

Instructor (Stephen Boyd):What would it be? It would be – the step you'd take would be 1, but the Newton step, which is actually the increment, would point exactly to the optimal point because what you're doing is, you're making a quadratic approximation. The function's quadratic, it's not an approximation. Then you're solving the quadratic problem by the linear equations, and you will get this. You will take a step of one, and you'll converge to global solution in one step. If it's non-quadratic, it might not point there, it might point you know, over here. By the way, the infeasible start Newton method will always point to point on this line because if you make a single step of size 1, you are feasible, and therefore, you're on this line.

So it might even point – I mean, I don't know, seems unlikely, but you know, it could point over here. I mean, I could make some sick function that would do that. It would point over here. But if you made a full step, you are then feasible and all subsequent steps will be on this line – will stay in this line. You make a half-step; you might be here, and so on. So that's the picture. You must – the line search is actually done on the norm of the residual and let me tell you what that is. The residual is r dual and r primal. It's like this. It's this norm. Okay?

So this is; I guess people call it a Marek function or Leethanol function for this process. That's a function, which proves convergence of something by – in this case, it proves that r d and r p goes to zero. It certifies r p and r d going to zero because it's a function that is, you know, non-negative, non-negative positive only and zero only if you're zero, and goes down at each step. So this is what goes down. It's not very different. And then r primal and r dual, that's just these. No, I'm missing, that's not quite it. Sorry. This – you have to have plus – there it is. Sorry, it's right there.

That's – this is r dual and that's r primal. Okay? So when you run one of these, you'll want to look at r primal and r dual. Here's what you want to see. You want to see r primal go to zero and stay zero. Actually, if it goes to zero and later becomes non-zero, you have a bug. And then, r dual will then, very rapidly, go to zero. Okay? It'll, in fact, quadratically in the end-game. So you have the norm. By the way, it's the norm, it is not the norm squared, here. So just to make a point here. Obviously, both of those go down but if you worked out the following, if you worked out the d d t of the norm of the residual as a function of the step-length, at t = zero and in the Newton direction, you will actually get, I believe it's minus 1 or something like that. Or it's actually minus the residual. There you go, it's that. I haven't – this is worked out in the book, but you can work this out; it's this thing. What this says is that, if you take a step of alpha, you ex – sorry, if you take a step of t, the predicted decrease is simply this. It just scales the residual down; that's your predicted decrease. You have to degrade that because this is a lower bound by convexity; it's a lower bound. And you scale it back and alpha can be any of the constants you like, point 1, point 0 1. Actually, it appears to not have any diff – make any difference what value of alpha you use, over a huge range. So this is what goes down as a residual. That's your line search. By the way, sometimes when you get feasible, you can switch your line search back to f because now, it coincides precisely with the original Newton method and so on.

Okay. This is what I was trying to – this is – I was trying to recover this, here so that was the – this is the descent cond – that's the directional derivative of this at t = zero plus is this thing. Okay. Actually, you don't have to worry about all these things. This is just sort of to get a rough idea of how the method works. It is – this is actually a very, very good method choice, in some cases. If you have a point that's like not quite feasible, or something like that, but it's easy – for example, if you're doing a so-called warm start. Warm start means you've solved a problem just like the last one, except now it changes a little bit. So you're doing – doesn't really matter what you're doing – but you just solved a problem just like the last one. The data changes a little bit; one more piece of information comes up in – comes in, in an estimation problem, something like that. One more data point becomes available; one more time-step has evolved or something like that. Then what you do is; you can use this with warm start. And in fact, if you're lucky, it would be something like, you know, just two Newton steps, or one would get you right back where you want, incorporating the new data. So if you want to do an online estimation, update stuff as new information becomes available, for example in maximum likelihood, this would work very, very well. Okay. Let's talk about solving KKT systems. This is very important because well, I guess a lot of people don't seem to know it. I guess people who do these things know it, but then – I don't; they don't get around much or something like that. So to solve a KKT system, and that's for infeasible start Newton method or for standard Newton method, with equality constraints, you have to solve a system of equations that looks like that. And that's called a KKT system just because this thing comes up. By the way, it doesn't only come up in optimization; it comes up in mechanics; it comes up in – comes up in tons of places. I've seen it – I guess I opened the first book of some book on simulating flows in porous media and, like on Page two, I saw this. So I mean, they didn't call it a; they didn't call it h; but it was, basically, this system. So this system of equations is probably worth knowing something about. So here's how you could solve it. I mean, the first thing is just treat the whole thing as dense; it's m plus n. Right? Because this is, that's m by n and that's m high. And so in the worst case, you just commit to this and solve that, just using absolutely standard methods. Which is to say, in other words, this is the h – let's see if I can get it right – h, you know a prime a and then zeroes of the right size backslash v w. So that's you're – well, that was all wrong. Minus and this should be g h. Okay? So this is – that's your just default basic method. This will simply treat this matrix as dense. It is true there's a block of things here that are zeroes and this is fine. It'll do whatever pivoting it needs to make this stable and it'll – it may even do the right thing. One thing it cannot possibly do is, it can't do a Cholesky factorization on this This matrix is symmetric but it is, for sure, not positive definite because the zero's on the diagonal here. And in fact, your homework which we – I forgot to announce that. We've backed off and said that you don't have to do prob – is it 10.1b, I think it is? Yeah, so. Some people have already done it and we heard from them, last night. Anyway. Were you one of them?

Student:[Inaudible]

Instructor (Stephen Boyd):No, okay. So all right. So it's certainly not positive that you can't use a Cholesky factorization because this has negative eigen values. Okay so if you – one way to do this, probably the simplest, is just an LDL trans – I mean sort of the correct method, if you're just gonna solve the equations, is LDL transpose factorization. That's a generic factorization for a matrix which is symmetric, but not positive definite. If it was positive definite, of course you'd use a Cholesky factorization and you'd use a LDL transpose. Now in fact, there are – there's a whole sub-field now on KKT solvers. And you can go to Google and type "KKT solver," you'll get maybe 30 papers on it, and codes if you want.

So this comes up enough that it's studied, carefully. But if you wanted to use a generic methods, you'd use something like an LDL transpose factorization. By the way, this might not be a bad idea, if this is sparse like crazy, here. So if your a is huge and sparse, if h is sparse – and what does h sparse mean, in terms – about f? I mean, actually it's not all uncommon that h is diagonal in these systems. In fact, I guess that's your homework. In your homework, h will be diagonal. Okay? So then this thing has just got tons of sparsity; it's a screa – I mean structure; it's just screaming at you. If you have a di – you know, ignoring it, that's a little block of zeroes. You'd ignore that; that's maybe okay.

But a giant, like diagonal matrix just sitting, there and if you ignore that structure, that's kind of irresponsible. That's over the line, I think. Anyway, LDL transpose factorization will probably work quite reliably here and would take you up to some very, very, very large systems at this work. You can also do it by elimination and the elimination is interesting. It works –it's – well you eliminate the first block and you get this. So the first thing you have to solve is this equation. This is the – one of these you'll see is the Schur complement is going to come up, somewhere. Well, I guess this is the Schur – well, one of these is the Schur. That's the Schur complement.

This is zero minus – it's zero minus this times the inverse of that times that. So that is the Schur complement. Actually, it's the negative Schur complement because there's a minus sign there. Here's what's interesting if you do this method. This is interesting in cases where h is easily inverted. If h is dense, and there's 1,000 variables and 100 inequalities, this is actually not at all a good method to use. You might even just use LDL transpose, here. Well, I don't know. This would be equivalent to it. However, if h is, for example, diagonal, meaning that f is separable, then this is – h inverse is a non-issue, if h is banded.

If h is sparse enough, h inverse is something very, very easily computed. Actually, I should say correctly. Although, that's what people would say; they don't mean it. You don't actually compute h inverse, obviously. What you'd do is you'd have a Cholesky factor of h sparse Cholesky factor, hopefully. And you would back substitute the columns of a transpose. Okay, but you know that, so I won't say that. So people would still say form this, when you're not really – you do form this matrix. What you don't form is you don't form that, for sure.

Okay. I mean, unless h is diagonal, then you would form it. But if h is banded, you would not because h inverse is full. Okay. So you'd form this. The interesting thing is that, the – when you solve the reduce system, you get a negative definite system of equations. I flipped the sign on it because if it's negative definite, it means you flip the sign, it's positive definite. And this is the Schur complement – well, assuming h is invertible, then this can be solved by Cholesky. So in fact, if you solve this by elimination method, you will have two calls to a Cholesky factorizer.

The first one is the h, itself. By the way, that might not even be a real call, if h is diagonal then there's no call because diagonal is trivial. But in other cases, there'd be a call that you'd do a Cholesky on h. You'd form this matrix; that's the Schur complement and then, you'd actually solve this by some kind of Cholesky factorization. So that's the – so I guess people – so that's another way. So people would call that, maybe, the elimination in Cholesky, would call – but you're gonna make – you may make – you may be making up to two calls to Cholesky factorization.

Okay. Now, in some cases, h is singular. But in this case, you can do things like this: You can add any multiple of a transpose a with a q in the middle like this. Solve this equation, and it's the same. So sometimes, it'd be something like h would be all positive and, like, one or some little block of zeroes, it doesn't matter. But this can be done. That's all subsumed in solving this set of equations intelligently. So let's look at an example. It happens to be exactly the example that you're gonna solve on your homework. Of course, that does imply that all of the source code for this, you can find because it's all posted. So. I think every example in the book, all the source code is there for you. If you want to see how we did it, you can but we'd encourage you to try to do it yourself. But there's no reason for you to bang your head on the wall, so you're welcome, at some point, to look at ours. See if you like ours. Actually, you can find it anywhere; it's tons of places. Okay. So let's look at – so here, you want to minimize minus the sum log x i. That's the so-called log barrier for x strictly positive subject to a x = b. We have a point x, which satisfies a x = b and x is positive. So we start with that, if we do the prime, the whatever you call it, the standard Newton method. Now, the dual problem for this is this. It's maximize minus b transpose new plus sum log a transpose new plus n. So. The n, of course, doesn't matter but if you want the optimal value of this equaling the optimal value of that, then it matters. And in this case, the function in the Legrangean, which involves log – it's a x plus – it's actually the sum of the logs plus nu transpose a x minus b. That Legrangean is strictly convex in the x i and that means you can, if you solve the dual, you can recover the primal optimum from, by minimizing the Legrangean. Okay? So that's how that works here. So I this case, you can solve either one – either one you can reconstruct the solution, in fact basically, if you solve the primal problem, you're gonna solve the dual one, whether you like it or not and vice-versa. By the way, this is quite interesting, this topic because – in fact, I even for a long time, was sort of ignorant about this. A lot of times, people get all excited or bent out of shape over, like the dual or something like that, because what will happen is this. They'll have a problem. I don't know; they'll have this problem here. X i will have size, you know, I don't know, 10, 000 – x will have the size 10,000 and there'll be 100 equality constraints. And they'll say, "Wow, you could solve a problem with 10,000 variables, or look at that, you could solve this dual and it only has 100 variables. Boy, I would really rather, like, I'd really rather to solve a problem with 100 variables, than 10,000." Okay? And that would be true, if you don't know what you're doing, speaking with respect to numerical linear algebra. It's absolutely true, for sure. In that case, if everything cost you n cube or n is the size of the system, there's no doubt you want to be solving 100 variable systems, not 10,000. Okay? What's actually gonna work out here is, it turns out is, it's actually absolutely identical. Solving the primal, dual, any of these, all of these methods, the order is the same. It's only the same if you know what you're doing with respect to linear algebra. If you don't, if you don't exploit sparsity, then you will find gross differences. By the way, if you're in that – if you make that mistake, and I've been there myself, so I speak as a reformed someone who once didn't know the subtleties. You can make stunning claims. You can say things like, "Wow, I found the best way to solve this problem. It's amazing. You solved the dual instead. Isn't that genius, isn't it?" Right? Now by the way, there are cases where there might be some convenience to solving the dual, like it might – you might end up with a distributed algorithm. But in terms of numerical linear algebra, you're – it's just wrong. Solving the dual is the same as the primal, if you know what you're doing. Okay. So. These are the two problems. You'll solve this one. Well, you can solve whichever one you like. So here's some examples. The first is – and they have different initialization requirements, and these are quite interesting. And in fact, that is actually the only thing that should drive the choice of method, assuming you're gonna do everything right. Initialization. So if you're gonna do Newton method with equality constraints, that requires a positive x that satisfies a x = b. Now by the way, this is checking whether a linear program has a strict – a set of linear – well, a certain, a standard form of linear program has a strictly feasible starting point. That can be hard as – that can be hard. There's no reason to believe that's simple. If I just walk up to you and say, “Here’s a; here’s b; please find me a positive x that satisfies x = " That’s as hard as a linear program, basically. Because I’m basically saying, “Find me – check feasibility of the standard form linear program.” So on the other hand, it depends on what the problem is, right? If this is some resource allocation problem, or something like that, it might be completely easy to say what – to get a point like that. For example, if it’s a portfolio allocation problem, I mean, it might be that, here’s an initial x zero. Put – if you have ten assets, $1.00 to invest, ten cents on every asset. It might be a stupid portfolio allocation but one thing is certain; it’s feasible. I’m just saying that, you know, depends on the situation there. It can easily be – in some cases, it’s easy to get such an x zero and others to stop. Okay. So if you do this, here’s what happens. This is a plot of f minus p star so it’s, in fact, this is the sub-optimality. And this scalar, you can see is extremely broad; it goes from like, whatever, 300 down to 10 to the minus 10. And this is the thing you want to look for. If you don’t see this – if you don’t see that – if you don’t see these plots rolling over, then you haven’t achieved quadratic convergence. So by the way, sometimes well, you can get things where it’s much more subtle. I mean these, it’s just sort of very much in your face, here. These are the right numbers, you know 10, 15, 20 steps. These are, you know, big problems and let me say a little bit about this. Sometimes, it’s much more subtle and it just curves a little bit. And if you’re banging your head, that’s enough. But if it kinda looks straight, then you really can hardly claim that you’ve reached – that you’re achieving quadratic convergence. It probably means something is wrong. So by the way, out here, what would the step-length be? What would you expect the step-length to be out here?

Student:[Inaudible]

Instructor (Stephen Boyd):1. Right. So real quadratic phase is characterized by unit steps and quadratic convergence. That would be out here. What would you expect the step-lengths to be here?

Student:[Inaudible]

Instructor (Stephen Boyd):What? Well they might – they certainly could be less than 1, but they can also be 1. You don’t – I mean, you don’t – the answer is you don’t know. But you wouldn’t be surprised if you started some, seeing if there were lots of steps out here where you were less than 1. Okay? So this is just four different – four different, maybe, starting points, here. It’s the four different trajectories. Okay that’s the first. This is the method you’ll implement. Now you can also – you can also solve the dual here, but totally different size. The dual has got far fewer variables, right? Well, not far fewer; it’s got m and I’m assuming sort of here, well m has to be less than n, but it’s got fewer variables.

So it’s got m variables. Okay. Oh by the way, here’s an extreme case. Well, let’s go all the way to the caricature. Ready? Here’s the extreme case. Find the analytic center with one inequality – one equality. So that’s just a transpose x = b x positive. Okay? Here, I guess there’s a couple of things you could say. Let’s see how to solve it. The dual has how many variables?

Student:[Inaudible]

Instructor (Stephen Boyd):One. How do you solve a one-dimensional convex optimization problem?

Student:[Inaudible]

Instructor (Stephen Boyd):How?

Student:[Inaudible]

Instructor (Stephen Boyd):No, you could a line – I mean you could do it by section, if you like. I mean, you could do anything you like at that point. Okay. And you'd think, "Oh, wait a minute, wait. I've got a 1,0000,000 variable problem with 1." So this might be an image; that's an equality constraint. By the way, it's a stupid problem, but anyway, you have an image with an equality constraint and you want to calculate the analytic center, which would be some – and you could wrap some big story around it, like, "It's the maximum entropy image." or something with this – just something, who knows, satisfying one equality constraint.

Now, you could do it by bisection, here. Your dual has one variable and you'd say, "That's ridiculous. How could the primal, that has 1,000,000 variables – how could that be as easy to solve as the dual?" It seems suspicious, doesn't it? So we'll get to it. Actually, it's pretty straightforward, how that works out. The KKT system would then look like this; the Newton system would look like this, where that's diagonal. Right? So it would look exactly like that. Indeed, it is 1,000,000 by 1,000,000. I would not recommend storing it as a full matrix, just so you know.

But if you stored this exactly in this form, you should look at that and say, "Well, that might be 1,000,000 equations by – with 1,000,000 variables." How fast can you solve this, roughly?

Student:[Inaudible]

Instructor (Stephen Boyd):It's in m steps, okay? So actually, what are we talking about, for 1,000,000?

Student:[Inaudible]

Instructor (Stephen Boyd):Milliseconds, at most. Okay? So anyway, so these are very important to know, these things, and to recognize them. I guess, there's a lot of people who know you could solve that fast. Although, a lot of them haven't totally internalized it. But what they haven't also done, they haven't, like, propagated this knowledge into the rest of their life. So they're doing signal processing or statistics, or whatever, and they haven't put it all together. They don't know what a statistics problem looks like, that happens to be one that you can solve unbelievably efficiently, like that.

So, okay. All right, let's look over here. So this is a Newton applied to the dual. By the way, this is interesting, totally different initialization requirement. But you need to – here, is you need to find a vector nu for which a transpose nu is a positive vector because that's how you need to initialize this problem. It's completely unconstrained. Okay? And you're gonna – so you need this, totally different problem, initialization requirement. And this one has, you know, nine steps to converge, and again, you see this thing roll over. That's the good – that gives you that good Newton feeling.

Okay, if you apply infeasible start Newton method to this problem – so I initialize it with, let's say x = just all ones. Okay? Just all – I just plug in all ones. You're very happily in the domain, here but you don't satisfy a x = b. And that takes – here are the four things, here. Oh, what I haven't marked here is where primal feasibility is achieved. And so let's make some guesses. If I told you that primal feasibility was first achieved on this run here, what would you tell me?

Student:[Inaudible]

Instructor (Stephen Boyd):You believe that? When do you think it was achieved? I'll go back; you say when. Probably somewhere around here. Why would I say that? Well, I'd say it for this reason. Let's – if you put everything together, it works like this. If you take a step of 1, you will be primal feasible at the next step and you will remain so. So you'll satisfy a x = b, if you take a step of 1. We associate quadratic convergence with two phenomena. No.1) Quadratic convergence. That's this rolling over of this – that's this thing. That's the famous doubling the number of digits of accuracy at each step. That's this roll-over. And we associate it with a step of 1. So when you sort of start seeing this roll-over, you have taken a step – you're probably taking steps of 1, in fact, almost certainly. That means you're feasible. So although I haven't marked it here, a primal feasibility is probably achieved somewhere in here. You know, it could be up here, I don't know. And same for this one. Primal feasibility, maybe, is achieved like here, I'm guessing. Maybe here, I don't really know. Maybe I'll go back and replot these with filled and open circles, to indicate when primal feasibility is achieved. By the way, number that the – note the number of steps is not, you know, it's changing within a factor of 2. You know, it's 9 steps vs. 15 and 20 and there's one that's 21 steps. This is not worth getting all bent out of shape about; it's not worth anything. Okay. Now, let's look at what's required in each of – to solve for the step in each method. If you use the first method, this is the one – by the way, this is your homework problem. You have to solve this system here. Now, in this case, you end up solving a times the diagonal matrix times a transpose w = b. That's the only step here that costs you anything. That's diagonal, here. So this is – you have to form this matrix and then solve this system. Okay? Now, in the second case, for the dual problem, the dual problem is completely unconstrained. Here it is. It's this right – you just simply maximize this function. You calculate the gradient and the hessian; you calculate the hessian inverse times the gradient, end of story. Now the hessian of this, I think you know actually from the last homework. The right way to do it is, really under no circumstances, should you sit down and start calculating partial squared this partial nu i partial nu j. That works, in theory; it's a huge pain. Instead, you use a composition rule. The hessian of this is the hessian of a function which is a sum of logs that's gonna be diagonal hessian. Then you multiply on the left and right by a and a transpose, and I forget which one goes where. But one of them goes on one side and the other on the other. That's the chain rule for the hessian. Well, there's the answer; you do this. Now, so to compute Newton method for the dual, you get this. Now, what's kind of cool about this is, it's actually quite interesting. To solve this primal 1 – sorry, this primal Newton, and Newton diagonal and see I have to be very careful. This is the – if you're applying Newton to the dual, here, this is applying Newton to the primal. You don't solve the exact same set of equations, but the structure of the equations is identical. They all have the – they both have the form. They're different matrices, but they have the form a diagonal a transpose. Down here, a diagonal a transpose, different diagonal. Everybody see this? So you solve the same thing. In this extreme case, that I talked about, a is a row vector that's, you know, 1 by 1,000,000, then when you solve this system, this is basically a number and – but then it's here, too. So it's identical. Okay. If you use block elimination, to solve the KKT system here, that's to solve the infeasible start Newton method, it's the same story. You solve the same thing here, but the right-hand side is different, but the matrix is the same and that's what determines how fast it is. So in each case, you have to solve a d a transpose w = h, with a diagonal positive diagonal. So that's actually all that's gonna happen in all of these cases. So if a is sparse, then it basically says the computational complexity of the three methods is absolutely identical, per Newton step. If you add to that, the fact that the Newton method is gonna take somewhere between five and 20 steps or something like that, and you call that just a constant, then it means that all the methods are just the same. There's no – and in particular, it means there's no dramatic – there's no dramatic improvement you're gonna make here, just none. So, okay. Everybody get all this? So by the way, if you don't know how to solve these linear equations, it's everything's different. Right? The three methods are dramatically different but all you're doing is displaying your ignorance of how to solve linear equations.

Student:[Inaudible] apart from history of solving [inaudible] problem, you don't have any constant. Is it a benefit because it's easier to find the starting point?

Instructor (Stephen Boyd):No, no, no. Not at all, I mean, you could use the infeasible start Newton method. [Crosstalk]

Instructor (Stephen Boyd):You don't have a feasible point.

Student: Because you could have a feasible start point then you'd have more [inaudible] conditions.

Instructor (Stephen Boyd):No, I've been – I know what you're saying. I would like – I mean, I also had this urge and, in the past, did not know these things. Yeah, you want to say one of these is – you would like to say one of these methods is better, right? Well, we've just discussed the computational complexity issue. If you know – if you know what your doing, that's a big "if," it's identical. So the cost per Newton step is the same in all cases, not because you're solving the same set of equations. Because you're solving

– you’re not solving the same set of equations. What you’re doing is, you’re solving a set of equations with exactly the same structure. That’s why it’s comparable. Okay?

Then, you could ask, “Does one take more in steps than another?” Well, I don’t know, you know. No not really, I mean within a factor of 2, they’re all the same. Okay? So you know, if you – here, in this example, the dual method looks, you know solving Newton to the dual looks better. I can make another example where it’s worse. Okay? Okay. So then, everything comes down to initialization and, there, I can think of situations, practical situations, where all three methods are the best one. So or there’s just no difference between them.

So I hate to say it, but I think they all have, at the moment, they all have equal status. The main thing is the initialization, as to which one is better or whatever, depends on the situation. If you have an obvious dual feasible point, fine. If you have an – if there’s an obvious primal feasible starting point, fine. If you have a guess of a point, which is nearly feasible, but not quite, you might want to use infeasible start Newton method and so on. So okay.

So now, we’ll look at some other examples. They’re fun and they all have the same theme, which is, you know, you look at a Newton method and then we’ll talk about forming it. And, but what you’re really doing is, you’re looking for structure to solve here. So let’s look at network flow optimization. So network flow optimization is, this is a single commodity flow; it’s a beautiful problem. You should all know about it. So you have a network, a directed graph like this; it doesn’t really matter, I mean. And you have flow on the edges. And a flow – these arrows actually only tell you the orientation of the flow.

So it’s like an electri – in a circuit analysis, you make an arrow here and that tells you that, if current flows that way, you will regard it as positive. If current flows this way, you’ll regard it as negative. So that’s the – now, it could be that, in fact, things can only flow this way. There are many cases where that’s the case. But generally, when I mark these, this is what you have. So you have – and you have things like you – this could be a single commodity network or something like commodity flow. So these – then you have external inputs to the network. Oh, you have one very important requirement. The requirement is that in elect – in circuit theory, you call it KCL, which is Kirkoff Current Law and it basically says that, at each point, flow is conserved. The sum of the flows at any node, is zero. Okay? So that’s what it says. And sum of the flows means, if things – it says, some people would say the sum of the in-flow is equal to the sum of the out-flow. Another – but this is the same and you get this in lots of things. You get in physics; you get it all over the place, in anything. It’s mass conservation, for example, or it’s just a conserva – it just basically says the flow is conserved at nodes. And if it’s a differential equation, I guess you’d say it’s a divergence free or something like that, whatever it is there. Okay. So and you could think of this as any way you like. These could be flows of packets on a network, and the fact that it’s conserved says the buffer is bounded or, you know, there’s either no buffer or the buffer’s bounded and it’s not growing or something like that. And it’s in steady state. It could be actual current; these could be amperes and it says you're in steady state, or something like that. Okay. Okay, so this is – that's flow conservation. You have some external sources and syncs. So it's generally, if you're pumping stuff in, you call it a source. If you're pulling stuff off, it's called a sinc. Actually, I'm just drawing these as if all the variables were positive here, for example. But of course, this could be a source if this flow is negative. Now, in this case, you have to have the sum of the source as zero. That's very easy to work out. Otherwise, it doesn't make any sense. So this is – that's a network. Now, there'll be lots if you fix the external flows. So if this is a distribution network or a current network, or something like that – we'll make it a product distribution network. If this one is pumping in something, this is pumping in something; this is pumping in something, then what you're pulling off , here at the sinc has to be equal to the sum of these. But there are lots of flows in here that satisfy the flow conservation equations. So you can actually route the flow any way you'd like. Routing the flow, basically means, what comes in here, some you ship there; some you ship there; some you ship there and so on. Okay? So that's the picture. Anyway, this is just written as a x = b. B is the source vector here, at each node and a is the incidence matrix. So that's the matrix – you've seen this, should've seen it, that looks like this, where you have nodes and edges like that. And you have a plus 1 or a minus 1 to indicate where that edge goes, the from and to node. So that's just a x = b. Now, among all the possible feasible flows – by the way, there's all sorts of interesting things you can say . The Null space of a, for example, is called the circulation because, if I add something to Null space here – so the circulation might be something like that. Notice that it does not affect the external flow. It adds to this flow, adds to this one, subtracts from this one and subtracts from that one. It subtracts because I'm going – my circulation is going against these things. So that's an element of the Null space. You might ask why on earth would you want to have anything to do with one of these? Well, we'll see why. What you want to do is, you have a cost now, on every edge, and you want to satisfy the flow equations, and you want to minimize the total cost. So by the way, let me just quickly ask something. Right here, there's something that should pop out if you're fully trained in all the neural pathways or hooked up correctly. Created. You should be able to look at that and without even thinking, 50 milliseconds later, a picture of the KKT structure should show up in your head. And what is it?

Student:[Inaudible]

Instructor (Stephen Boyd):Okay. When you see that sum of scalar functions, what is the hessian?

Student:[Inaudible]

Instructor (Stephen Boyd):Thank you. So you should just look at that and see in your head, this, without anybody saying anything else. That's what you should see, when you see this. Okay? So I mean, it should be just an autonomous response. Okay. All right. So anyway, we'll go through this; this is the reduced incidence matrix. And let's look at the KKT system for this. The KKT system looks like that. So h is diagonal. That's the important part, is that of course, you should always see this when you see an equality constraint minimization problem, but in this case, that's very important; that's diagonal.

You can solve this via elimination. You have a h inverse a transpose a transpose w = this thing. Now, of course, when you actually go to write the code, you have to get all the right-hand sides correct, and all that stuff or it's not going to work at all. But in your first pass over a problem, you want to think about the structure. And so basically, in this problem, the inverse shouldn't bother you at all because that's a diagonal matrix. So it's – although normally an inverse would set up a green – a red flag, which is, you need to look at that. Is that hard, easy to do and so on? In this case, it just goes away. It just – yeah, of course, you have to actually carry it out correctly in your code. But non-issue.

All right. So basically, what it says is, if you want to solve a network flow problem, each step of Newton's method is gonna require you to solve a systems of form a diagonal a transpose. By the way, that's a theme; you might have noticed it. You will hear about a diagonal a transpose, actually just everywhere. You might ask like, you know, what do you do when, you know, if you wanted to ask, "What does spice do?" You know what spice is, right? Circuit simulator program? You might ask, "What does spice do?" Would you like to know what it does if you profile it? It's solving systems that look like a d a transpose. That's the only thing it does.

Everything. And I'm talking about for all calculations it does. Everything, you know, [inaudible] as if it – actually, how many people know what I'm talking about? Anyway, or know about spice? Okay. So you get the same, you know, if you do a thermal simulation, right, you're solving a d a transpose, with a finite element method. If you solve a Poisson equation, what do you think you're doing? You're solving a systems perform a d a transpose. So it's just, you know, there's no reason – I mean, you might as well accept the fact that it's everywhere. And you will – you have seen it; you will see it; you will use it. You'll use it, many times when you don't even know you're using it.

Okay. So I'll have to think about where you're using it in the machine learning, or something like that, but I'm sure you are somewhere. I just don't know where exactly, right now but I'll think of it and tell you. I'm sure there's somewhere where you do this, some base network calculation or something. But anyway. Okay. So you have to solve that. By the way, the sparsity pattern of a diagonal a transpose is extremely easy to work out when a has a graph incidence matrix. It's very, very easy. It turns out that a diagonal a transpose i j is not zero, if and only if – that's a nodes by nodes matrix that you're solving

And that's true, if and only – that's non-zero if and only if it finds j r connected by an arc. So that even tells you, now, about the sparsity pattern you're gonna see. In fact, I can ask you some questions now and we'll see how well you've entered – this isn't – I just want you to guess or whatever and you tell me. Suppose I have a network. I want to do network flow optimization. On my network, the fan-out or the, sorry, the degree of each node on the network is like small, like 3 or 5. But it's got 10,000 nodes. Do you think we can solve – we can do optimal flow fast on it? Yeah, probably because if you form a transpose a, it basically says – it tells you something about the number of row – non zeroes per row in this thing, right? And the answer is, you know, each row's got only – you're only – if each node is only connected to, like, three others, that's the number of non-zeroes per row or column. Okay? So that's a, you know, how fast can you do it? I don't know; it depends on the gods of heuristics sparsity ordering methods and whether they're smiling on you that day or not. But the point is, it's likely you can solve that problem very, very fast. Now, let me ask you this. Let's do a – let's have a node, a single node is connected to a huge number of others. What does that do in sparsity pattern in a h inverse a transpose? Everybody see what I'm saying? So in fact, let me draw my network. I have a network here, which is kind of sparse; everybody is connected to just a few others, you know. I don't know, it doesn't really matter. Okay? But I've got some super node here and it's actually connected to a whole bunch of them. What does it do?

Student:[Inaudible] it to the bottom, won't it just preserve your arrow structure?

Instructor (Stephen Boyd):Okay, as you permute it. Okay, very good. So your proposal is that this should be ordered last. Okay? And what is the sparsity pattern on a h inverse a transpose, if you order this last? In this case, what's the sparsity pattern on this thing? It's arrow – yeah, well no, it's like sparse arrow or something like that. It's – you get a sprinkling of entries from these guys. But from this one guy attached to all of them, that is gonna give you a dense row and column in this matrix. Okay? Now by the way, you could order it to be the end, but in fact, any decent heuristic for ordering is gonna put that guy at the end, anyway.

So but you'll know it's gonna work. In fact, you'll know that if you had a network with 10,000 nodes, 10,000 nodes and the degree of most edges is 3, but 100 of them are connected to giant piles of nodes – we'll call those "super nodes" or whatever. It doesn't matter, whatever you want to call them, right? Because you're maybe optimizing the flow on a peer to peer network or something like that – you would know what the structure is. The structure will be dense, with 100 much denser rows and columns and you'll know, actually, that can be solved very, very well. Maybe you'll know, but anyway. Okay. So.

I didn't expect everyone to follow all of this, but this is – actually, this is the kind of thinking you want, I think you want to do for this stuff. You want to just be on your toes, connecting sparsity and structure to the actual problem at hand. And the problem at hand, it'll be things like, you know, bottlenecks – you know, bottlenecks, super nodes, that's for this one. But for every problem, there's gonna be something like that.

Okay, we'll look at one more. It's an even more advanced one, but it comes up in a lot of things. You want to calculate the analytic center of a matrix inequality. Let me, just first, motivate it a little bit. This comes up all the time, but I can tell you what this is. This is the following problem. I have a co-variance matrix and I tell you some things about – well, let's see. I have a random variable z, let's say, and let's make it galcion or something like that; let's just say it's galcion. Okay? So it's n zero x – actually I'm gonna call this co-variance matrix x. Okay?

And here's what I'm gonna tell you: I'm gonna tell you some variances of linear functionals of z. So I will tell you, for example I've got a vector c, c i and c i transpose z. I'll tell you the variance of that. Now this is nothing but c i transpose x c i. Everyone agree with that? It’s nothing but that. Therefore, that’s the same as the trace of x c i c i transpose. The linear – the trick – the variance of this linear combination is a linear functional of x, a co-variance matrix. Everybody agree? Okay.

And now, so I’ll give you an example. I have a random variable and I say, “Well, the first component has variance 12; the second has variance 3; the fourth has this variance; the fifth has that. Oh and by the way, the following, you know, the sum – this linear combination of them has this variance.” You can even do things like give the off-diagonal elements. I can say – I can say, actually element 3 and 5 are correlated 37 percent; element 6 and 12 are correlated minus 19 percent. So I just give you a bunch of partial information about a co-variance matrix. And then I say, “Please compute for me the maximum entropy distribution that satisfies galcion; that satisfies those known constraints on the distribution.” Everybody understand that? Well, that would be this problem. Maximize the determinant, subject to some equality constraints. Okay. I mention this just to say this problem comes up. Okay? This comes up. Okay, so now, let’s talk about, let’s solve this problem. Well, first of all, you look at it naively. You would say how many variables are there? Well, let’s see, x is a matrix variable; it’s square; it’s n by n. And so, therefore, the number of entries it has is this in it. Okay? Now, if there’s p of these equality constraints, it basically says that if I form the KKT conditions or whatever, I’m gonna have a matrix that’s this big. Okay? So and therefore, our baseline cost is that. Okay? Now that grows like n q, n to the 6th – I’m sorry that’s n to the 6th. Everyone agree? Because this is n squared inside a cube. This would be – this would start getting tough. You can’t do a matrix bigger than 100 by 100 with this scheme. Not to mention, by the way, this scheme would be an enormous pain in the ass to actually write up. Because, here, you’d have to find some way that, to encode, you know, this thing, the vec operation there. Okay? Everybody see what I’m saying? You could – I could assign this problem. You could write a Newton method on it, just like that, theoretically. It’s be a pain in the ass, so we wouldn’t do it, but you could – this is like, no problem. However, the complexity is gonna be n to the 6th, which is not a great complexity. Okay? So let’s actually see how you do this the intelligent way, you know, exploiting all the structure in here. By the way, the structure, at this time, is not going to be sparsity. So this is just an example. I mean, I wouldn’t expect you to know this. A lot of people don’t know it, actually to this date, don’t know it. But we’ll go through it. We’ll do a little bit of it just to give you the flavor of it. The Newton step is solving a set of – well, look, the Newton step solves a set of n, basically, n squared over 2 plus p equations in n squared over 2 plus p unknowns. The unknowns are delta X Newton and the dual variables nu, of which there are p of them. So that’s what you end up – you have to solve a set of equations like that. It’s a dense set of equations, by the way, completely dense in general. Right? So there’s no sparsity trick is gonna help here. Those equations, however, look like this. They look like that. So that’s a matrix equation, here. So it’s x inverse delta x x inverse. This is, in fact, something like the hessian, or it’s complicated but it looks like that. And there’s a very simple way to derive this, actually. What you do is, you just write out the optimality conditions, which is this, here. Then you linearize this at the current x and the current nu and write out that equation. If you do that, you’ll get this. This – you’d get the same thing if you looked up and figured it out what the hessian of log dead is, and you’d, I don’t know, get a big headache and stuff like that. But then, ultimately, you’d find out the Newton step can be expressed as a set of matrix equations that looks like that. So that’s how – and you have to solve for this and that. Now, this doesn’t look good. However, there’s a block in this system that we can solve fast, and I’ll show you what it is. If I fix w, can you now solve fast for delta x, here, if I fix this part? If I fix this part, then I’m just solving x inverse delta x x inverse is equal to a fixed matrix. How do you get delta x? That’s a set of n n plus 1 over 2 equations in n n plus 2, n plus 1 over 2 unknowns. If you just go to your generic backslash solver or whatever, that’s order into the 6th. However, in that case, it’s very easy. You simply take this thing minus that, and you multiply on the left by x, on the right by x. Okay? All of those operations, matrix multiplication, those are order n cubed. So if you know what you’re doing, you can get – if the ws were fixed, you could get delta x here in n cubed operations. N cubed is a whole lot more favorable than n to the 6th, just for the record. Okay? So that tells you that there’s – although there’s a dense block in this system, it’s one that is easily solved by special structure. By the way, if you’re curious what the matrix structure is, it’s called kronecker. It’s a kronecker product structure or tensor product structure, something like that. So that’s why this is fast to solve. So you solve it by block elimination. You eliminate delta x to get this. You substitute this into the second equation, and you get this. And now, you look at this and you realize that’s a set. This, you can write this. This here is something that looks like this. It’s, here, g w = b. The entries of g are this thing. If you work out what the computational complexity is – you have to calculate every row, every entry of g and then, solve this. That’s cheap; that’s p cubed. But the computational complexity comes out to be this. That’s gonna be one of the leading terms is gonna be this. It depends on the order of m – n and p, but they all look like this. The cool part is, it’s order n cubed. I mean, so whatever the complexity is, it for sure beats the generic one because this thing actually starts out with an n cubed term; it’s n to the 6th. It’s also got a p cubed term. We’ve got p cubed here, but our complexity clearly is way, way, way better. So. So I didn’t expect people to follow all that, but it’s just to show you that these – once you learn these ideas of looking for structure, and block elimination, and things like that, it will come up in lots of places and it’ll be surprising. By the way, I didn’t know – there’s been periods when I didn’t know this. I’ve solved problems without [inaudible] and that was a mistake. In other words, I’m saying I made a mistake. We did it for years and so on, before we knew all these things. I do want to articulate these things because sometimes, you see these things and they just look like some secret trick. And you think, “Why would anyone think to do that? It’s just some one-off little trick.” Anyway, so that’s – it’s not. It’s just block elimination, with some sophisticated methods for solving structured equations. Okay, so that finishes our discussion of Newton method. And we’re on to our last topic, which is interior point methods. And let me explain first the big picture, how that’s gonna go. It’s gonna go like this. You’re gonna have a problem with inequality constraints. Let’s forget the equality constraints because they don’t matter. So we’re gonna have a problem with inequality constraints, here. We’re gonna solve it, that’s our next step, by reducing it to solving a sequence, we’ll solve a sequence of smooth unconstrained problems. By the way, I’m gonna put something above it. Like, I’m gonna put a little node above it. I’ll fill that node in later. But we’re gonna have an inequality constraint problem. We solve it – we’re gonna solve it by solving a sequence of smooth unconstrained problems. How do we solve smooth unconstrained problems? Newton’s method. What does that mean? We actually are solving a sequence of – this arrow means solve this one by solving a sequence and I mean a small number, like 10 or 15, or something like that. I don't mean some huge long limit. I mean solve a sequence of these. So you solve 5, 10 smooth ones. The smooth ones we do by Newton's method. Newton's method is the same as solving a set of quadratic unconstrained problems. Now to solve a quadratic unconstrained equation, that's the same as – these are linear equations. Oops, there you go. These are linear equations. Okay? So and we know there's lots to say about this. By the way, this node up here, would be – we have a non – you might start with a non-differentiable constrained problem. Right? Like you want to minimize the L 1 norm subject to a x = b. Let's say you wanted to do that, for example. Okay? The objective is not differentiable; you can't solve it. I mean, you can't use Newton's method on it or anything like that. It doesn't fit this category. So there, we do a transformation. So this one is a transformation. To minimize the one norm, you do a transformation that goes like this. You'd minimize the sum of the t i subject to absolute value x i is less than t i, which you'd write as, I guess, x i less than t i minus x i less than t i. Right? Everybody knows that. And so you take a problem which is non-differentiable unconstrained – no sorry, equality constrained and you convert it; you add new equality constraints. Now you've added all these new variables and constraints and you might start getting all nervous and weird about it. But if you just remember that, I guess you know, the size of the problem is not what matters. What matters is the structure. Then you'll be okay. And actually, you'll even have a sense, when you're doing this, that when your adding these inequalities, they're very local. And so they're gonna contribute to sparsity in very, very favorable ways. Okay. So but I'll assume you start here. So smooth unconstrained, you do that by Newton's method. In each step of Newton's method, you are actually solving a quadratic unconstrained problems, otherwise known as solving a set of linear equations. So this is how this – this is the hierarchy. The first thing is just to know how it's done. The second is to understand that, at the end of the day, when you profile a code running it, it's only solving linear equations. That's why you need to know how to solve linear equations. And the more you know how to solve them and about exploiting structure and things like that, the better off you are. Because the equations you get down here are gonna inherit structure all the way up from this one at the top, I mean L 1 optimization being a perfect example. Okay? So you'll – so. A lot of people don't know this, but this is – but you will. So. We're now going to talk about this step. How do you solve an inequality-constrained problem by solving a sequence of smooth unconstrained problems? Or equality constrained, I mean they're basically the same. That's what we're gonna do. So we're gonna solve this problem. All the fs are twice differentiable. By the way, to get to this point, you may have to do [inaudible] in an L 1 problem; you may have to do a transformation, to get to this point. So if your fs are not differentiable, you might have to do a transformation. But I'm assuming you've done that and now, we have the inequality constrained, smooth inequality constrained minimization problem. Okay. Well, we'll start by assuming that we have – the problem is strictly feasible and so I can find a point that satisfies equality constraints; all the f is are negative. So that's Slater condition, and that means that optimal – the – if I write down the primal and dual – if I write down the KKT conditions, they're necessary and sufficient conditions for optimality. So what'll be examples of problems like that? Well, obviously l p, q p, q c q p, g p, there are all obvious, g p and convex 1, of course. Examples that are not quite like this, you have to be careful with. Things like s d ps and s o c ps, you can actually formulate them this way, but it's quite tricky. And it's better to this by cone programming, which we'll get to soon. Okay, another example would be entropy maximization. So entropy maximization says this: Minimize negative entropy, subject to some inequality constraints and equality constraints. Right? So if x really is a distribution here on some finite set, then one of the constraints, of course, is that x is a base sum to 1. So one row in a is all ones. And then, the other rows in a, these are just expected value of any function of a random variable. So I can have a random variable on 10,000 points and I could say – I could give you some moments. I could give you some probabilities that are fixed. I can give you inequal – I could give you intervals. I could say the probability that x is in this set is between point 2 and point 3; those would translate to this. I could tell you all sorts of things about the skewness, and the kurtosis, and whatever else I like. Those will all translate to linear equality and inequality constraints. And then, I might say, "Please compute for me the maximum entropy distribution that satisfies those conditions, those prior conditions on the probability." And that would be this problem. This is a good example of a problem like that because here, the inequality constraints are affine here. So they're obviously smooth and the objective is also smooth. Okay. All right. So we'll get to this idea of the log barrier. And if – it's actually an embarrassingly simple idea. It works like this: I want to convert that problem q a to one I can handle. What can I handle? Well, we just finished looking at equality constrained smooth minimization. So what I'm gonna do is take this irritation function, associated with a hard constraint. I mean, it looks like this. It's zero and then it goes up to plus infinity. Okay? It's the indicator function of the feasible – of actually r minus. Okay? And what I'm gonna do is, I'm just gonna replace that with a function that goes to infinity as you go here and then, but it's otherwise smooth. And what we'll do – I mean a typical example would be like a log function. Right? So this is called a barrier function, something like this. And basically, what it says that, instead of being totally cool until you hit zero – this is for a constraint, and then basically, going totally insane because that's what this – but that's just semantics of the original problem. It basically says that it depends on which one you want to pick here. But the point is, you start getting nervous when a constraint starts getting close to infeasible. That's what this, you know – then how close do you get before you get before you start going nuts? Well, that depends on how good your approximation is. And over here, you might feel more and more comfortable the more margin you have. I'm just anthropomorphizing this, but that's kind of the idea. So the idea is, you get the someone who, instead of someone who goes insane, is totally cool until a constraint is violated and then goes insane. You want something where, the more margin you have, the cooler you are, moderately. But and you start getting nervous when you start getting close to this. Of course, you still go insane if you're – it it's infeasible here. Right? Okay. So that's called a barrier. The most common barrier is something like – the way to write it out is this. Is you have 1 over t – you have a parameter, 1 over t times the sum of the log of the negative things. That is a smooth convex function. You know that, sorry. There, at the minus sign, that's a smooth convex function. And the parameter t simply sets, basically, the scale of how nervous you are, depending on how close something is to getting feasible. If t is big, it basically says that you're get – that you're logarithmic barrier here is actually giving you a very good approximation of the true function here. It's, you know, it's very flat, near zero, then shoots up to plus infinity. If t is smaller, like 1, it means it might be this one, out here.

Something like that. Okay. Now that is a smooth problem, this thing. This is – sorry, that, well this is. But this is a smooth problem. We can solve that by Newton's method. Okay? Now, so we can solve this. And now, you can think of all sorts of things. You could set t – you'd set t high enough. I mean, so there's a lot of things to do. I mean, one thing you'd want to do is prove, for example, that as t gets bigger, the solution of this actually approaches the solution of the original problem. I mean, that's really what you want to do. All right? I guess this is the original problem, here. I minus is the one that goes, like, zero and then plus infinity. So that's the original – you want to show that this approximates that and it kind of makes sense that it would. The other thing is there's obviously something fishy here. If someone proposes to solve this by this, it's got – it should raise up all sorts of like red flags for you. Let me tell you how. Suppose some says, "Look, I can't solve," it's kind of like this. And this actually happens. There's lots of like grown people who say stuff like this, words like this. There's an L – here's an L 1 function, and they go, "I can't – you can't solve that." And you go, "Why?" And they go, "It's non differentiable. Look." Now let's leave alone the fact that you solve it precisely – what's interesting about L 1 is precisely that point. It's that point there that makes, when you solve these things, lots of the entries zero. Okay? So let's leave that alone. So the standard method, I call "the sandpaper method." So "the sandpaper method" is this. I'm not gonna say who does this, or anything like that, but there's still like, you know, grown people who go around and say this stuff. They go, "Well, everyone knows you can't solve that. Well, what you do is, you get out your sandpaper and you go towards this tip and you sand it off. You go like that." I mean, a typical one would be to say, "Look, I'll write it this way. There and then, minus there, there's my function." Everybody cool on that? Okay? Now this function, on the right hand side, analytic, totally smooth. Okay? However, the corner has been sanded down at the scale of 10 to the minus 5. Everybody agree? And they go, "Cool. That's like – that's just not – it's not that – that's not twice differentiable, that's like 5 times differentiable; it's like 48 times differentiable. So I can use Newton on it." Now, do you go for that? You're applying Newton method to something that looks like the absolute value function with just a little bit of light sanding, to get rid of the that corner. What's gonna happen? What is the Newton step, for example, like right there? What's the Newton step on a sanded, absolute value function?

Student:[Inaudible]

Instructor (Stephen Boyd):Well, the gradient. What's the gradient? Like, minus 1. What's the hessian there? Oh, by the way, please don't calculate anything, just use your eyeball. What's the hessian?

Student:[Inaudible]

Instructor (Stephen Boyd):What is it? Is it zero? I don't think so. The hessian of the absolute value's function is zero. It's not – the hessian of this is not zero, but you're very close. What is the hessian?

Student:[Inaudible] Instructor

Almost zero. Thank you. It’s almost zero. What’s the Newton step, at that point then? It’s like some insane huge long step, right? Okay. But more than that, how well is Newton’s method gonna work on a sanded-down absolute value function? Well, what do you have to ask yourself? You have to ask yourself this question. If you form a quadratic model of this function at a point, how good an approximator of the function is it? Well, what do you think? Do you think Newton method is gonna work on this? Well, of course, our theorists tell us, yes it will. And, indeed, it’s a proof, of course it will. But how well do you think it’s gonna work in practice, Newton’s method on a sanded absolute value function? You get two guesses and then, we’ll quit. The first is, like, super-well. And the second is the opposite. It doesn’t work, like at all in practice. So which do you think it is?

Student:[Inaudible]

Instructor (Stephen Boyd):The second one, you’re right. Okay.

Student: Why?

Instructor (Stephen Boyd):Because it’s obviously – and if you – and then you say, “Well, how does that work in the theory?” Right down here, the hessian, this third derivative, goes insane. Right? Because the hessian goes from, you know, it goes through something where it goes super high; it’s super low; it gets way high then it gets low again. You have to get down to the scale of 10 to the minus 5 to actually see this curvature. So the point is you’re right. Newton’s method, on a sanded down thing, isn’t gonna work at all. So I’m just saying that you shouldn’t – when people do this business, where they go, “Oh, it’s non-differentiable. Not a problem, give me my sandpaper.” you shouldn’t – generally speaking, you shouldn’t believe stuff like that. This one’s gonna work out, but this one does not. Yeah?

Student: You were saying, before, that if we’re solving a problem like that, that we have to do something to make it differential before we even start this process.

Instructor (Stephen Boyd):Right.

Student: So if that’s not how you do it, then what would you do to make it.

Instructor (Stephen Boyd):No that is what you would do and that’s the correct way to treat L 1 constraints, in most cases. You do exactly what I said; you add new variables, new constraints and then, use these types of methods. So okay, we’ll quit here.

End of Audio]

Duration: 79 minutes